ACM - Amiga C Manual

Anders Bjerin

Book Two

Part IV: Appendices

http://aminet.net/package/dev/c/ACM

The complete boiled-down C manual for the Amiga which describes how to open and work with Screens, Windows, Graphics, Gadgets, Requesters, Alerts, Menus, IDCMP, Sprites, VSprites, AmigaDOS, Low Level Graphics Routines, Hints and Tips, etc. The manual also explains how to use your C Compiler and gives you important information about how the Amiga works and how your programs should be designed. The manual consists of 15 chapters together with more than 100 fully executable examples with source code.

A. EXAMPLES

A.1 SCREENS

Example1

This program will open a low-resolution, non-Interlaced, eight colour Custom Screen. It will display it for 30 seconds, and then close it.

```
/* Example1                                                             */
/* This program will open a low-resolution, non-Interlaced, eight colour */
/* Custom Screen. It will display it for 30 secondes, and then close it. */



/* If your program is using Intuition you should include intuition.h: */
#include <intuition/intuition.h>



struct IntuitionBase *IntuitionBase;



/* Declare a pointer to a Screen structure: */
struct Screen *my_screen;

/* Declare and initialize your NewScreen structure: */
struct NewScreen my_new_screen=
{
  0,            /* LeftEdge  Should always be 0. */
  0,            /* TopEdge   Top of the display.*/
  320,          /* Width     We are using a low-resolution screen. */
  200,          /* Height    Non-Interlaced NTSC (American) display. */
  3,            /* Depth     8 colours. */
  0,            /* DetailPen Text should be drawn with colour reg. 0 */
  1,            /* BlockPen  Blocks should be drawn with colour reg. 1 */
  NULL,         /* ViewModes No special modes. (Low-res, Non-Interlaced) */
  CUSTOMSCREEN, /* Type      Your own customized screen. */
  NULL,         /* Font      Default font. */
  "MY SCREEN",  /* Title     The screen' title. */
  NULL,         /* Gadget    Must for the moment be NULL. */
  NULL          /* BitMap    No special CustomBitMap. */
};



main()
{
  /* Before we can use the functions in the Intuition Library we need */
  /* to open it. (See chapter 0 INTRODUCTION for more information.)   */
  IntuitionBase = (struct IntuitionBase *)
    OpenLibrary( "intuition.library", 0 );

  if( IntuitionBase == NULL )
    exit(); /* Could NOT open the Intuition Library! */



  /* We will now try to open the screen: */
  my_screen = (struct Screen *) OpenScreen( &my_new_screen );

  /* The "(struct Screen *)" is not necessary but it tells the compiler */
  /* that the function OpenScreen() returns a pointer to a Screen       */
  /* structure. (See chapter 0 INTRODUCTION for more information about  */
  /* casting.)                                                          */


  /* Have we opened the screen succesfully? */
  if(my_screen == NULL)
  {
    /* Could NOT open the Screen! */

    /* Close the Intuition Library since we have opened it: */
    CloseLibrary( IntuitionBase );

    exit();
  }



  /* We have opened the screen, and everything seems to be OK. */

  /* Wait for 30 seconds: */
  Delay( 50 * 30);

  /* Delay(time) is a function which stops the process for a while.      */
  /* "time" is the number of ticks it should wait. (50 ticks per second) */



  /* We should always close the screens we have opened before we leave: */
  CloseScreen( my_screen );



  /* Close the Intuition Library since we have opened it: */
  CloseLibrary( IntuitionBase );

  /* THE END */
}
```

Example2

Same as Example1 except that the screen will be a high-resolution, Interlaced, 4 colour Custom Screen.

```
/* Example2                                                           */
/* This program will open a high-resolution, Interlaced, four colour     */
/* Custom Screen. It will display it for 30 secondes, and then close it. */



/* If your program is using Intuition you should include intuition.h: */
#include <intuition/intuition.h>

/* Since we are using an interlaced display (ViewModes = INTERLACE) we */
/* need to include the headerfile "display.h" which declares the       */
/* constant "INTERLACE".                                               */
#include <graphics/display.h>



struct IntuitionBase *IntuitionBase;



/* Declare a pointer to a Screen structure: */
struct Screen *my_screen;

/* Declare and initialize your NewScreen structure: */
struct NewScreen my_new_screen=
{
  0,                /* LeftEdge  Should always be 0. */
  0,                /* TopEdge   Top of the display.*/
  640,              /* Width     We are using a high-resolution screen. */
  400,              /* Height    Interlaced NTSC (American) display. */
  2,                /* Depth     4 colours. */
  0,                /* DetailPen Text should be drawn with colour reg. 0 */
  1,                /* BlockPen  Blocks should be drawn with colour reg. 1 */
  HIRES|INTERLACE, /* ViewModes High-resolution, Interlaced */
  CUSTOMSCREEN,     /* Type      Your own customized screen. */
  NULL,             /* Font      Default font. */
  "MY SCREEN",      /* Title     The screen' title. */
  NULL,             /* Gadget    Must for the moment be NULL. */
  NULL              /* BitMap    No special CustomBitMap. */
};



main()
{
  /* Open the Intuition Library: */
  IntuitionBase = (struct IntuitionBase *)
    OpenLibrary( "intuition.library", 0 );

  if( IntuitionBase == NULL )
    exit(); /* Could NOT open the Intuition Library! */



  /* We will now try to open the screen: */
  my_screen = (struct Screen *) OpenScreen( &my_new_screen );

  /* Have we opened the screen succesfully? */
  if(my_screen == NULL)
  {
    /* Could NOT open the Screen! */

    /* Close the Intuition Library since we have opened it: */
    CloseLibrary( IntuitionBase );

    exit();
  }



  /* We have opened the screen, and everything seems to be OK. */
  /* Wait for 30 seconds: */
  Delay( 50 * 30);



  /* We should always close the screens we have opened before we leave: */
  CloseScreen( my_screen );



  /* Close the Intuition Library since we have opened it: */
  CloseLibrary( IntuitionBase );

  /* THE END */
}
```

Example3

Same as Example1 except that we will use the TOPAZ_SIXTY Italic style as default font. (See chapter 3 GRAPHICS for more information about text styles.)

```
/* Example3                                                             */
/* This program will open a low-resolution, non-Interlaced, eight colour */
/* Custom Screen. It will use the TOPAZ_SIXTY Italic style as default    */
/* font.                                                                 */



/* If your program is using Intuition you should include intuition.h: */
#include <intuition/intuition.h>



struct IntuitionBase *IntuitionBase;



/* Declare and initialize the TextAttr structure: */
struct TextAttr my_font=
{
  "topaz.font", /* Font Name      Topaz */
  TOPAZ_SIXTY,  /* Font Height    64/32 character, 9 lines tall */
  FSF_ITALIC,   /* Style          Italic */
  FPF_ROMFONT   /* Preferences    The font exist in ROM */
};

/* Style:                                              */
/*   FS_NORMAL      : Normal style.                    */
/*   FSF_ITALIC     : Italic style.                    */
/*   FSF_BOLD       : Bold style.                      */
/*   FSF_UNDERLINED : Underlined.                      */
/*   FSF_EXTENDED   : Extended style (wider than normal) */
/* See file graphics/text.h for more information.      */



/* Declare a pointer to a Screen structure: */
struct Screen *my_screen;

/* Declare and initialize your NewScreen structure: */
struct NewScreen my_new_screen=
{
  0,            /* LeftEdge  Should always be 0. */
  0,            /* TopEdge   Top of the display.*/
  320,          /* Width     We are using a low-resolution screen. */
  200,          /* Height    Non-Interlaced NTSC (American) display. */
  3,            /* Depth     8 colours. */
  0,            /* DetailPen Text should be drawn with colour reg. 0 */
  1,            /* BlockPen  Blocks should be drawn with colour reg. 1 */
  NULL,         /* ViewModes No special modes. (Low-res, Non-Interlaced) */
  CUSTOMSCREEN, /* Type      Your own customized screen. */
  &my_font,     /* Font      Topaz 60 (Italic style) font. */
  "MY SCREEN",  /* Title     The screen' title. */
  NULL,         /* Gadget    Must for the moment be NULL. */
  NULL          /* BitMap    No special CustomBitMap. */
};
```

```
main()
{
  /* Open the Intuition Library: */
  IntuitionBase = (struct IntuitionBase *)
    OpenLibrary( "intuition.library", 0 );

  if( IntuitionBase == NULL )
    exit(); /* Could NOT open the Intuition Library! */



  /* We will now try to open the screen: */
  my_screen = (struct Screen *) OpenScreen( &my_new_screen );

  /* The "(struct Screen *)" is not necessary but it tells the compiler */
  /* that the function OpenScreen() returns a pointer to a Screen */
  /* structure. (See chapter "Amiga C" for more information) */

  /* Have we opened the screen succesfully? */
  if(my_screen == NULL)
  {
    /* Could NOT open the Screen! */

    /* Close the Intuition Library since we have opened it: */
    CloseLibrary( IntuitionBase );

    exit();
  }



  /* We have opened the screen, and everything seems to be OK. */
  /* Wait for 30 seconds: */
  Delay( 50 * 30);



  /* We should always close the screens we have opened before we leave: */
  CloseScreen( my_screen );



  /* Close the Intuition Library since we have opened it: */
  CloseLibrary( IntuitionBase );

  /* THE END */
}
```

Example4

This program will open two screens, one (low-resolution 32 colours) at the top of the display, and one (high-resolution 16 colours) a bit down.

```
/* Example4                                                         */
/* This program will open two screens, one (low-resolution 32 colours) */
/* at the top of the display, and the other one (high-resolution 16    */
/* colours) a bit further down.                                        */



/* If your program is using Intuition you should include intuition.h: */
#include <intuition/intuition.h>


struct IntuitionBase *IntuitionBase;


/* Declare two pointer to a Screen structure: */
struct Screen *my_screen1;
struct Screen *my_screen2;

/* Declare and initialize your NewScreen structure for screen 1: */
struct NewScreen my_new_screen1=
{
  0,            /* LeftEdge  Should always be 0. */
  0,            /* TopEdge   Top of the display.*/
  320,          /* Width     We are using a low-resolution screen. */
  100,          /* Height    */
  5,            /* Depth     32 colours. */
  0,            /* DetailPen Text should be drawn with colour reg. 0 */
  1,            /* BlockPen  Blocks should be drawn with colour reg. 1 */
  NULL,         /* ViewModes No special modes. (Low-res, Non-Interlaced) */
  CUSTOMSCREEN, /* Type      Your own customized screen. */
  NULL,         /* Font      Default font. */
  "MY SCREEN1", /* Title     The screen' title. */
  NULL,         /* Gadget    Must for the moment be NULL. */
  NULL          /* BitMap    No special CustomBitMap. */
};

/* Declare and initialize your NewScreen structure for screen 2: */
struct NewScreen my_new_screen2=
{
  0,            /* LeftEdge  Should always be 0. */
  105,          /* TopEdge   Top of the display.*/
  640,          /* Width     We are using a low-resolution screen. */
  95,           /* Height    */
  4,            /* Depth     16 colours. */
  0,            /* DetailPen Text should be drawn with colour reg. 0 */
  1,            /* BlockPen  Blocks should be drawn with colour reg. 1 */
  HIRES,        /* ViewModes High-resolution, Non-Interlaced */
  CUSTOMSCREEN, /* Type      Your own customized screen. */
  NULL,         /* Font      Default font. */
  "MY SCREEN2", /* Title     The screen' title. */
  NULL,         /* Gadget    Must for the moment be NULL. */
  NULL          /* BitMap    No special CustomBitMap. */
};



main()
{
  /* Open the Intuition Library: */
  IntuitionBase = (struct IntuitionBase *)
    OpenLibrary( "intuition.library", 0 );

  if( IntuitionBase == NULL )
    exit(); /* Could NOT open the Intuition Library! */


  /* We will now try to open the first screen: */
  my_screen1 = (struct Screen *) OpenScreen( &my_new_screen1 );

  /* Have we opened screen1 succesfully? */
  if(my_screen1 == NULL)
  {
    /* Could NOT open the Screen1! */

    /* Close the Intuition Library since we have opened it: */
    CloseLibrary( IntuitionBase );

    exit();
  }


  /* We will now try to open the second screen: */
  my_screen2 = (struct Screen *) OpenScreen( &my_new_screen2 );

  /* Have we opened screen2 succesfully? */
  if(my_screen2 == NULL)
  {
    /* Could NOT open Screen2! */

    /* Close Screen1 before we leave since we have opened it: */
    CloseScreen( my_screen1 );

    /* Close the Intuition Library since we have opened it: */
    CloseLibrary( IntuitionBase );

    exit();
  }


  /* We have opened the screens, and everything seems to be OK. */
  /* Wait for 30 seconds: */
  Delay( 50 * 30);


  /* We should always close the screens we have opened before we leave: */
  CloseScreen( my_screen2 );
  CloseScreen( my_screen1 );


  /* Close the Intuition Library since we have opened it: */
  CloseLibrary( IntuitionBase );

  /* THE END */
}
```

Example5

Same as Example4 except that after 10 seconds the low-resolution screen will move down 75 lines. After another 10 seconds it will be put in front of all other screens. 10 seconds later it will move down another 75 lines. The program will wait 10 seconds before the screens are closed and the program exits.

```
/* Example5                                                           */
/* This program will open two screens, one (low-resolution 32 colours) */
/* at the top of the display, and the other one (high-resolution 16    */
/* colours) a bit further down. After 10 seconds the low-resolution    */
/* screen will move down 75 lines. After another 10 seconds it will be */
/* put in front of all other screens. 10 seconds later it will move    */
/* down another 75 lines. The program will wait 10 seconds before the  */
/* screens are closed and the program exits.                           */


/* If your program is using Intuition you should include intuition.h: */
#include <intuition/intuition.h>


struct IntuitionBase *IntuitionBase;


/* Declare two pointer to a Screen structure: */
struct Screen *my_screen1;
struct Screen *my_screen2;

/* Declare and initialize your NewScreen structure for screen 1: */
struct NewScreen my_new_screen1=
{
  0,            /* LeftEdge  Should always be 0. */
  0,            /* TopEdge   Top of the display.*/
  320,          /* Width     We are using a low-resolution screen. */
  100,          /* Height    */
  5,            /* Depth     32 colours. */
  0,            /* DetailPen Text should be drawn with colour reg. 0 */
  1,            /* BlockPen  Blocks should be drawn with colour reg. 1 */
  NULL,         /* ViewModes No special modes. (Low-res, Non-Interlaced) */
  CUSTOMSCREEN, /* Type      Your own customized screen. */
  NULL,         /* Font      Default font. */
  "MY SCREEN1", /* Title     The screen' title. */
  NULL,         /* Gadget    Must for the moment be NULL. */
  NULL          /* BitMap    No special CustomBitMap. */
};

/* Declare and initialize your NewScreen structure for screen 2: */
struct NewScreen my_new_screen2=
{
  0,            /* LeftEdge  Should always be 0. */
  105,          /* TopEdge   Top of the display.*/
  640,          /* Width     We are using a low-resolution screen. */
  95,           /* Height    */
  4,            /* Depth     16 colours. */
  0,            /* DetailPen Text should be drawn with colour reg. 0 */
  1,            /* BlockPen  Blocks should be drawn with colour reg. 1 */
  HIRES,        /* ViewModes High-resolution, Non-Interlaced */
  CUSTOMSCREEN, /* Type      Your own customized screen. */
  NULL,         /* Font      Default font. */
  "MY SCREEN2", /* Title     The screen' title. */
  NULL,         /* Gadget    Must for the moment be NULL. */
  NULL          /* BitMap    No special CustomBitMap. */
};
```

```
main()
{
  /* Open the Intuition Library: */
  IntuitionBase = (struct IntuitionBase *)
    OpenLibrary( "intuition.library", 0 );

  if( IntuitionBase == NULL )
    exit(); /* Could NOT open the Intuition Library! */


  /* We will now try to open the first screen: */
  my_screen1 = (struct Screen *) OpenScreen( &my_new_screen1 );

  /* Have we opened screen1 succesfully? */
  if(my_screen1 == NULL)
  {
    /* Could NOT open the Screen1! */

    /* Close the Intuition Library since we have opened it: */
    CloseLibrary( IntuitionBase );

    exit();
  }


  /* We will now try to open the second screen: */
  my_screen2 = (struct Screen *) OpenScreen( &my_new_screen2 );

  /* Have we opened screen2 succesfully? */
  if(my_screen2 == NULL)
  {
    /* Could NOT open Screen2! */

    /* Close Screen1 before we leave since we have opened it: */
    CloseScreen( my_screen1 );

    /* Close the Intuition Library since we have opened it: */
    CloseLibrary( IntuitionBase );

    exit();
  }


  /* We have opened the screens, and everything seems to be OK. */

  /* Wait for 10 seconds: */
  Delay( 50 * 10);

  /* Move the low-resolution screen down 75 lines: */
  MoveScreen( my_screen1, 0, 75 );

  /* Wait for 10 seconds: */
  Delay( 50 * 10);

  /* Put the low-resolution screen in front of all other screens: */
  ScreenToFront( my_screen1 );

  /* Wait for 10 seconds: */
  Delay( 50 * 10);

  /* Move the low-resolution screen down another 75 lines: */
  MoveScreen( my_screen1, 0, 75 );

  /* Wait for 10 seconds: */
  Delay( 50 * 10);


  /* We should always close the screens we have opened before we leave: */
  CloseScreen( my_screen2 );
  CloseScreen( my_screen1 );



  /* Close the Intuition Library since we have opened it: */
  CloseLibrary( IntuitionBase );

  /* THE END */
}
```

Example6

This program will open a low-resolution, non-Interlaced, 4 colour Custom Screen. It will after 5 seconds start to change the screens colours, and will after a while close the screen and exit.

```
/* Example6                                                           */
/* This program will open a low-resolution, non-Interlaced, 4 colour     */
/* Custom Screen. It will after 5 secondes start to change the screens   */
/* colours, and will after a while close the screen and exit.            */



/* If your program is using Intuition you should include intuition.h: */
#include <intuition/intuition.h>



struct IntuitionBase *IntuitionBase;
struct GfxBase *GfxBase;


/* Declare a pointer to a Screen structure: */
struct Screen *my_screen;

/* Declare and initialize your NewScreen structure: */
struct NewScreen my_new_screen=
{
  0,            /* LeftEdge  Should always be 0. */
  0,            /* TopEdge   Top of the display.*/
  320,          /* Width     We are using a low-resolution screen. */
  200,          /* Height    Non-Interlaced NTSC (American) display. */
  2,            /* Depth     4 colours. */
  0,            /* DetailPen Text should be drawn with colour reg. 0 */
  1,            /* BlockPen  Blocks should be drawn with colour reg. 1 */
  NULL,         /* ViewModes No special modes. (Low-res, Non-Interlaced) */
  CUSTOMSCREEN, /* Type      Your own customized screen. */
  NULL,         /* Font      Default font. */
  "MY SCREEN",  /* Title     The screen' title. */
  NULL,         /* Gadget    Must for the moment be NULL. */
  NULL          /* BitMap    No special CustomBitMap. */
};



main()
{
  /* Open the Intuition Library: */
  IntuitionBase = (struct IntuitionBase *)
    OpenLibrary( "intuition.library", 0 );

  if( IntuitionBase == NULL )
    exit(); /* Could NOT open the Intuition Library! */



  /* Before we can use the function SetRGB4() we need to open the */
  /* graphics Library. (See chapter 0 INTRODUCTION for more       */
  /* information.)                                                */
  GfxBase = (struct GfxBase *)
    OpenLibrary( "graphics.library", 0);

  if( GfxBase == NULL )
```

```
{
  /* Could NOT open the Graphics Library! */

  /* Close the Intuition Library since we have opened it: */
  CloseLibrary( IntuitionBase );

  exit();
}



/* We will now try to open the screen: */
my_screen = (struct Screen *) OpenScreen( &my_new_screen );

/* The "(struct Screen *)" is not necessary but it tells the compiler */
/* that the function OpenScreen() returns a pointer to a Screen */
/* structure. (See chapter "Amiga C" for more information) */

/* Have we opened the screen succesfully? */
if(my_screen == NULL)
{
  /* Could NOT open the Screen! */

  /* Close the Graphics Library since we have opened it: */
  CloseLibrary( GfxBase );

  /* Close the Intuition Library since we have opened it: */
  CloseLibrary( IntuitionBase );

  exit();
}



/* We have opened the screen, and everything seems to be OK. */

/* Wait for 5 seconds: */
Delay( 50 * 5);



/* Change colour register 1 to red: */
SetRGB4( &my_screen->ViewPort, 1, 15, 0, 0 );

/* Wait for 1 second: */
Delay( 50 * 1);



/* Change colour register 1 to green: */
SetRGB4( &my_screen->ViewPort, 1, 0, 15, 0 );

/* Wait for 1 second: */
Delay( 50 * 1);
```

```
  /* Change colour register 1 to blue: */
  SetRGB4( &my_screen->ViewPort, 1, 0, 0, 15 );

  /* Wait for 1 second: */
  Delay( 50 * 1);



  /* Change colour register 1 to white: */
  SetRGB4( &my_screen->ViewPort, 1, 15, 15, 15 );

  /* Wait for 1 second: */
  Delay( 50 * 1);



  /* Change colour register 0 to black: */
  SetRGB4( &my_screen->ViewPort, 0, 0, 0, 0 );

  /* Wait for 1 second: */
  Delay( 50 * 1);



  /* Wait for 5 seconds: */
  Delay( 50 * 5);



  /* We should always close the screens we have opened before we leave: */
  CloseScreen( my_screen );



  /* Close the Graphics Library since we have opened it: */
  CloseLibrary( GfxBase );

  /* Close the Intuition Library since we have opened it: */
  CloseLibrary( IntuitionBase );

  /* THE END */
}
```

A.2 WINDOWS

Example1

This program will open a normal window which is connected to the Workbench Screen. It will display it for 30 seconds, and then close it.

```c
/* Example1                                                     */
/* This program will open a normal window which is connected to the    */
/* Workbench Screen. It will display it for 30 seconds, and then close */
/* it.                                                          */



/* If your program is using Intuition you should include intuition.h: */
#include <intuition/intuition.h>



struct IntuitionBase *IntuitionBase;



/* Declare a pointer to a Window structure: */
struct Window *my_window;

/* Declare and initialize your NewWindow structure: */
struct NewWindow my_new_window=
{
  50,            /* LeftEdge    x position of the window. */
  25,            /* TopEdge     y positio of the window. */
  150,           /* Width       150 pixels wide. */
  100,           /* Height      100 lines high. */
  0,             /* DetailPen   Text should be drawn with colour reg. 0 */
  1,             /* BlockPen    Blocks should be drawn with colour reg. 1 */
  NULL,          /* IDCMPFlags  No IDCMP flags. */
  SMART_REFRESH, /* Flags       Intuition should refresh the window. */
  NULL,          /* FirstGadget No Custom Gadgets. */
  NULL,          /* CheckMark   Use Intuition's default CheckMark (v). */
  "MY WINDOW",   /* Title       Title of the window. */
  NULL,          /* Screen      Connected to the Workbench Screen. */
  NULL,          /* BitMap      No Custom BitMap. */
  0,             /* MinWidth    We do not need to care about these */
  0,             /* MinHeight   since we havent supplied the window with */
  0,             /* MaxWidth    a Sizing Gadget. */
  0,             /* MaxHeight */
  WBENCHSCREEN   /* Type        Connected to the Workbench Screen. */
};



main()
{
  /* Open the Intuition Library: */
  IntuitionBase = (struct IntuitionBase *)
    OpenLibrary( "intuition.library", 0 );

  if( IntuitionBase == NULL )
    exit(); /* Could NOT open the Intuition Library! */



  /* We will now try to open the window: */
  my_window = (struct Window *) OpenWindow( &my_new_window );
```

```c

  /* The "(struct Window *)" is not necessary but it tells the compiler */
  /* that the function OpenWindow() returns a pointer to a Window       */
  /* structure. (See chapter 0 INTRODUCTION for more information about  */
  /* "casting".)                                                        */

  /* Have we opened the window succesfully? */
  if(my_window == NULL)
  {
    /* Could NOT open the Window! */

    /* Close the Intuition Library since we have opened it: */
    CloseLibrary( IntuitionBase );

    exit();
  }



  /* We have opened the window, and everything seems to be OK. */
  /* Wait for 30 seconds: */
  Delay( 50 * 30);



  /* We should always close the windows we have opened before we leave: */
  CloseWindow( my_window );



  /* Close the Intuition Library since we have opened it: */
  CloseLibrary( IntuitionBase );

  /* THE END */
}
```

Example2

This program will open a high resolution 16 colour Custom Screen and a normal window which is connected to it. It will display it for 30 seconds, and then close the Custom Screen and the window.

```
/* Example2                                                     */
/* This program will open a high resolution 16 colour Custom Screen */
/* and a normal window which is connected to it. It will display it */
/* for 30 seconds, and then close the Custom Screen and the window. */



/* If your program is using Intuition you should include intuition.h: */
#include <intuition/intuition.h>



struct IntuitionBase *IntuitionBase;



/* Declare a pointer to a Screen structure: */
struct Screen *my_screen;

/* Declare and initialize your NewScreen structure: */
struct NewScreen my_new_screen=
{
  0,            /* LeftEdge  Should always be 0. */
  0,            /* TopEdge   Top of the display.*/
  640,          /* Width     We are using a high-resolution screen. */
  200,          /* Height    Non-Interlaced NTSC (American) display. */
  4,            /* Depth     16 colours. */
  0,            /* DetailPen Text should be drawn with colour reg. 0 */
  1,            /* BlockPen  Blocks should be drawn with colour reg. 1 */
  HIRES,        /* ViewModes High-resolution. (Non-Interlaced) */
  CUSTOMSCREEN, /* Type      Your own customized screen. */
  NULL,         /* Font      Default font. */
  "MY SCREEN",  /* Title     The screen' title. */
  NULL,         /* Gadget    Must for the moment be NULL. */
  NULL          /* BitMap    No special CustomBitMap. */
};




/* Declare a pointer to a Window structure: */
struct Window *my_window;

/* Declare and initialize your NewWindow structure: */
struct NewWindow my_new_window=
{
  50,            /* LeftEdge    x position of the window. */
  25,            /* TopEdge     y positio of the window. */
  150,           /* Width       150 pixels wide. */
  100,           /* Height      100 lines high. */
  0,             /* DetailPen   Text should be drawn with colour reg. 0 */
  1,             /* BlockPen    Blocks should be drawn with colour reg. 1 */
  NULL,          /* IDCMPFlags  No IDCMP flags. */
  SMART_REFRESH, /* Flags       Intuition should refresh the window. */
  NULL,          /* FirstGadget No Custom Gadgets. */
  NULL,          /* CheckMark   Use Intuition's default CheckMark (v). */
  "MY WINDOW",   /* Title       Title of the window. */
  NULL,          /* Screen      We will later connect it to the screen. */
  NULL,          /* BitMap      No Custom BitMap. */
  0,             /* MinWidth    We do not need to care about these */
  0,             /* MinHeight   since we havent supplied the window with */
  0,             /* MaxWidth    a Sizing Gadget. */
  0,             /* MaxHeight */
  CUSTOMSCREEN   /* Type        Connected to the Workbench Screen. */
};



main()
{
  /* Open the Intuition Library: */
  IntuitionBase = (struct IntuitionBase *)
    OpenLibrary( "intuition.library", 0 );

  if( IntuitionBase == NULL )
    exit(); /* Could NOT open the Intuition Library! */



  /* We will now try to open the screen: */
  my_screen = (struct Screen *) OpenScreen( &my_new_screen );

  /* Have we opened the screen succesfully? */
  if(my_screen == NULL)
  {
    /* Could NOT open the Screen! */

    /* Close the Intuition Library since we have opened it: */
    CloseLibrary( IntuitionBase );

    exit();
  }



  /* Before we can open the window we need to give the NewWindow */
  /* structure a pointer to the opened Custom Screen: */
  my_new_window.Screen = my_screen;



  /* We will now try to open the window: */
  my_window = (struct Window *) OpenWindow( &my_new_window );

  /* Have we opened the window succesfully? */
  if(my_window == NULL)
  {
    /* Could NOT open the Window! */

    /* Close the screen since we have opened it: */
    CloseScreen( my_screen );

    /* Close the Intuition Library since we have opened it: */
    CloseLibrary( IntuitionBase );
    exit();
  }



  /* We have opened the window, and everything seems to be OK. */
  /* Wait for 30 seconds: */
  Delay( 50 * 30);



  /* We should always close what we have opened: */
  CloseWindow( my_window );

  /* Remember that all windows connected to a screen must be closed */
  /* before you may close the screen! */
  CloseScreen( my_screen );



  /* Close the Intuition Library since we have opened it: */
  CloseLibrary( IntuitionBase );

  /* THE END */
}
```

Example3

This program will open a normal window which is connected to the Workbench Screen. The window will use all System Gadgets, and will automatically Activate the window. It will display it for 30 seconds, and then close it. (Remember that the Close Gadget does NOT close the window by itself, it will only inform you that the user wants to close it. But in this example we will not listen to what the user wants.)

```
/* Example3                                                        */
/* This program will open a normal window which is connected to the      */
/* Workbench Screen. The window will use all System Gadgets, and will    */
/* automatically Activate the window. It will display it for 30 seconds, */
/* and then close it. (Remember that the Close Gadget does NOT close the */
/* window by itself, it will only inform you that the user wants to      */
/* close it. But in this example we will not listen to what the user     */
/* wants.)                                                               */



/* If your program is using Intuition you should include intuition.h: */
#include <intuition/intuition.h>



struct IntuitionBase *IntuitionBase;



/* Declare a pointer to a Window structure: */
struct Window *my_window;

/* Declare and initialize your NewWindow structure: */
struct NewWindow my_new_window=
{
  50,            /* LeftEdge    x position of the window. */
  25,            /* TopEdge     y positio of the window. */
  200,           /* Width       200 pixels wide. */
  100,           /* Height      100 lines high. */
  0,             /* DetailPen   Text should be drawn with colour reg. 0 */
  1,             /* BlockPen    Blocks should be drawn with colour reg. 1 */
  NULL,          /* IDCMPFlags  No IDCMP flags. */
  SMART_REFRESH| /* Flags       Intuition should refresh the window. */
  WINDOWCLOSE|   /*             Close Gadget. */
  WINDOWDRAG|    /*             Drag gadget. */
  WINDOWDEPTH|   /*             Depth arrange Gadgets. */
  WINDOWSIZING|  /*             Sizing Gadget. */
  ACTIVATE,      /*             The window should be Active when opened. */
  NULL,          /* FirstGadget No Custom Gadgets. */
  NULL,          /* CheckMark   Use Intuition's default CheckMark (v). */
  "MY WINDOW",   /* Title       Title of the window. */
  NULL,          /* Screen      Connected to the Workbench Screen. */
  NULL,          /* BitMap      No Custom BitMap. */
  80,            /* MinWidth    We will not allow the window to become */
  30,            /* MinHeight   smaller than 80 x 30, and not bigger */
  300,           /* MaxWidth    than 300 x 200. */
  200,           /* MaxHeight */
  WBENCHSCREEN   /* Type        Connected to the Workbench Screen. */
};



main()
{
  /* Open the Intuition Library: */
  IntuitionBase = (struct IntuitionBase *)
    OpenLibrary( "intuition.library", 0 );

  if( IntuitionBase == NULL )
    exit(); /* Could NOT open the Intuition Library! */



  /* We will now try to open the window: */
  my_window = (struct Window *) OpenWindow( &my_new_window );

  /* Have we opened the window succesfully? */
  if(my_window == NULL)
  {
    /* Could NOT open the Window! */

    /* Close the Intuition Library since we have opened it: */
    CloseLibrary( IntuitionBase );

    exit();
  }



  /* We have opened the window, and everything seems to be OK. */
  /* Wait for 30 seconds: */
  Delay( 50 * 30);



  /* We should always close the windows we have opened before we leave: */
  CloseWindow( my_window );



  /* Close the Intuition Library since we have opened it: */
  CloseLibrary( IntuitionBase );

  /* THE END */
}
```

Example4

This program will open two normal windows which are connected to the Workbench Screen. The windows will use all System Gadgets. It will display them for 30 seconds, and then close them.

```
/* Example4                                                         */
/* This program will open two normal windows which are connected to the  */
/* Workbench Screen. The windows will use all System Gadgets. It will    */
/* display them for 30 seconds, and then close them.                     */



/* If your program is using Intuition you should include intuition.h: */
#include <intuition/intuition.h>



struct IntuitionBase *IntuitionBase;



/* Declare a pointer to Window structure number one: */
struct Window *my_window1;

/* Declare and initialize your NewWindow structure number one: */
struct NewWindow my_new_window1=
{
  50,            /* LeftEdge    x position of the window. */
  25,            /* TopEdge     y positio of the window. */
  200,           /* Width       200 pixels wide. */
  100,           /* Height      100 lines high. */
  0,             /* DetailPen   Text should be drawn with colour reg. 0 */
  1,             /* BlockPen    Blocks should be drawn with colour reg. 1 */
  NULL,          /* IDCMPFlags  No IDCMP flags. */
  SMART_REFRESH| /* Flags       Intuition should refresh the window. */
  WINDOWCLOSE|   /*             Close Gadget. */
  WINDOWDRAG|    /*             Drag gadget. */
  WINDOWDEPTH|   /*             Depth arrange Gadgets. */
  WINDOWSIZING,  /*             Sizing Gadget. */
  NULL,          /* FirstGadget No Custom Gadgets. */
  NULL,          /* CheckMark   Use Intuition's default CheckMark (v). */
  "MY WINDOW 1", /* Title       Title of the window. */
  NULL,          /* Screen      Connected to the Workbench Screen. */
  NULL,          /* BitMap      No Custom BitMap. */
  80,            /* MinWidth    We will not allow the window to become */
  30,            /* MinHeight   smaller than 80 x 30, and not bigger */
  300,           /* MaxWidth    than 300 x 200. */
  200,           /* MaxHeight */
  WBENCHSCREEN   /* Type        Connected to the Workbench Screen. */
};



/* Declare a pointer to Window structure number two: */
struct Window *my_window2;

/* Declare and initialize your NewWindow structure number two: */
struct NewWindow my_new_window2=
{
  300,           /* LeftEdge    x position of the window. */
  50,            /* TopEdge     y positio of the window. */
  200,           /* Width       200 pixels wide. */
  100,           /* Height      100 lines high. */
  0,             /* DetailPen   Text should be drawn with colour reg. 0 */
  1,             /* BlockPen    Blocks should be drawn with colour reg. 1 */
  NULL,          /* IDCMPFlags  No IDCMP flags. */
  SMART_REFRESH| /* Flags       Intuition should refresh the window. */
  WINDOWCLOSE|   /*             Close Gadget. */
  WINDOWDRAG|    /*             Drag gadget. */
  WINDOWDEPTH|   /*             Depth arrange Gadgets. */
  WINDOWSIZING|  /*             Sizing Gadget. */
  ACTIVATE,      /*             The window should be Active when opened. */
  NULL,          /* FirstGadget No Custom Gadgets. */
  NULL,          /* CheckMark   Use Intuition's default CheckMark (v). */
  "MY WINDOW 2", /* Title       Title of the window. */
  NULL,          /* Screen      Connected to the Workbench Screen. */
  NULL,          /* BitMap      No Custom BitMap. */
  80,            /* MinWidth    We will not allow the window to become */
  30,            /* MinHeight   smaller than 80 x 30, and not bigger */
  0,             /* MaxWidth    than the default sixe (200x100). */
  0,             /* MaxHeight */
  WBENCHSCREEN   /* Type        Connected to the Workbench Screen. */
};



main()
{
  /* Open the Intuition Library: */
  IntuitionBase = (struct IntuitionBase *)
    OpenLibrary( "intuition.library", 0 );

  if( IntuitionBase == NULL )
    exit(); /* Could NOT open the Intuition Library! */



  /* We will now try to open the first window: */
  my_window1 = (struct Window *) OpenWindow( &my_new_window1 );

  /* Have we opened the first window succesfully? */
  if(my_window1 == NULL)
  {
    /* Could NOT open the first Window! */

    /* Close the Intuition Library since we have opened it: */
    CloseLibrary( IntuitionBase );

    exit();
  }



  /* We will now try to open the second window: */
  my_window2 = (struct Window *) OpenWindow( &my_new_window2 );

  /* Have we opened the second window succesfully? */
  if(my_window2 == NULL)
  {
    /* Could NOT open the second Window! */

    /* We must close the first window since we have opened it: */
    CloseWindow( my_window1 );

    /* Close the Intuition Library since we have opened it: */
    CloseLibrary( IntuitionBase );

    exit();
  }



  /* We have opened the windows, and everything seems to be OK. */
  /* Wait for 30 seconds: */
  Delay( 50 * 30);



  /* We should always close the windows we have opened before we leave: */
  CloseWindow( my_window1 );
  CloseWindow( my_window2 );



  /* Close the Intuition Library since we have opened it: */
  CloseLibrary( IntuitionBase );

  /* THE END */
}
```

Example5

This program will open a Borderless window which is connected to the Workbench Screen. It will display it for 30 seconds, and then quit.

```
/* Example5                                                         */
/* This program will open a Borderless window which is connected to the */
/* Workbench Screen. It will display it for 30 seconds, and then quit.  */



/* If your program is using Intuition you should include intuition.h: */
#include <intuition/intuition.h>



struct IntuitionBase *IntuitionBase;



/* Declare a pointer to a Window structure: */
struct Window *my_window;

/* Declare and initialize your NewWindow structure: */
struct NewWindow my_new_window=
{
  50,            /* LeftEdge    x position of the window. */
  25,            /* TopEdge     y positio of the window. */
  200,           /* Width       200 pixels wide. */
  100,           /* Height      100 lines high. */
  0,             /* DetailPen   Text should be drawn with colour reg. 0 */
  1,             /* BlockPen    Blocks should be drawn with colour reg. 1 */
  NULL,          /* IDCMPFlags  No IDCMP flags. */
  SMART_REFRESH| /* Flags       Intuition should refresh the window. */
  BORDERLESS|    /*             No borders. */
  ACTIVATE,      /*             The window should be Active when opened. */
  NULL,          /* FirstGadget No Custom Gadgets. */
  NULL,          /* CheckMark   Use Intuition's default CheckMark (v). */
  "MY WINDOW",   /* Title       Title of the window. */
  NULL,          /* Screen      Connected to the Workbench Screen. */
  NULL,          /* BitMap      No Custom BitMap. */
  0,             /* MinWidth    We do not need to care about these */
  0,             /* MinHeight   since we havent supplied the window with */
  0,             /* MaxWidth    a Sizing Gadget. */
  0,             /* MaxHeight */
  WBENCHSCREEN   /* Type        Connected to the Workbench Screen. */
};



main()
{
  /* Open the Intuition Library: */
  IntuitionBase = (struct IntuitionBase *)
    OpenLibrary( "intuition.library", 0 );

  if( IntuitionBase == NULL )
    exit(); /* Could NOT open the Intuition Library! */



  /* We will now try to open the window: */
  my_window = (struct Window *) OpenWindow( &my_new_window );

  /* Have we opened the window succesfully? */
  if(my_window == NULL)
  {
    /* Could NOT open the Window! */

    /* Close the Intuition Library since we have opened it: */
    CloseLibrary( IntuitionBase );

    exit();
  }



  /* We have opened the window, and everything seems to be OK. */
  /* Wait for 30 seconds: */
  Delay( 50 * 30);



  /* We should always close the windows we have opened before we leave: */
  CloseWindow( my_window );



  /* Close the Intuition Library since we have opened it: */
  CloseLibrary( IntuitionBase );

  /* THE END */
}
```

Example6

Same as Example5 except that the window will also use all System Gadgets.

```c
/* Example6                                                  */
/* This program will open a Borderless window which is connected to the */
/* Workbench Screen. It will use all System Gadgets and display it for  */
/* 30 seconds, and then quit.                                */



/* If your program is using Intuition you should include intuition.h: */
#include <intuition/intuition.h>



struct IntuitionBase *IntuitionBase;



/* Declare a pointer to a Window structure: */
struct Window *my_window;

/* Declare and initialize your NewWindow structure: */
struct NewWindow my_new_window=
{
  50,            /* LeftEdge    x position of the window. */
  25,            /* TopEdge     y positio of the window. */
  200,           /* Width       200 pixels wide. */
  100,           /* Height      100 lines high. */
  0,             /* DetailPen   Text should be drawn with colour reg. 0 */
  1,             /* BlockPen    Blocks should be drawn with colour reg. 1 */
  NULL,          /* IDCMPFlags  No IDCMP flags. */
  SMART_REFRESH| /* Flags       Intuition should refresh the window. */
  BORDERLESS|    /*             No borders. */
  WINDOWCLOSE|   /*             Close Gadget. */
  WINDOWDRAG|    /*             Drag gadget. */
  WINDOWDEPTH|   /*             Depth arrange Gadgets. */
  WINDOWSIZING|  /*             Sizing Gadget. */
  ACTIVATE,      /*             The window should be Active when opened. */
  NULL,          /* FirstGadget No Custom Gadgets. */
  NULL,          /* CheckMark   Use Intuition's default CheckMark (v). */
  "MY WINDOW",   /* Title       Title of the window. */
  NULL,          /* Screen      Connected to the Workbench Screen. */
  NULL,          /* BitMap      No Custom BitMap. */
  80,            /* MinWidth    We will not allow the window to become */
  30,            /* MinHeight   smaller than 80 x 30, and not bigger */
  300,           /* MaxWidth    than 300 x 200. */
  200,           /* MaxHeight */
  WBENCHSCREEN   /* Type        Connected to the Workbench Screen. */
};



main()
{
  /* Open the Intuition Library: */
  IntuitionBase = (struct IntuitionBase *)
    OpenLibrary( "intuition.library", 0 );

  if( IntuitionBase == NULL )
    exit(); /* Could NOT open the Intuition Library! */



  /* We will now try to open the window: */
  my_window = (struct Window *) OpenWindow( &my_new_window );

  /* Have we opened the window succesfully? */
  if(my_window == NULL)
  {
    /* Could NOT open the Window! */

    /* Close the Intuition Library since we have opened it: */
    CloseLibrary( IntuitionBase );

    exit();
  }



  /* We have opened the window, and everything seems to be OK. */
  /* Wait for 30 seconds: */
  Delay( 50 * 30);



  /* We should always close the windows we have opened before we leave: */
  CloseWindow( my_window );



  /* Close the Intuition Library since we have opened it: */
  CloseLibrary( IntuitionBase );

  /* THE END */
}
```

Example7

This program will open three windows, two are normal and the third is a Backdrop window. The windows will use all System Gadgets, except the Backdrop window, which only can use the close-window gadget. After 30 seconds the program quits. (Try to push either window 1 or 2 behind the Backdrop window.)

```
/* Example7                                                          */
/* This program will open three windows, two are normal and the third is */
/* a Backdrop window. The windows will use all System Gadgets, except    */
/* the Backdrop window, which only can use the close-window gadget.      */
/* After 30 seconds the program quits.                                   */



/* If your program is using Intuition you should include intuition.h: */
#include <intuition/intuition.h>



struct IntuitionBase *IntuitionBase;



/* Declare a pointer to Window structure number one: */
struct Window *my_window1;

/* Declare and initialize your NewWindow structure number one: */
struct NewWindow my_new_window1=
{
  50,            /* LeftEdge    x position of the window. */
  25,            /* TopEdge     y positio of the window. */
  200,           /* Width       200 pixels wide. */
  100,           /* Height      100 lines high. */
  0,             /* DetailPen   Text should be drawn with colour reg. 0 */
  1,             /* BlockPen    Blocks should be drawn with colour reg. 1 */
  NULL,          /* IDCMPFlags  No IDCMP flags. */
  SMART_REFRESH| /* Flags       Intuition should refresh the window. */
  WINDOWCLOSE|   /*             Close Gadget. */
  WINDOWDRAG|    /*             Drag gadget. */
  WINDOWDEPTH|   /*             Depth arrange Gadgets. */
  WINDOWSIZING,  /*             Sizing Gadget. */
  NULL,          /* FirstGadget No Custom Gadgets. */
  NULL,          /* CheckMark   Use Intuition's default CheckMark (v). */
  "MY WINDOW 1", /* Title       Title of the window. */
  NULL,          /* Screen      Connected to the Workbench Screen. */
  NULL,          /* BitMap      No Custom BitMap. */
  80,            /* MinWidth    We will not allow the window to become */
  30,            /* MinHeight   smaller than 80 x 30, and not bigger */
  300,           /* MaxWidth    than 300 x 200. */
  200,           /* MaxHeight */
  WBENCHSCREEN   /* Type        Connected to the Workbench Screen. */
};



/* Declare a pointer to Window structure number two: */
struct Window *my_window2;

/* Declare and initialize your NewWindow structure number two: */
struct NewWindow my_new_window2=
{
  300,           /* LeftEdge    x position of the window. */
  50,            /* TopEdge     y positio of the window. */
```

```
  200,           /* Width       200 pixels wide. */
  100,           /* Height      100 lines high. */
  0,             /* DetailPen   Text should be drawn with colour reg. 0 */
  1,             /* BlockPen    Blocks should be drawn with colour reg. 1 */
  NULL,          /* IDCMPFlags  No IDCMP flags. */
  SMART_REFRESH| /* Flags       Intuition should refresh the window. */
  WINDOWCLOSE|   /*             Close Gadget. */
  WINDOWDRAG|    /*             Drag gadget. */
  WINDOWDEPTH|   /*             Depth arrange Gadgets. */
  WINDOWSIZING|  /*             Sizing Gadget. */
  ACTIVATE,      /*             The window should be Active when opened. */
  NULL,          /* FirstGadget No Custom Gadgets. */
  NULL,          /* CheckMark   Use Intuition's default CheckMark (v). */
  "MY WINDOW 2", /* Title       Title of the window. */
  NULL,          /* Screen      Connected to the Workbench Screen. */
  NULL,          /* BitMap      No Custom BitMap. */
  80,            /* MinWidth    We will not allow the window to become */
  30,            /* MinHeight   smaller than 80 x 30, and not bigger */
  0,             /* MaxWidth    than the default sixe (200x100). */
  0,             /* MaxHeight */
  WBENCHSCREEN   /* Type        Connected to the Workbench Screen. */
};



/* Declare a pointer to Window structure number three: */
struct Window *my_window3;

/* Declare and initialize your NewWindow structure number three: */
struct NewWindow my_new_window3=
{
  10,            /* LeftEdge    x position of the window. */
  10,            /* TopEdge     y positio of the window. */
  400,           /* Width       400 pixels wide. */
  150,           /* Height      150 lines high. */
  0,             /* DetailPen   Text should be drawn with colour reg. 0 */
  1,             /* BlockPen    Blocks should be drawn with colour reg. 1 */
  NULL,          /* IDCMPFlags  No IDCMP flags. */
  SMART_REFRESH| /* Flags       Intuition should refresh the window. */
  BACKDROP|      /*             Backdrop window. */
  WINDOWCLOSE|   /*             Close Gadget. */
  ACTIVATE,      /*             The window should be Active when opened. */
  NULL,          /* FirstGadget No Custom Gadgets. */
  NULL,          /* CheckMark   Use Intuition's default CheckMark (v). */
  "BACKDROP",    /* Title       Title of the window. */
  NULL,          /* Screen      Connected to the Workbench Screen. */
  NULL,          /* BitMap      No Custom BitMap. */
  0,             /* MinWidth    We do not need to care about these */
  0,             /* MinHeight   since we havent supplied the window with */
  0,             /* MaxWidth    a Sizing Gadget. */
  0,             /* MaxHeight */
  WBENCHSCREEN   /* Type        Connected to the Workbench Screen. */
};



main()
```

```
{
  /* Open the Intuition Library: */
  IntuitionBase = (struct IntuitionBase *)
    OpenLibrary( "intuition.library", 0 );

  if( IntuitionBase == NULL )
    exit(); /* Could NOT open the Intuition Library! */



  /* We will now try to open the first window: */
  my_window1 = (struct Window *) OpenWindow( &my_new_window1 );

  /* Have we opened the first window succesfully? */
  if(my_window1 == NULL)
  {
    /* Could NOT open the first Window! */

    /* Close the Intuition Library since we have opened it: */
    CloseLibrary( IntuitionBase );

    exit();
  }



  /* We will now try to open the second window: */
  my_window2 = (struct Window *) OpenWindow( &my_new_window2 );

  /* Have we opened the second window succesfully? */
  if(my_window2 == NULL)
  {
    /* Could NOT open the second Window! */

    /* We must close the first window since we have opened it: */
    CloseWindow( my_window1 );

    /* Close the Intuition Library since we have opened it: */
    CloseLibrary( IntuitionBase );

    exit();
  }



  /* We will now try to open the third window: (The Backdrop window) */
  my_window3 = (struct Window *) OpenWindow( &my_new_window3 );

  /* Have we opened the third window succesfully? */
  if(my_window3 == NULL)
  {
    /* Could NOT open the third Window! */

    /* We must close the window one and two since we have opened them: */
    CloseWindow( my_window2 );
    CloseWindow( my_window1 );

    /* Close the Intuition Library since we have opened it: */
    CloseLibrary( IntuitionBase );

    exit();
  }



  /* We have opened the windows, and everything seems to be OK. */
  /* Wait for 30 seconds: */
  Delay( 50 * 30);



  /* We should always close the windows we have opened before we leave: */
  /* (It does not matter in which order we close the windows.) */
  CloseWindow( my_window1 );
  CloseWindow( my_window2 );
  CloseWindow( my_window3 );



  /* Close the Intuition Library since we have opened it: */
  CloseLibrary( IntuitionBase );

  /* THE END */
}
```

Example8

This program will open a SuperBitMap window which is connected to the Workbench Screen. Since it is a SuperBitMap we also make the window into a Gimmezerozero window. The window will use all System Gadgets, and some boxes will be drawn. It will display the window for 30 seconds, and then close it. (Shrink the window, and then enlarge it again, and you will noticed that the lines are still there!)

```
/* Example8                                                              */
/* This program will open a SuperBitMap window which is connected to the */
/* Workbench Screen. Since it is a SuperBitMap we also make the window   */
/* into a Gimmezerozero window. The window will use all System Gadgets,  */
/* and some boxes will be drawn. It will display the window for 30       */
/* seconds, and then close it.                                           */



/* If your program is using Intuition you should include intuition.h: */
#include <intuition/intuition.h>



#define WIDTH  320
#define HEIGHT 150
#define DEPTH    2



struct IntuitionBase *IntuitionBase;
struct GfxBase *GfxBase;



/***********************************************************/
/* 1. Declare and initialize a NewWindow structure with your */
/*    requirements:                                        */
/***********************************************************/

/* Declare a pointer to a Window structure: */
struct Window *my_window;

/* Declare and initialize your NewWindow structure: */
struct NewWindow my_new_window=
{
  10,            /* LeftEdge    x position of the window. */
  30,            /* TopEdge     y positio of the window. */
  200,           /* Width       200 pixels wide. */
  50,            /* Height      50 lines high. */
  0,             /* DetailPen   Text should be drawn with colour reg. 0 */
  1,             /* BlockPen    Blocks should be drawn with colour reg. 1 */
  NULL,          /* IDCMPFlags  No IDCMP flags. */
  SUPER_BITMAP|  /* Flags       SuperBitMap. (No refreshing necessary) */
  GIMMEZEROZERO| /*             It is also a Gimmezerozero window. */
  WINDOWCLOSE|   /*             Close Gadget. */
  WINDOWDRAG|    /*             Drag gadget. */
  WINDOWDEPTH|   /*             Depth arrange Gadgets. */
  WINDOWSIZING|  /*             Sizing Gadget. */
  ACTIVATE,      /*             The window should be Active when opened. */
  NULL,          /* FirstGadget No Custom Gadgets. */
  NULL,          /* CheckMark   Use Intuition's default CheckMark (v). */
  "SuperBitMap", /* Title       Title of the window. */
  NULL,          /* Screen      Connected to the Workbench Screen. */
  NULL,          /* BitMap      We will change this later. */
  80,            /* MinWidth    We will not allow the window to become */
  30,            /* MinHeight   smaller than 80 x 30, and not bigger */
```

```
  WIDTH,         /* MaxWidth    than 320 x 150. */
  HEIGHT,        /* MaxHeight */
  WBENCHSCREEN   /* Type        Connected to the Workbench Screen. */
};



/*********************************/
/* 2. Declare a BitMap structure: */
/*********************************/

struct BitMap my_bitmap;



main()
{
  int loop;

  /* Open the Intuition Library: */
  IntuitionBase = (struct IntuitionBase *)
    OpenLibrary( "intuition.library", 0 );

  if( IntuitionBase == NULL )
    exit(); /* Could NOT open the Intuition Library! */



  /* Open the Graphics Library: */
  GfxBase = (struct GfxBase *)
    OpenLibrary( "graphics.library", 0);

  if( GfxBase == NULL )
  {
    /* Could NOT open the Graphics Library! */

    /* Close the Intuition Library since we have opened it: */
    CloseLibrary( IntuitionBase );

    exit();
  }



  /********************************************************/
  /* 3. Initialize your own BitMap by calling the function: */
  /********************************************************/

  InitBitMap( &my_bitmap, DEPTH, WIDTH, HEIGHT );

  /* &my_bitmap: A pointer to the my_bitmap structure. */
  /* DEPTH:      Number of bitplanes to use. */
  /* WIDTH:      The width of the BitMap. (Must be a multiple of 16) */
  /* HEIGHT:     The height of the BitMap. */
```

```
/**********************************************/
/* 4. Allocate display memory for the BitMap: */
/**********************************************/

for( loop=0; loop < DEPTH; loop++)
  if((my_bitmap.Planes[loop] = (PLANEPTR)
    AllocRaster( WIDTH, HEIGHT )) == NULL )
  {
    /* PANIC! Not enough memory */

    /* Deallocate the display memory, Bitplan by Bitplan. */
    for( loop=0; loop < DEPTH; loop++)
      if( my_bitmap.Planes[loop] ) /* Deallocate this Bitplan? */
        FreeRaster( my_bitmap.Planes[loop], WIDTH, HEIGHT );

    /* Close the Graphics Library since we have opened it: */
    CloseLibrary( GfxBase );

    /* Close the Intuition Library since we have opened it: */
    CloseLibrary( IntuitionBase );

    exit();
  }

/* The (PLANEPTR) is not necessary, but you will now not recieve any   */
/* warnings messages about "pointers do not point to same type of      */
/* object". This is because my_bitmap.Planes expects to get a          */
/* memory pointer to some display memory (PLANEPTR), while AllocRaster */
/* returns an APTR (memory pointer). It is actually no difference      */
/* between them, but two different names (declarartions) makes the    */
/* paranoid C compiler worried. To calm it down we make this "casting" */



/***************************/
/* 5. Clear all Bitplanes: */
/***************************/

for( loop=0; loop < DEPTH; loop++)
  BltClear( my_bitmap.Planes[loop], RASSIZE( WIDTH, HEIGHT ), 0);

/* The memory we allocated for the Bitplanes, is normaly "dirty", and */
/* therefore needs cleaning. We can here use the Blitter to clear the */
/* memory since it is the fastest way to do it, and the easiest.      */
/* RASSIZE is a macro which calculates memory size for a Bitplane of  */
/* the size WIDTH x HEIGHT. We will later go into more details about  */
/* these functions etc, so do not worry about them... yet.            */



/*******************************************************************/
/* 6. Make sure the NewWindow's BitMap pointer is pointing to your */
/*    BitMap structure:                                            */
/*******************************************************************/

my_new_window.BitMap=&my_bitmap;
```

```
/***************************************/
/* 7. At last you can open the window: */
/***************************************/

my_window = (struct Window *) OpenWindow( &my_new_window );

/* Have we opened the window succesfully? */
if(my_window == NULL)
{
  /* Could NOT open the Window! */

  /* Deallocate the display memory, Bitplan by Bitplan. */
  for( loop=0; loop < DEPTH; loop++)
    if( my_bitmap.Planes[loop] ) /* Deallocate this Bitplan? */
      FreeRaster( my_bitmap.Planes[loop], WIDTH, HEIGHT );

  /* Close the Graphics Library since we have opened it: */
  CloseLibrary( GfxBase );

  /* Close the Intuition Library since we have opened it: */
  CloseLibrary( IntuitionBase );

  exit();
}



/* Do not bother aboute these commands, since I will explain more */
/* about them later. I have included them here since I want to put */
/* some graphics into the window, so you can see how a SuperBitMap */
/* window works. (Shrink the window, and then enlarge it again, and */
/* you will noticed that the lines are still there!) */

SetDrMd( my_window->RPort, JAM1 );

SetAPen( my_window->RPort, 1 );
Move( my_window->RPort,  10,  10 );
Draw( my_window->RPort, 100,  10 );
Draw( my_window->RPort, 100, 100 );
Draw( my_window->RPort,  10, 100 );
Draw( my_window->RPort,  10,  10 );

SetAPen( my_window->RPort, 2 );
Move( my_window->RPort,  12,  12 );
Draw( my_window->RPort,  98,  12 );
Draw( my_window->RPort,  98,  98 );
Draw( my_window->RPort,  12,  98 );
Draw( my_window->RPort,  12,  12 );

SetAPen( my_window->RPort, 3 );
Move( my_window->RPort,  14,  14 );
Draw( my_window->RPort,  96,  14 );
Draw( my_window->RPort,  96,  96 );
Draw( my_window->RPort,  14,  96 );
Draw( my_window->RPort,  14,  14 );
```

```
  /* We have opened the window, and everything seems to be OK. */
  /* Wait for 30 seconds: */
  Delay( 50 * 30);



  /********************************************************************/
  /* 8. Do not forget to close the window, AND deallocate the display */
  /*    memory:                                                       */
  /********************************************************************/

  /* We should always close the windows we have opened before we leave: */
  CloseWindow( my_window );

  /* Deallocate the display memory, Bitplan by Bitplan. */
  for( loop=0; loop < DEPTH; loop++)
    if( my_bitmap.Planes[loop] ) /* Deallocate this Bitplan? */
      FreeRaster( my_bitmap.Planes[loop], WIDTH, HEIGHT );



  /* Close the Graphics Library since we have opened it: */
  CloseLibrary( GfxBase );

  /* Close the Intuition Library since we have opened it: */
  CloseLibrary( IntuitionBase );

  /* THE END */
}
```

Example9

This program will open a normal window with all system gadgets connected to it. If you activate the window, the pointer will change shapes into a "nice" arrow.

```
/* Example9                                                         */
/* This program will open a normal window with all system gadgets      */
/* connected to it. If you activate the window, the pointer will chage */
/* shapes into a "nice" arrow.                                         */



/* If your program is using Intuition you should include intuition.h: */
#include <intuition/intuition.h>



struct IntuitionBase *IntuitionBase;



/* Declare a pointer to a Window structure: */
struct Window *my_window;

/* Declare and initialize your NewWindow structure: */
struct NewWindow my_new_window=
{
  50,            /* LeftEdge    x position of the window. */
  50,            /* TopEdge     y positio of the window. */
  200,           /* Width       200 pixels wide. */
  150,           /* Height      150 lines high. */
  0,             /* DetailPen   Text should be drawn with colour reg. 0 */
  1,             /* BlockPen    Blocks should be drawn with colour reg. 1 */
  NULL,          /* IDCMPFlags  No IDCMP flags. */
  SMART_REFRESH| /* Flags       Intuition should refresh the window. */
  WINDOWCLOSE|   /*             Close Gadget. */
  WINDOWDRAG|    /*             Drag gadget. */
  WINDOWDEPTH|   /*             Depth arrange Gadgets. */
  WINDOWSIZING,  /*             Sizing Gadget. */
  NULL,          /* FirstGadget No Custom Gadgets. */
  NULL,          /* CheckMark   Use Intuition's default CheckMark (v). */
  "MY WINDOW",   /* Title       Title of the window. */
  NULL,          /* Screen      Connected to the Workbench Screen. */
  NULL,          /* BitMap      No Custom BitMap. */
  80,            /* MinWidth    We will not allow the window to become */
  30,            /* MinHeight   smaller than 80 x 30, and not bigger */
  300,           /* MaxWidth    than 300 x 200. */
  200,           /* MaxHeight */
  WBENCHSCREEN   /* Type        Connected to the Workbench Screen. */
};



/* Declare and initialize Sprite data for the Pointer: */
USHORT chip my_sprite_data[36]=
{
         0x0000, 0x0000, /* Used by Intuition only. */

         0x0000, 0x0100,
         0x0000, 0x0300,
         0x0200, 0x0700,
         0x0600, 0x0D00,
         0x0E00, 0x1900,
         0x1E00, 0x31FC,
         0x3FFC, 0x60FE,
         0x7FFE, 0xc003,
         0x3FFE, 0x4001,
         0x1E0E, 0x21F1,
         0x0E0E, 0x1119,
         0x060E, 0x0919,
         0x020E, 0x0519,
         0x000E, 0x0319,
         0x000E, 0x0119,
         0x0000, 0x001F,

         0x0000, 0x0000  /* Used by Intuition only. */
};



main()
{
  /* Open the Intuition Library: */
  IntuitionBase = (struct IntuitionBase *)
    OpenLibrary( "intuition.library", 0 );

  if( IntuitionBase == NULL )
    exit(); /* Could NOT open the Intuition Library! */



  /* We will now try to open the window: */
  my_window = (struct Window *) OpenWindow( &my_new_window );

  /* Have we opened the window succesfully? */
  if(my_window == NULL)
  {
    /* Could NOT open the Window! */

    /* Close the Intuition Library since we have opened it: */
    CloseLibrary( IntuitionBase );

    exit();
  }



  /* We will now call the function SetPointer() to change the windows */
  /* default pointer. If you now Activate the window, by clicking */
  /* somewhere inside it, the pointer will change: */
  SetPointer( my_window, my_sprite_data, 16, 16, 0, -7);

  /* my_window:       Pointer to the window. */
  /* &my_sprite_data: Pointer to the Sprite Data. */
  /* 16:              Height, 16 lines. */
  /* 16:              Width, 16 pixels. */
  /* 0:               XOffset, left side. (Position of the "Hot Spot") */
  /* -7:              YOffset, 7 lines down.         -"- */
/* We have opened the window, and everything seems to be OK. */
/* Wait for 30 seconds: */
Delay( 50 * 30);


/* We should always close the windows we have opened before we leave: */
CloseWindow( my_window );


/* Close the Intuition Library since we have opened it: */
CloseLibrary( IntuitionBase );

/* THE END */
}
```

Example10

This program will open a two normal windows with all system gadgets connected to them. If the first window is Activated, the pointer will change shapes into a Zzz symbol, if the second window is activated, the pointer will look like a pistol.

```
/* Example10                                                           */
/* This program will open a two normal windows with all system gadgets   */
/* connected to them. If the first window is Activated, the pointer will */
/* chage shapes into a Zzz symbol, if the second window is activated,    */
/* the pointer will look like a pistol.                                  */



/* If your program is using Intuition you should include intuition.h: */
#include <intuition/intuition.h>



struct IntuitionBase *IntuitionBase;



/* Declare a pointer to the first Window structure: */
struct Window *my_window1;

/* Declare and initialize your first NewWindow structure: */
struct NewWindow my_new_window1=
{
  50,            /* LeftEdge    x position of the window. */
  25,            /* TopEdge     y positio of the window. */
  200,           /* Width       200 pixels wide. */
  150,           /* Height      150 lines high. */
  0,             /* DetailPen   Text should be drawn with colour reg. 0 */
  1,             /* BlockPen    Blocks should be drawn with colour reg. 1 */
  NULL,          /* IDCMPFlags  No IDCMP flags. */
  SMART_REFRESH| /* Flags       Intuition should refresh the window. */
  WINDOWCLOSE|   /*             Close Gadget. */
  WINDOWDRAG|    /*             Drag gadget. */
  WINDOWDEPTH|   /*             Depth arrange Gadgets. */
  WINDOWSIZING,  /*             Sizing Gadget. */
  NULL,          /* FirstGadget No Custom Gadgets. */
  NULL,          /* CheckMark   Use Intuition's default CheckMark (v). */
  "Zzz",         /* Title       Title of the window. */
  NULL,          /* Screen      Connected to the Workbench Screen. */
  NULL,          /* BitMap      No Custom BitMap. */
  80,            /* MinWidth    We will not allow the window to become */
  30,            /* MinHeight   smaller than 80 x 30, and not bigger */
  300,           /* MaxWidth    than 300 x 200. */
  200,           /* MaxHeight */
  WBENCHSCREEN   /* Type        Connected to the Workbench Screen. */
};



/* Declare a pointer to the second Window structure: */
struct Window *my_window2;

/* Declare and initialize your second NewWindow structure: */
struct NewWindow my_new_window2=
{
  300,           /* LeftEdge    x position of the window. */
  25,            /* TopEdge     y positio of the window. */
  200,           /* Width       200 pixels wide. */
  150,           /* Height      150 lines high. */
  0,             /* DetailPen   Text should be drawn with colour reg. 0 */
  1,             /* BlockPen    Blocks should be drawn with colour reg. 1 */
  NULL,          /* IDCMPFlags  No IDCMP flags. */
  SMART_REFRESH| /* Flags       Intuition should refresh the window. */
  WINDOWCLOSE|   /*             Close Gadget. */
  WINDOWDRAG|    /*             Drag gadget. */
  WINDOWDEPTH|   /*             Depth arrange Gadgets. */
  WINDOWSIZING,  /*             Sizing Gadget. */
  NULL,          /* FirstGadget No Custom Gadgets. */
  NULL,          /* CheckMark   Use Intuition's default CheckMark (v). */
  "BANG!",       /* Title       Title of the window. */
  NULL,          /* Screen      Connected to the Workbench Screen. */
  NULL,          /* BitMap      No Custom BitMap. */
  80,            /* MinWidth    We will not allow the window to become */
  30,            /* MinHeight   smaller than 80 x 30, and not bigger */
  300,           /* MaxWidth    than 300 x 200. */
  200,           /* MaxHeight */
  WBENCHSCREEN   /* Type        Connected to the Workbench Screen. */
};



/* Declare and initialize Sprite data for the Pointers: */

/* Zzz: (16 x 16  pixels) */
USHORT chip sprite_data_Zzz[36]=
{
         0x0000, 0x0000,  /* Used by Intuition only. */

         0x0300, 0x0000,
         0x1F9C, 0x0300,
         0x3FFE, 0x1F9C,
         0x63E3, 0x3FFE,
         0x7A3B, 0x3FFE,
         0xF7B7, 0x7FFE,
         0xEF63, 0x7FFE,
         0xE23F, 0x7FFE,
         0x7FFE, 0x3FC0,
         0x3fC0, 0x0F80,
         0x0FB0, 0x0000,
         0x0078, 0x0030,
         0x0030, 0x0000,
         0x0004, 0x0000,
         0x000E, 0x0004,
         0x0004, 0x0000,

         0x0000, 0x0000  /* Used by Intuition only. */
};

/* Pistol: (16 x 11  pixels) */
USHORT chip sprite_data_Pistol[26]=
{
         0x0000, 0x0000,  /* Used by Intuition only. */

         0x0000, 0x4010,
        0x0000, 0xFFF8,
        0x01E0, 0xFE18,
        0x00E0, 0x071C,
        0x0000, 0x03FC,
        0x001C, 0x027E,
        0x001C, 0x02BF,
        0x001E, 0x01FF,
        0x001E, 0x003F,
        0x000E, 0x001F,
        0x0000, 0x001F,

        0x0000, 0x0000  /* Used by Intuition only. */
};



main()
{
  /* Open the Intuition Library: */
  IntuitionBase = (struct IntuitionBase *)
    OpenLibrary( "intuition.library", 0 );

  if( IntuitionBase == NULL )
    exit(); /* Could NOT open the Intuition Library! */



  /* We will now try to open the first window: */
  my_window1 = (struct Window *) OpenWindow( &my_new_window1 );

  /* Have we opened the first window succesfully? */
  if(my_window1 == NULL)
  {
    /* Could NOT open the first Window! */

    /* Close the Intuition Library since we have opened it: */
    CloseLibrary( IntuitionBase );

    exit();
  }



  /* We will now try to open the second window: */
  my_window2 = (struct Window *) OpenWindow( &my_new_window2 );

  /* Have we opened the second window succesfully? */
  if(my_window2 == NULL)
  {
    /* Could NOT open the second Window! */

    /* Close the first window: */
    CloseWindow( my_window1 );

    /* Close the Intuition Library since we have opened it: */
    CloseLibrary( IntuitionBase );

    exit();
  }



  /* We will now call the function SetPointer() to change the windows */
  /* default pointer: */
  SetPointer( my_window1, sprite_data_Zzz, 16, 16, 0, 0);
  SetPointer( my_window2, sprite_data_Pistol, 11, 16, 0, -1);



  /* We have opened the windows, and everything seems to be OK. */
  /* Wait for 30 seconds: */
  Delay( 50 * 30);



  /* We should always close the windows we have opened before we leave: */
  CloseWindow( my_window2 );
  CloseWindow( my_window1 );



  /* Close the Intuition Library since we have opened it: */
  CloseLibrary( IntuitionBase );

  /* THE END */
}
```

A.3 GRAPHICS

Example1

This program will open a normal window which is connected to the Workbench Screen. We will then draw a strange line with help of Intuition's Border structure.

```
/* Example1                                                    */
/* This program will open a normal window which is connected to the */
/* Workbench Screen. We will then draw a strange line with help of  */
/* Intuition's Border structure.                               */


/* If your program is using Intuition you should include intuition.h: */
#include <intuition/intuition.h>


struct IntuitionBase *IntuitionBase;


/* Declare a pointer to a Window structure: */
struct Window *my_window;

/* Declare and initialize your NewWindow structure: */
struct NewWindow my_new_window=
{
  40,            /* LeftEdge    x position of the window. */
  20,            /* TopEdge     y positio of the window. */
  250,           /* Width       250 pixels wide. */
  40,            /* Height      40 lines high. */
  0,             /* DetailPen   Text should be drawn with colour reg. 0 */
  1,             /* BlockPen    Blocks should be drawn with colour reg. 1 */
  NULL,          /* IDCMPFlags  No IDCMP flags. */
  SMART_REFRESH| /* Flags       Intuition should refresh the window. */
  WINDOWDRAG|    /*             Drag gadget. */
  WINDOWDEPTH|   /*             Depth arrange Gadgets. */
  ACTIVATE,      /*             The window should be Active when opened. */
  NULL,          /* FirstGadget No Custom Gadgets. */
  NULL,          /* CheckMark   Use Intuition's default CheckMark (v). */
  "STRANGE LINE",/* Title       Title of the window. */
  NULL,          /* Screen      Connected to the Workbench Screen. */
  NULL,          /* BitMap      No Custom BitMap. */
  0,             /* MinWidth    We do not need to care about these */
  0,             /* MinHeight   since we have not supplied the window */
  0,             /* MaxWidth    with a Sizing Gadget. */
  0,             /* MaxHeight */
  WBENCHSCREEN   /* Type        Connected to the Workbench Screen. */
};


/* The coordinates for the lines: */
SHORT my_points[]=
{
  10,10, /* Start at position (10,10) */
  25,10, /* Draw a line to the right to position (25,10) */
  25,14, /* Draw a line down to position (25,14) */
  35,14, /* Draw a line to the right to position (35,14) */
  35,12  /* Finish of by drawing a line up to position (35,12) */
};


/* The Border structure: */
struct Border my_border=
{
  0, 0,        /* LeftEdge, TopEdge. */
  3,           /* FrontPen, colour register 3. */
  0,           /* BackPen, for the moment unused. */
  JAM1,        /* DrawMode, draw the lines with colour 3. */
  5,           /* Count, 5 pair of coordinates in the array. */
  my_points,   /* XY, pointer to the array with the coordinates. */
  NULL,        /* NextBorder, no other Border structures are connected. */
};


main()
{
  /* Open the Intuition Library: */
  IntuitionBase = (struct IntuitionBase *)
    OpenLibrary( "intuition.library", 0 );

  if( IntuitionBase == NULL )
    exit(); /* Could NOT open the Intuition Library! */



  /* We will now try to open the window: */
  my_window = (struct Window *) OpenWindow( &my_new_window );

  /* Have we opened the window succesfully? */
  if(my_window == NULL)
  {
    /* Could NOT open the Window! */

    /* Close the Intuition Library since we have opened it: */
    CloseLibrary( IntuitionBase );

    exit();
  }


  /* Tell Intuition to draw a strange line, using my_border structure: */
  DrawBorder( my_window->RPort, &my_border, 10, 12 );



  /* We have opened the window, and everything seems to be OK. */
  /* Wait for 30 seconds: */
  Delay( 50 * 30);



  /* We should always close the windows we have opened before we leave: */
  CloseWindow( my_window );


  /* Close the Intuition Library since we have opened it: */
  CloseLibrary( IntuitionBase );

  /* THE END */
}
```

Example2

This program will open a normal window which is connected to the Workbench Screen. We will then draw two rectangles with different colours. This shows how you can link Border structures to each other in order to get the desired effects.

```
/* Example2                                                        */
/* This program will open a normal window which is connected to the  */
/* Workbench Screen. We will then draw two rectangles with different */
/* colours. This shows how you can link Border structures to each    */
/* other in order to get the desired effects.                        */



/* If your program is using Intuition you should include intuition.h: */
#include <intuition/intuition.h>



struct IntuitionBase *IntuitionBase;



/* Declare a pointer to a Window structure: */
struct Window *my_window;

/* Declare and initialize your NewWindow structure: */
struct NewWindow my_new_window=
{
  40,            /* LeftEdge    x position of the window. */
  20,            /* TopEdge     y positio of the window. */
  250,           /* Width       250 pixels wide. */
  150,           /* Height      150 lines high. */
  0,             /* DetailPen   Text should be drawn with colour reg. 0 */
  1,             /* BlockPen    Blocks should be drawn with colour reg. 1 */
  NULL,          /* IDCMPFlags  No IDCMP flags. */
  SMART_REFRESH| /* Flags       Intuition should refresh the window. */
  WINDOWDRAG|    /*             Drag gadget. */
  WINDOWDEPTH|   /*             Depth arrange Gadgets. */
  ACTIVATE,      /*             The window should be Active when opened. */
  NULL,          /* FirstGadget No Custom Gadgets. */
  NULL,          /* CheckMark   Use Intuition's default CheckMark (v). */
  "RECTANGLES",  /* Title       Title of the window. */
  NULL,          /* Screen      Connected to the Workbench Screen. */
  NULL,          /* BitMap      No Custom BitMap. */
  0,             /* MinWidth    We do not need to care about these */
  0,             /* MinHeight   since we have not supplied the window */
  0,             /* MaxWidth    with a Sizing Gadget. */
  0,             /* MaxHeight */
  WBENCHSCREEN   /* Type        Connected to the Workbench Screen. */
};



/* The coordinates for the small rectangle: */
SHORT small_points[]=
{
   0,  0, /* Start at position (0,0) */
  80,  0, /* Draw a line to the right to position (80,0) */
  80, 40, /* Draw a line down to position (80,40) */
   0, 40, /* Draw a line to the left to position (0,40) */
   0,  0  /* Finish of by drawing a line up to position (0,0) */
};
```

```
/* The coordinates for the big rectangle: */
SHORT big_points[]=
{
    0,  0, /* Start at position (0,0) */
  100,  0, /* Draw a line to the right to position (100,0) */
  100, 50, /* Draw a line down to position (100,50) */
    0, 50, /* Draw a line to the left to position (0,50) */
    0,  0  /* Finish of by drawing a line up to position (0,0) */
};



/* The small Border structure: */
struct Border small_rectangle=
{
  10, 5,         /* LeftEdge, TopEdge. */
  3,             /* FrontPen, colour register 3. */
  0,             /* BackPen, for the moment unused. */
  JAM1,          /* DrawMode, draw the lines with colour 3. */
  5,             /* Count, 5 pair of coordinates in the array. */
  small_points,  /* XY, pointer to the array with the coordinates. */
  NULL           /* NextBorder, no other Border structures are connected. */
};



/* The BIG Border structure: */
struct Border big_rectangle=
{
  0, 0,              /* LeftEdge, TopEdge. */
  1,                 /* FrontPen, colour register 1. */
  0,                 /* BackPen, for the moment unused. */
  JAM1,              /* DrawMode, draw the lines with colour 1. */
  5,                 /* Count, 5 pair of coordinates in the array. */
  big_points,        /* XY, pointer to the array with the coordinates. */
  &small_rectangle   /* NextBorder, pointing to the small_rectangle. */
};



main()
{
  /* Open the Intuition Library: */
  IntuitionBase = (struct IntuitionBase *)
    OpenLibrary( "intuition.library", 0 );

  if( IntuitionBase == NULL )
    exit(); /* Could NOT open the Intuition Library! */



  /* We will now try to open the window: */
  my_window = (struct Window *) OpenWindow( &my_new_window );

  /* Have we opened the window succesfully? */
  if(my_window == NULL)
  {
    /* Could NOT open the Window! */

    /* Close the Intuition Library since we have opened it: */
    CloseLibrary( IntuitionBase );

    exit();
  }



  /* Tell Intuition to draw the rectangles: */
  DrawBorder( my_window->RPort, &big_rectangle, 10, 15 );



  /* Wait for 30 seconds: */
  Delay( 50 * 30);



  /* We should always close the windows we have opened before we leave: */
  CloseWindow( my_window );



  /* Close the Intuition Library since we have opened it: */
  CloseLibrary( IntuitionBase );

  /* THE END */
}
```

Example3

This program will open a normal window which is connected to the Workbench Screen. We will then print a text string with help of Intuition's IntuiText structure.

```
/* Example3                                                  */
/* This program will open a normal window which is connected to the */
/* Workbench Screen. We will then print a text string whith help of */
/* Intuition's IntuiText structure.                          */



/* If your program is using Intuition you should include intuition.h: */
#include <intuition/intuition.h>



struct IntuitionBase *IntuitionBase;



/* Declare a pointer to a Window structure: */
struct Window *my_window;

/* Declare and initialize your NewWindow structure: */
struct NewWindow my_new_window=
{
  40,            /* LeftEdge    x position of the window. */
  20,            /* TopEdge     y positio of the window. */
  400,           /* Width       400 pixels wide. */
  150,           /* Height      150 lines high. */
  0,             /* DetailPen   Text should be drawn with colour reg. 0 */
  1,             /* BlockPen    Blocks should be drawn with colour reg. 1 */
  NULL,          /* IDCMPFlags  No IDCMP flags. */
  SMART_REFRESH| /* Flags       Intuition should refresh the window. */
  WINDOWDRAG|    /*             Drag gadget. */
  WINDOWDEPTH|   /*             Depth arrange Gadgets. */
  ACTIVATE,      /*             The window should be Active when opened. */
  NULL,          /* FirstGadget No Custom Gadgets. */
  NULL,          /* CheckMark   Use Intuition's default CheckMark (v). */
  "TEXT",        /* Title       Title of the window. */
  NULL,          /* Screen      Connected to the Workbench Screen. */
  NULL,          /* BitMap      No Custom BitMap. */
  0,             /* MinWidth    We do not need to care about these */
  0,             /* MinHeight   since we have not supplied the window */
  0,             /* MaxWidth    with a Sizing Gadget. */
  0,             /* MaxHeight */
  WBENCHSCREEN   /* Type        Connected to the Workbench Screen. */
};



UBYTE my_text[]="This is the text that will be printed!";

struct IntuiText my_intui_text=
{
  1,         /* FrontPen, colour register 1. */
  2,         /* BackPen, colour register 2. */
  JAM2,      /* DrawMode, draw the characters with colour 1, on a colour */
             /* 2 background. (White text on a black background) */
  10, 20,    /* LeftEdge, TopEdge. */
  NULL,      /* ITextFont, use default font. */
  my_text,   /* IText, the text that will be printed. */
             /* (Remember my_text = &my_text[0].) */
  NULL,      /* NextText, no other IntuiText structures are connected. */
};



main()
{
  /* Open the Intuition Library: */
  IntuitionBase = (struct IntuitionBase *)
    OpenLibrary( "intuition.library", 0 );

  if( IntuitionBase == NULL )
    exit(); /* Could NOT open the Intuition Library! */



  /* We will now try to open the window: */
  my_window = (struct Window *) OpenWindow( &my_new_window );

  /* Have we opened the window succesfully? */
  if(my_window == NULL)
  {
    /* Could NOT open the Window! */

    /* Close the Intuition Library since we have opened it: */
    CloseLibrary( IntuitionBase );

    exit();
  }



  /* Tell Intuition to print the text: */
  PrintIText( my_window->RPort, &my_intui_text, 0, 0 );



  /* We have opened the window, and everything seems to be OK. */
  /* Wait for 30 seconds: */
  Delay( 50 * 30);



  /* We should always close the windows we have opened before we leave: */
  CloseWindow( my_window );



  /* Close the Intuition Library since we have opened it: */
  CloseLibrary( IntuitionBase );

  /* THE END */
}
```

Example4

Same as Example3 except that the text will be printed with underlined italic characters.

```
/* Example4                                                     */
/* This program will open a normal window which is connected to the */
/* Workbench Screen. We will then print a text string whith help of */
/* Intuition's IntuiText structure. The text will be in underlined  */
/* italic characters.                                           */


/* If your program is using Intuition you should include intuition.h: */
#include <intuition/intuition.h>


struct IntuitionBase *IntuitionBase;


/* Declare a pointer to a Window structure: */
struct Window *my_window;

/* Declare and initialize your NewWindow structure: */
struct NewWindow my_new_window=
{
  40,            /* LeftEdge    x position of the window. */
  20,            /* TopEdge     y positio of the window. */
  400,           /* Width       400 pixels wide. */
  150,           /* Height      150 lines high. */
  0,             /* DetailPen   Text should be drawn with colour reg. 0 */
  1,             /* BlockPen    Blocks should be drawn with colour reg. 1 */
  NULL,          /* IDCMPFlags  No IDCMP flags. */
  SMART_REFRESH| /* Flags       Intuition should refresh the window. */
  WINDOWDRAG|    /*             Drag gadget. */
  WINDOWDEPTH|   /*             Depth arrange Gadgets. */
  ACTIVATE,      /*             The window should be Active when opened. */
  NULL,          /* FirstGadget No Custom Gadgets. */
  NULL,          /* CheckMark   Use Intuition's default CheckMark (v). */
  "STYLE!",      /* Title       Title of the window. */
  NULL,          /* Screen      Connected to the Workbench Screen. */
  NULL,          /* BitMap      No Custom BitMap. */
  0,             /* MinWidth    We do not need to care about these */
  0,             /* MinHeight   since we have not supplied the window */
  0,             /* MaxWidth    with a Sizing Gadget. */
  0,             /* MaxHeight */
  WBENCHSCREEN   /* Type        Connected to the Workbench Screen. */
};


struct TextAttr my_font=
{
  "topaz.font",                 /* Topaz font. */
  TOPAZ_EIGHTY,                 /* 80/40 characters (high-/low-res). */
  FSF_ITALIC | FSF_UNDERLINED,  /* Underlined italic characters. */
  FPF_ROMFONT                   /* Exist in ROM. */
};

UBYTE my_text[]="Nice style! Italic and Underlined!";

struct IntuiText my_intui_text=
{
  1,         /* FrontPen, colour register 1. */
  2,         /* BackPen, colour register 2. */
  JAM2,      /* DrawMode, draw the characters with colour 1, on a colour */
             /* 2 background. (White text on a black background) */
  10, 20,    /* LeftEdge, TopEdge. */
  &my_font,  /* ITextFont, use my_font. */
  my_text,   /* IText, the text that will be printed. */
             /* (Remember my_text = &my_text[0].) */
  NULL       /* NextText, no other IntuiText structures are connected. */
};


main()
{
  /* Open the Intuition Library: */
  IntuitionBase = (struct IntuitionBase *)
    OpenLibrary( "intuition.library", 0 );

  if( IntuitionBase == NULL )
    exit(); /* Could NOT open the Intuition Library! */



  /* We will now try to open the window: */
  my_window = (struct Window *) OpenWindow( &my_new_window );

  /* Have we opened the window succesfully? */
  if(my_window == NULL)
  {
    /* Could NOT open the Window! */

    /* Close the Intuition Library since we have opened it: */
    CloseLibrary( IntuitionBase );

    exit();
  }



  /* Tell Intuition to print the text: */
  PrintIText( my_window->RPort, &my_intui_text, 0, 0 );


  /* We have opened the window, and everything seems to be OK. */
  /* Wait for 30 seconds: */
  Delay( 50 * 30);



  /* We should always close the windows we have opened before we leave: */
  CloseWindow( my_window );


  /* Close the Intuition Library since we have opened it: */
  CloseLibrary( IntuitionBase );

  /* THE END */
}
```

Example5

This program will open a normal window which is connected to the Workbench Screen. We will then draw the little nice arrow we talked so much about.

```
/* Example5                                                      */
/* This program will open a normal window which is connected to the    */
/* Workbench Screen. We will then draw the little nice arrow we talked */
/* so much about.                                                */


/* If your program is using Intuition you should include intuition.h: */
#include <intuition/intuition.h>


struct IntuitionBase *IntuitionBase;


/* Declare a pointer to a Window structure: */
struct Window *my_window;

/* Declare and initialize your NewWindow structure: */
struct NewWindow my_new_window=
{
  40,            /* LeftEdge    x position of the window. */
  20,            /* TopEdge     y positio of the window. */
  100,           /* Width       100 pixels wide. */
  80,            /* Height      80 lines high. */
  0,             /* DetailPen   Text should be drawn with colour reg. 0 */
  1,             /* BlockPen    Blocks should be drawn with colour reg. 1 */
  NULL,          /* IDCMPFlags  No IDCMP flags. */
  SMART_REFRESH| /* Flags       Intuition should refresh the window. */
  WINDOWDRAG|    /*             Drag gadget. */
  WINDOWDEPTH|   /*             Depth arrange Gadgets. */
  ACTIVATE,      /*             The window should be Active when opened. */
  NULL,          /* FirstGadget No Custom Gadgets. */
  NULL,          /* CheckMark   Use Intuition's default CheckMark (v). */
  "ARROW",       /* Title       Title of the window. */
  NULL,          /* Screen      Connected to the Workbench Screen. */
  NULL,          /* BitMap      No Custom BitMap. */
  0,             /* MinWidth    We do not need to care about these */
  0,             /* MinHeight   since we have not supplied the window */
  0,             /* MaxWidth    with a Sizing Gadget. */
  0,             /* MaxHeight */
  WBENCHSCREEN   /* Type        Connected to the Workbench Screen. */
};


/* REMEMBER! Image data MUST be put in chip-memory! */
USHORT chip my_image_data[]=
{
  0x1000, /* BitPlane ZERO */
  0x3800,
  0x7C00,
  0xFE00,
  0x1000,
  0x1000,
  0x1000,
  0x1000
};

struct Image my_image=
{
  45, 35,          /* LeftEdge, TopEdge. */
  7,               /* Width, 7 pixels/bitts wide. */
  8,               /* Height, 8 lines high. */
  1,               /* Depth, only one Bitplane. */
  my_image_data,   /* ImageData, pointer to my_image_data. */
  0x0001,          /* PickPlane, bitplane Zero affects. */
  0x0000,          /* PlaneOnOff, 0's on all other Bitplanes. */
                   /* [The pixels' colour will be either 0000 (blue) or */
                   /* 0001 (white).] */
  NULL             /* NextImage, no more Images. */
};


main()
{
  /* Open the Intuition Library: */
  IntuitionBase = (struct IntuitionBase *)
    OpenLibrary( "intuition.library", 0 );

  if( IntuitionBase == NULL )
    exit(); /* Could NOT open the Intuition Library! */


  /* We will now try to open the window: */
  my_window = (struct Window *) OpenWindow( &my_new_window );

  /* Have we opened the window succesfully? */
  if(my_window == NULL)
  {
    /* Could NOT open the Window! */

    /* Close the Intuition Library since we have opened it: */
    CloseLibrary( IntuitionBase );

    exit();
  }


  /* Tell Intuition to draw the image: */
  DrawImage( my_window->RPort, &my_image, 0, 0 );


  /* We have opened the window, and everything seems to be OK. */
  /* Wait for 30 seconds: */
  Delay( 50 * 30);


  /* We should always close the windows we have opened before we leave: */
  CloseWindow( my_window );


  /* Close the Intuition Library since we have opened it: */
  CloseLibrary( IntuitionBase );

  /* THE END */
}
```

Example6

Same as Example5 except that we will draw it several times in different colours. This shows how PlanePick/PlaneOnOff works.

```
/* Example6                                                          */
/* This program will open a normal window which is connected to the    */
/* Workbench Screen. We will then draw the little nice arrow we talked */
/* so much about. This time, however, we draw it several times in      */
/* different colours. This shows how PlanePick/PlaneOnOff works.       */



/* If your program is using Intuition you should include intuition.h: */
#include <intuition/intuition.h>



struct IntuitionBase *IntuitionBase;



/* Declare a pointer to a Window structure: */
struct Window *my_window;

/* Declare and initialize your NewWindow structure: */
struct NewWindow my_new_window=
{
  40,            /* LeftEdge    x position of the window. */
  20,            /* TopEdge     y positio of the window. */
  150,           /* Width       150 pixels wide. */
  80,            /* Height      80 lines high. */
  0,             /* DetailPen   Text should be drawn with colour reg. 0 */
  1,             /* BlockPen    Blocks should be drawn with colour reg. 1 */
  NULL,          /* IDCMPFlags  No IDCMP flags. */
  SMART_REFRESH| /* Flags       Intuition should refresh the window. */
  WINDOWDRAG|    /*             Drag gadget. */
  WINDOWDEPTH|   /*             Depth arrange Gadgets. */
  ACTIVATE,      /*             The window should be Active when opened. */
  NULL,          /* FirstGadget No Custom Gadgets. */
  NULL,          /* CheckMark   Use Intuition's default CheckMark (v). */
  "ARROWS",      /* Title       Title of the window. */
  NULL,          /* Screen      Connected to the Workbench Screen. */
  NULL,          /* BitMap      No Custom BitMap. */
  0,             /* MinWidth    We do not need to care about these */
  0,             /* MinHeight   since we have not supplied the window */
  0,             /* MaxWidth    with a Sizing Gadget. */
  0,             /* MaxHeight */
  WBENCHSCREEN   /* Type        Connected to the Workbench Screen. */
};



/* REMEMBER! Image data MUST be put in chip-memory! */
USHORT chip my_image_data[]=
{
  0x1000, /* BitPlane ZERO */
  0x3800,
  0x7C00,
  0xFE00,
  0x1000,
  0x1000,
```

```
  0x1000,
  0x1000
};

/* Orange arrow on black background: */
struct Image my_image4=
{
  70, 30,         /* LeftEdge, TopEdge. */
  7,              /* Width, 7 pixels/bitts wide. */
  8,              /* Height, 8 lines high. */
  1,              /* Depth, only one Bitplane. */
  my_image_data,  /* ImageData, pointer to my_image_data. */
  0x0001,         /* PickPlane, bitplane Zero affects. */
  0x0002,         /* PlaneOnOff, Bitplane One will be filled with 1's. */
                  /* [The pixels' colour will be either 0010 (black) or */
                  /* 0011 (orange).] */
  NULL            /* NextImage, last structure in the list. */
};

/* Orange arrow on white background: */
struct Image my_image3=
{
  50, 30,         /* LeftEdge, TopEdge. */
  7,              /* Width, 7 pixels/bitts wide. */
  8,              /* Height, 8 lines high. */
  1,              /* Depth, only one Bitplane. */
  my_image_data,  /* ImageData, pointer to my_image_data. */
  0x0002,         /* PickPlane, bitplane One affects. */
  0x0001,         /* PlaneOnOff, Bitplane Zero will be filled with 1's. */
                  /* [The pixels' colour will be either 0001 (white) or */
                  /* 0011 (orange).] */
  &my_image4      /* NextImage, linked to my_image2. */
};

/* Black arrow on blue background: */
struct Image my_image2=
{
  30, 30,         /* LeftEdge, TopEdge. */
  7,              /* Width, 7 pixels/bitts wide. */
  8,              /* Height, 8 lines high. */
  1,              /* Depth, only one Bitplane. */
  my_image_data,  /* ImageData, pointer to my_image_data. */
  0x0002,         /* PickPlane, bitplane One affects. */
  0x0000,         /* PlaneOnOff, 0's on all other Bitplanes. */
                  /* [The pixels' colour will be either 0000 (0:blue) or */
                  /* 0010 (2:black).] */
  &my_image3      /* NextImage, linked to my_image3. */
};

/* White arrow on blue background: */
struct Image my_image1=
{
  10, 30,         /* LeftEdge, TopEdge. */
  7,              /* Width, 7 pixels/bitts wide. */
  8,              /* Height, 8 lines high. */
  1,              /* Depth, only one Bitplane. */
  my_image_data,  /* ImageData, pointer to my_image_data. */
  0x0001,         /* PickPlane, bitplane Zero affects. */
  0x0000,         /* PlaneOnOff, 0's on all other Bitplanes. */
                  /* [The pixels' colour will be either 0000 (0:blue) or */
                  /* 0001 (1:white).] */
  &my_image2      /* NextImage, linked to my_image2. */
};



main()
{
  /* Open the Intuition Library: */
  IntuitionBase = (struct IntuitionBase *)
    OpenLibrary( "intuition.library", 0 );

  if( IntuitionBase == NULL )
    exit(); /* Could NOT open the Intuition Library! */



  /* We will now try to open the window: */
  my_window = (struct Window *) OpenWindow( &my_new_window );

  /* Have we opened the window succesfully? */
  if(my_window == NULL)
  {
    /* Could NOT open the Window! */

    /* Close the Intuition Library since we have opened it: */
    CloseLibrary( IntuitionBase );

    exit();
  }



  /* Tell Intuition to draw the images: */
  DrawImage( my_window->RPort, &my_image1, 0, 0 );



  /* We have opened the window, and everything seems to be OK. */
  /* Wait for 30 seconds: */
  Delay( 50 * 30);



  /* We should always close the windows we have opened before we leave: */
  CloseWindow( my_window );



  /* Close the Intuition Library since we have opened it: */
  CloseLibrary( IntuitionBase );

  /* THE END */
}
```

Example7

This program will open a normal window which is connected to the Workbench Screen. We will then draw the nice 4 colour face that was described in chapter 3.5 IMAGES.

```
/* Example7                                                          */
/* This program will open a normal window which is connected to the    */
/* Workbench Screen. We will then draw the nice 4 colour face that was */
/* described in chapter 3.5 IMAGES.                                  */



/* If your program is using Intuition you should include intuition.h: */
#include <intuition/intuition.h>



struct IntuitionBase *IntuitionBase;



/* Declare a pointer to a Window structure: */
struct Window *my_window;

/* Declare and initialize your NewWindow structure: */
struct NewWindow my_new_window=
{
  40,            /* LeftEdge    x position of the window. */
  20,            /* TopEdge     y positio of the window. */
  250,           /* Width       250 pixels wide. */
  80,            /* Height      80 lines high. */
  0,             /* DetailPen   Text should be drawn with colour reg. 0 */
  1,             /* BlockPen    Blocks should be drawn with colour reg. 1 */
  NULL,          /* IDCMPFlags  No IDCMP flags. */
  SMART_REFRESH| /* Flags       Intuition should refresh the window. */
  WINDOWDRAG|    /*             Drag gadget. */
  WINDOWDEPTH|   /*             Depth arrange Gadgets. */
  ACTIVATE,      /*             The window should be Active when opened. */
  NULL,          /* FirstGadget No Custom Gadgets. */
  NULL,          /* CheckMark   Use Intuition's default CheckMark (v). */
  "THE 4 COLOUR FACE",/* Title  Title of the window. */
  NULL,          /* Screen      Connected to the Workbench Screen. */
  NULL,          /* BitMap      No Custom BitMap. */
  0,             /* MinWidth    We do not need to care about these */
  0,             /* MinHeight   since we have not supplied the window */
  0,             /* MaxWidth    with a Sizing Gadget. */
  0,             /* MaxHeight */
  WBENCHSCREEN   /* Type        Connected to the Workbench Screen. */
};



/* REMEMBER! Image data MUST be put in chip-memory! */
USHORT chip my_image_data[]= /* Image data for a nice four colour face: */
{
  0x3E00, /* Bitplane ZERO */
  0x7F00,
  0xC980,
  0xBE80,
  0xFF80,
  0xFF80,
  0xEB80,
  0xEB80,
  0xFF80,
  0xDD80,
  0x6300,
  0x7F00,
  0x3E00,

  0x3E00, /* Bitplane ONE */
  0x7F00,
  0xFF80,
  0xFF80,
  0xC980,
  0xC980,
  0xDD80,
  0xDD80,
  0xFF80,
  0xFF80,
  0x7F00,
  0x7F00,
  0x3E00
};

struct Image my_image=
{
  40, 30,         /* LeftEdge, TopEdge. */
  9,              /* Width, 9 pixels/bitts wide. */
  13,             /* Height, 13 lines high. */
  2,              /* Depth, two Bitplanes, 4 colours. */
  my_image_data,  /* ImageData, pointer to my_image_data. */
  0x0003,         /* PickPlane, bitplane Zero and One affects. */
  0x0000,         /* PlaneOnOff, all Bitplanes are already "picked". */
  NULL            /* NextImage, no more Images. */
};



main()
{
  /* Open the Intuition Library: */
  IntuitionBase = (struct IntuitionBase *)
    OpenLibrary( "intuition.library", 0 );

  if( IntuitionBase == NULL )
    exit(); /* Could NOT open the Intuition Library! */



  /* We will now try to open the window: */
  my_window = (struct Window *) OpenWindow( &my_new_window );

  /* Have we opened the window succesfully? */
  if(my_window == NULL)
  {
    /* Could NOT open the Window! */

    /* Close the Intuition Library since we have opened it: */
    CloseLibrary( IntuitionBase );
    exit();
  }



  /* Tell Intuition to draw the face: */
  DrawImage( my_window->RPort, &my_image, 0, 0 );



  /* We have opened the window, and everything seems to be OK. */
  /* Wait for 30 seconds: */
  Delay( 50 * 30);



  /* We should always close the windows we have opened before we leave: */
  CloseWindow( my_window );



  /* Close the Intuition Library since we have opened it: */
  CloseLibrary( IntuitionBase );

  /* THE END */
}
```

Example8

This program will open a normal window which is connected to a 16-colour Custom screen. In the window we will draw the famous AMIGA-logo.

```
/* Example8                                                        */
/* This program will open a normal window which is connected to a  */
/* 16-colour Custom screen. In the window we will draw the Amiga Logo. */



/* If your program is using Intuition you should include intuition.h: */
#include <intuition/intuition.h>



struct IntuitionBase *IntuitionBase;
struct GfxBase *GfxBase;



/* Declare a pointer to a Screen structure: */
struct Screen *my_screen;

/* Declare and initialize your NewScreen structure: */
struct NewScreen my_new_screen=
{
  0,            /* LeftEdge  Should always be 0. */
  0,            /* TopEdge   Top of the display.*/
  320,          /* Width     We are using a low-resolution screen. */
  200,          /* Height    Non-Interlaced NTSC (American) display. */
  4,            /* Depth     16 colours. */
  0,            /* DetailPen Text should be printed with colour reg. 0 */
  1,            /* BlockPen  Blocks should be printed with colour reg. 1 */
  NULL,         /* ViewModes Low-resolution. (Non-Interlaced) */
  CUSTOMSCREEN, /* Type      Your own customized screen. */
  NULL,         /* Font      Default font. */
  "N I C E",    /* Title     The screen's title. */
  NULL,         /* Gadget    Must for the moment be NULL. */
  NULL          /* BitMap    No special CustomBitMap. */
};



/* Declare a pointer to a Window structure: */
struct Window *my_window;

/* Declare and initialize your NewWindow structure: */
struct NewWindow my_new_window=
{
  20,            /* LeftEdge    x position of the window. */
  20,            /* TopEdge     y position of the window. */
  147,           /* Width       147 pixels wide. */
  63,            /* Height      63 lines high. */
  0,             /* DetailPen   Text should be drawn with colour reg. 0 */
  1,             /* BlockPen    Blocks should be drawn with col. reg. 1 */
  NULL,          /* IDCMPFlags  No IDCMP flags. */
  SMART_REFRESH| /* Flags       Intuition should refresh the window. */
  WINDOWDRAG|    /*             Drag gadget. */
  WINDOWDEPTH|   /*             Depth arrange Gadgets. */
  ACTIVATE,      /*             The window should be Active when opened. */
  NULL,          /* FirstGadget No Custom Gadgets. */
  NULL,          /* CheckMark   Use Intuition's default CheckMark (v). */
  "AMIGA!",      /* Title       Title of the window. */
  NULL,          /* Screen      We will later connect it to the screen. */
  NULL,          /* BitMap      No Custom BitMap. */
  0,             /* MinWidth    We do not need to care about these */
  0,             /* MinHeight   since we have not supplied the window */
  0,             /* MaxWidth    with a Sizing Gadget. */
  0,             /* MaxHeight */
  CUSTOMSCREEN   /* Type        Connected to a Custom screen. */
};



/* Here is the data for the Amiga Logo: */
/* REMEMBER! Image data MUST be placed in chip-memory! */
SHORT chip amiga_logo_data[]=
{
  /* BitPlane ZERO */
  0xFFFF,0xFFFF,0xFFF8,0x080F,0xFFFF,0xFFFF,0xFFFF,0xFFFF,0xFFFF,
  0xFFFF,0xFFFF,0xFFF0,0x101F,0xFFFF,0xFFFF,0xFFFF,0xFFFF,0xFFFF,
  0xFFFF,0xFFFF,0xFFF0,0x101F,0xFFFF,0xFFFF,0xFFFF,0xFFFF,0xFFFF,
  0xFFFF,0xFFFF,0xFFE0,0x203F,0xFFFF,0xFFFF,0xFFFF,0xFFFF,0xFFFF,
  0xFFFF,0xFFFF,0xFFC0,0x407F,0xFFFF,0xFFFF,0xFFFF,0xFFFF,0xFFFF,
  0xFFFF,0xFFFF,0xFFC0,0x407F,0xFFFF,0xFFFF,0xFFFF,0xFFFF,0xFFFF,
  0xFFFF,0xFFFF,0xFF80,0x80FF,0xFFFF,0xFFFF,0xFFFF,0xFFFF,0xFFFF,
  0xFFFF,0xFFFF,0xFF7D,0x7DFF,0xFFFF,0xFFFF,0xFFFF,0xFFFF,0xFFFF,
  0xFFFF,0xFFFF,0xFF7D,0x7DFF,0xFFFF,0xFFFF,0xFFFF,0xFFFF,0xFFFF,
  0xFFFF,0xFFFF,0xFEFA,0xFBFF,0xFFFF,0xFFFF,0xFFFF,0xFFFF,0xFFFF,
  0xFFFF,0xFFFF,0xFDF5,0xF7FF,0xFFFF,0xFFFF,0xFFFF,0xFFFF,0xFFFF,
  0xFFFF,0xFFFF,0xFC04,0x07FF,0xFFFF,0xFFFF,0xFFFF,0xFFFF,0xFFFF,
  0xFFFF,0xFFFF,0xF808,0x0FFF,0xFFFF,0xFFFF,0xFFFF,0xFFFF,0xFFFF,
  0xFFFF,0xFFFF,0xF808,0x0FFF,0xFFFF,0xFFFF,0xFFFF,0xFFFF,0xFFFF,
  0xFFFF,0xFFFF,0xF010,0x1FFF,0xFFFF,0xFFFF,0xFFFF,0xFFFF,0xFFFF,
  0xFFFF,0xFFFF,0xE020,0x3FFF,0xFFFF,0xFFFF,0xFFFF,0xFFFF,0xFFFF,
  0xFFFF,0xFFFF,0xEFAF,0xBFFF,0xFFFF,0xFFFF,0xFFFF,0xFFFF,0xFFFF,
  0xFFFF,0xFFFF,0xDF5F,0x7FFF,0xFFFF,0xFFFF,0xFFFF,0xFFFF,0xFFFF,
  0xFFFF,0xFFFF,0xBEBE,0xFFFF,0xFFFF,0xFFFF,0xFFFF,0xFFFF,0xFFFF,
  0xFFFF,0xFFFF,0xBEBE,0xFFFF,0xFFFF,0xFFFF,0xFFFF,0xFFFF,0xFFFF,
  0xFFFF,0xFFFF,0x7D7D,0xFFFF,0xFFFF,0xFFFF,0xFFFF,0xFFFF,0xFFFF,
  0xFFFF,0xFFFE,0x0203,0xFFFF,0xFFFF,0xFFFF,0xFFFF,0xFFFF,0xFFFF,
  0xFFFF,0xFFFE,0x0203,0xFFFF,0xFFFF,0xFFFF,0xFFFF,0xFFFF,0xFFFF,
  0xFFFF,0xFFFC,0x0407,0xFFFF,0xFFFF,0xFFFF,0xFFFF,0xFFFF,0xFFFF,
  0xFFFF,0xFFF8,0x080F,0xFFFF,0xFFFF,0xFFFF,0xFFFF,0xFFFF,0xFFFF,
  0xFFFF,0xFFF8,0x080F,0x8FF8,0x00FF,0x0060,0x03FE,0x00FF,0xFC7F,
  0xFFFF,0xFFF0,0x101F,0x07FC,0x007E,0x0030,0x01F8,0x007F,0xF83F,
  0xFFFF,0xFFE0,0x203F,0x07FE,0x007E,0x0078,0x03E0,0x003F,0xF83F,
  0xFFFF,0xFFEF,0xAFBE,0x07FE,0x00FC,0x01F0,0x0F80,0xFE3F,0xF03F,
  0xFFFF,0xFFDF,0x5F7C,0x07FC,0x00F8,0x01F0,0x0F03,0xFE3F,0xE03F,
  0x0101,0xFFBE,0xBEF8,0x07FC,0x00F8,0x01F0,0x0E07,0xFF7F,0xC03F,
  0xBEBE,0xFFBE,0xBEF8,0x07FC,0x00F0,0x01F0,0x0C0F,0xFFFF,0xC03F,
  0xBEBE,0xFF7D,0x7DF0,0x07FC,0x00F0,0x03E0,0x181F,0xFFFF,0x803F,
  0xDF5F,0x7EFA,0xFBE0,0x07F8,0x00E0,0x03E0,0x101F,0xFFFF,0x003F,
  0xEFAF,0xBEFA,0xFBE0,0x07F8,0x80C2,0x03E0,0x003F,0xFFFF,0x003F,
  0xEFAF,0xBDF5,0xF7C0,0x07F8,0x80C0,0x03E0,0x003F,0xFFFE,0x003F,
  0xF7D7,0xDBEB,0xEF80,0x07F8,0x8084,0x07C0,0x203F,0xFFFC,0x003F,
  0xFBEB,0xEBEB,0xEF04,0x07F0,0x800C,0x07C0,0x007E,0x00F8,0x203F,
0xFBEB,0xE010,0x1F0C,0x07F1,0x8008,0x07C0,0x007F,0x0078,0x603F,
0xFC04,0x0010,0x1E00,0x07F1,0x8018,0x07C0,0x007F,0x80F0,0x003F,
0xFE02,0x0020,0x3C00,0x07F1,0x8038,0x0F80,0x607F,0x81E0,0x003F,
0xFF01,0x0040,0x7C00,0x07E1,0x8070,0x0F80,0x607F,0x81E0,0x003F,
0xFF01,0x1F5F,0x783C,0x07E3,0x8070,0x0F80,0x707F,0x81C1,0xE03F,
0xFF80,0xBEBE,0xF07C,0x07E3,0x80F0,0x0F80,0x703F,0x8183,0xE03F,
0xFFDF,0x7D7D,0xE0FC,0x07E3,0x81F0,0x1F00,0xF83F,0x8107,0xE03F,
0xFFDF,0x7D7D,0xE1FC,0x07C3,0x81E0,0x1F00,0xFC1F,0x810F,0xE03F,
0xFFEE,0x0203,0xC1FC,0x07C7,0x83E0,0x1F00,0xFE1F,0x000F,0xE03F,
0xFFF0,0x0407,0x01F0,0x0303,0x8780,0x0C00,0x7F00,0x000F,0x801F,
0xFFF0,0x0406,0x00E0,0x0001,0xC700,0x0000,0x3F80,0x0007,0x000F,
0xFFF8,0x080F,0x0070,0x0000,0xEF80,0x0000,0x1FE0,0x3803,0x8007,

/* BitPlane ONE */
0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,
0x0000,0x0000,0x0007,0xC7C0,0x0000,0x0000,0x0000,0x0000,0x0000,
0x0000,0x0000,0x0007,0xC7C0,0x0000,0x0000,0x0000,0x0000,0x0000,
0x0000,0x0000,0x000F,0x8F80,0x0000,0x0000,0x0000,0x0000,0x0000,
0x0000,0x0000,0x001F,0x1F00,0x0000,0x0000,0x0000,0x0000,0x0000,
0x0000,0x0000,0x001F,0x1F00,0x0000,0x0000,0x0000,0x0000,0x0000,
0x0000,0x0000,0x003E,0x3E00,0x0000,0x0000,0x0000,0x0000,0x0000,
0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,
0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,
0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,
0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,
0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,
0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,
0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,
0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,
0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,
0x0000,0x0000,0x0F8F,0x8000,0x0000,0x0000,0x0000,0x0000,0x0000,
0x0000,0x0000,0x1F1F,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,
0x0000,0x0000,0x3E3E,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,
0x0000,0x0000,0x3E3E,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,
0x0000,0x0000,0x7C7C,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,
0x0000,0x0000,0xF8F8,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,
0x0000,0x0000,0xF8F8,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,
0x0000,0x0001,0xF1F0,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,
0x0000,0x0003,0xE3E0,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,
0x0000,0x0003,0xE3E0,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,
0x0000,0x0007,0xC7C0,0x0800,0x0080,0x00C8,0x0600,0x0000,0x0040,
0x0000,0x000F,0x8F80,0x0800,0x0180,0x0380,0x1C01,0xFE40,0x0040,
0x0000,0x0000,0x0000,0x0800,0x0100,0x0200,0x1007,0x0040,0x0040,
0x0000,0x0000,0x0000,0x0800,0x0100,0x0200,0x100C,0x00C0,0x0040,
0x0000,0x0000,0x0000,0x0800,0x8100,0x0200,0x1018,0x0080,0x0040,
0x3E3E,0x0000,0x0000,0x0800,0x8100,0x0600,0x3030,0x0000,0x0040,
0x3E3E,0x0000,0x0000,0x0800,0x8102,0x0400,0x2020,0x0000,0x0040,
0x1F1F,0x0000,0x0000,0x0801,0x8106,0x0400,0x2060,0x0000,0x0040,
0x0F8F,0x8000,0x0000,0x0801,0x0104,0x0400,0x2040,0x0000,0x0040,
0x0F8F,0x8000,0x0004,0x0801,0x010C,0x0C00,0x6040,0x0000,0x2040,
0x07C7,0xC000,0x000C,0x0801,0x0118,0x0800,0x40C0,0x0000,0x6040,
0x03E3,0xE000,0x0018,0x0803,0x0110,0x0800,0x4080,0x0000,0xC040,
0x03E3,0xE000,0x0010,0x0802,0x0030,0x0800,0x4080,0x0180,0x8040,
0x0000,0x0000,0x0000,0x0802,0x0060,0x1800,0xC080,0x0300,0x0040,
0x0000,0x0000,0x0000,0x0802,0x00C0,0x1000,0x8080,0x0200,0x0040,
0x0000,0x0000,0x007C,0x0806,0x0080,0x1000,0x8080,0x0203,0xE040,
0x0000,0x1F1F,0x00C0,0x0804,0x0180,0x1000,0x8000,0x0206,0x0040,
```

```
0x0000,0x3E3E,0x0180,0x0804,0x0300,0x3001,0x8040,0x020C,0x0040,
0x0000,0x7C7C,0x0300,0x0804,0x0200,0x2001,0x0000,0x0218,0x0040,
0x0000,0x7C7C,0x0200,0x080C,0x0600,0x2001,0x0020,0x0210,0x0040,
0x0000,0xF8F8,0x0600,0x0808,0x0C00,0x2001,0x0000,0x0630,0x0040,
0x0005,0xF1F0,0x0000,0x0000,0x0800,0x0000,0x0000,0x1800,0x0000,
0x0005,0xF1F0,0x0000,0x0000,0x1800,0x0000,0x0040,0x7000,0x0000,
0x0000,0x0000,0xFF8F,0xFFFF,0x107F,0xFFFF,0xE01F,0xC7FC,0x7FF8,

/* BitPlane TWO */
0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,
0x0000,0x0000,0x0007,0xC7C0,0x0000,0x0000,0x0000,0x0000,0x0000,
0x0000,0x0000,0x0007,0xC7C0,0x0000,0x0000,0x0000,0x0000,0x0000,
0x0000,0x0000,0x000F,0x8F80,0x0000,0x0000,0x0000,0x0000,0x0000,
0x0000,0x0000,0x001F,0x1F00,0x0000,0x0000,0x0000,0x0000,0x0000,
0x0000,0x0000,0x001F,0x1F00,0x0000,0x0000,0x0000,0x0000,0x0000,
0x0000,0x0000,0x003E,0x3E00,0x0000,0x0000,0x0000,0x0000,0x0000,
0x0000,0x0000,0x007C,0x7C00,0x0000,0x0000,0x0000,0x0000,0x0000,
0x0000,0x0000,0x007C,0x7C00,0x0000,0x0000,0x0000,0x0000,0x0000,
0x0000,0x0000,0x00F8,0xF800,0x0000,0x0000,0x0000,0x0000,0x0000,
0x0000,0x0000,0x01F1,0xF000,0x0000,0x0000,0x0000,0x0000,0x0000,
0x0000,0x0000,0x01F1,0xF000,0x0000,0x0000,0x0000,0x0000,0x0000,
0x0000,0x0000,0x03E3,0xE000,0x0000,0x0000,0x0000,0x0000,0x0000,
0x0000,0x0000,0x03E3,0xE000,0x0000,0x0000,0x0000,0x0000,0x0000,
0x0000,0x0000,0x07C7,0xC000,0x0000,0x0000,0x0000,0x0000,0x0000,
0x0000,0x0000,0x0F8F,0x8000,0x0000,0x0000,0x0000,0x0000,0x0000,
0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,
0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,
0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,
0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,
0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,
0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,
0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,
0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,
0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,
0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,
0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,
0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,
0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,
0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,
0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,
0x3E3E,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,
0x3E3E,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,
0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,
0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,
0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,
0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,
0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,
0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,
0x01F1,0xF000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,
0x00F8,0xE000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,
0x007C,0x4000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,
0x007C,0x5F1F,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,
0x003E,0x3E3E,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,
0x001F,0x7C7C,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,
0x001F,0x7C7C,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,
0x000E,0xF8F8,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,
0x0005,0xF1F0,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,
  0x0005,0xF1F0,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,
  0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,

  /* BitPlane THREE */
  0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,
  0x0000,0x0000,0x0007,0xC7C0,0x0000,0x0000,0x0000,0x0000,0x0000,
  0x0000,0x0000,0x0007,0xC7C0,0x0000,0x0000,0x0000,0x0000,0x0000,
  0x0000,0x0000,0x000F,0x8F80,0x0000,0x0000,0x0000,0x0000,0x0000,
  0x0000,0x0000,0x001F,0x1F00,0x0000,0x0000,0x0000,0x0000,0x0000,
  0x0000,0x0000,0x001F,0x1F00,0x0000,0x0000,0x0000,0x0000,0x0000,
  0x0000,0x0000,0x003E,0x3E00,0x0000,0x0000,0x0000,0x0000,0x0000,
  0x0000,0x0000,0x007C,0x7C00,0x0000,0x0000,0x0000,0x0000,0x0000,
  0x0000,0x0000,0x007C,0x7C00,0x0000,0x0000,0x0000,0x0000,0x0000,
  0x0000,0x0000,0x00F8,0xF800,0x0000,0x0000,0x0000,0x0000,0x0000,
  0x0000,0x0000,0x01F1,0xF000,0x0000,0x0000,0x0000,0x0000,0x0000,
  0x0000,0x0000,0x01F1,0xF000,0x0000,0x0000,0x0000,0x0000,0x0000,
  0x0000,0x0000,0x03E3,0xE000,0x0000,0x0000,0x0000,0x0000,0x0000,
  0x0000,0x0000,0x03E3,0xE000,0x0000,0x0000,0x0000,0x0000,0x0000,
  0x0000,0x0000,0x07C7,0xC000,0x0000,0x0000,0x0000,0x0000,0x0000,
  0x0000,0x0000,0x0F8F,0x8000,0x0000,0x0000,0x0000,0x0000,0x0000,
  0x0000,0x0000,0x0F8F,0x8000,0x0000,0x0000,0x0000,0x0000,0x0000,
  0x0000,0x0000,0x1F1F,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,
  0x0000,0x0000,0x3E3E,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,
  0x0000,0x0000,0x3E3E,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,
  0x0000,0x0000,0x7C7C,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,
  0x0000,0x0000,0xF8F8,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,
  0x0000,0x0000,0xF8F8,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,
  0x0000,0x0001,0xF1F0,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,
  0x0000,0x0003,0xE3E0,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,
  0x0000,0x0003,0xE3E0,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,
  0x0000,0x0007,0xC7C0,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,
  0x0000,0x000F,0x8F80,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,
  0x0000,0x000F,0x8F80,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,
  0x0000,0x001F,0x1F00,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,
  0x0000,0x003E,0x3E00,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,
  0x3E3E,0x003E,0x3E00,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,
  0x3E3E,0x007C,0x7C00,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,
  0x0000,0x00F8,0xF800,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,
  0x0000,0x00F8,0xF800,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,
  0x0000,0x01F1,0xF000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,
  0x0000,0x03E3,0xE000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,
  0x0000,0x03E3,0xE000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,
  0x0000,0x07C7,0xC000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,
  0x0000,0x07C7,0xC000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,
  0x0000,0x0F8F,0x8000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,
  0x0000,0x1F1F,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,
  0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,
  0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,
  0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,
  0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,
  0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,
  0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,
  0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,
  0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000
};
```

```
/* Image structure for the Amiga Logo image: */
struct Image amiga_logo=
{
  0, 0,             /* LeftEdge, TopEdge. */
  141,              /* Width, 141 pixels wide. */
  50,               /* Height, 50 lines high. */
  4,                /* Depth, four Bitplanes, 16 colours. */
  amiga_logo_data, /* ImageData, pointer to king_tut_data. */
  0x000F,           /* PickPlane, Bitplane 0,1,2 and 3 affected. */
  0x0000,           /* PlaneOnOff, all Bitplanes are already "picked". */
  NULL              /* NextImage, no more Images. */
};



main()
{
  /* Open the Intuition Library: */
  IntuitionBase = (struct IntuitionBase *)
    OpenLibrary( "intuition.library", 0 );

  if( IntuitionBase == NULL )
    exit(); /* Could NOT open the Intuition Library! */



  /* Open the Graphics Library: */
  GfxBase = (struct GfxBase *)
    OpenLibrary( "graphics.library", 0 );

  if( GfxBase == NULL )
  {
    /* Could NOT open the Graphics Library! */

    /* Close the Intuition Library since we have opened it: */
    CloseLibrary( IntuitionBase );

    exit();
  }



  /* We will now try to open the screen: */
  my_screen= (struct Screen *) OpenScreen( &my_new_screen );

  /* Have we opened the screen succesfully? */
  if(my_screen == NULL)
  {
    /* Could NOT open the Screen! */

    /* Close the Graphics Library since we have opened it: */
    CloseLibrary( GfxBase );

    /* Close the Intuition Library since we have opened it: */
    CloseLibrary( IntuitionBase );
  exit();
}



/******************************************************************/
/* We are here using the function SetRGB4() to change the Custom  */
/* screen's colour. You should not bother about this function yet */
/* since it has nothing to do with Intuition. It is actually a    */
/* low-level graphic function (that is why we needed to open the  */
/* Graphics Library) which will be discussed later.               */
/******************************************************************/

SetRGB4( &my_screen->ViewPort,  0, 0x0, 0x0, 0x0 );
SetRGB4( &my_screen->ViewPort,  1, 0xF, 0xE, 0xD );
SetRGB4( &my_screen->ViewPort,  2, 0x9, 0x9, 0x9 );
SetRGB4( &my_screen->ViewPort,  3, 0x0, 0x0, 0xF );
SetRGB4( &my_screen->ViewPort,  4, 0x0, 0x8, 0xF );
SetRGB4( &my_screen->ViewPort,  5, 0x0, 0xB, 0xA );
SetRGB4( &my_screen->ViewPort,  6, 0x0, 0xF, 0x0 );
SetRGB4( &my_screen->ViewPort,  7, 0xA, 0xF, 0x0 );

SetRGB4( &my_screen->ViewPort,  8, 0xD, 0xF, 0x0 );
SetRGB4( &my_screen->ViewPort,  9, 0xF, 0xF, 0x0 );
SetRGB4( &my_screen->ViewPort, 10, 0xF, 0xC, 0x0 );
SetRGB4( &my_screen->ViewPort, 11, 0xF, 0xA, 0x0 );
SetRGB4( &my_screen->ViewPort, 12, 0xF, 0x7, 0x0 );
SetRGB4( &my_screen->ViewPort, 13, 0xF, 0x5, 0x0 );
SetRGB4( &my_screen->ViewPort, 14, 0xF, 0x2, 0x0 );
SetRGB4( &my_screen->ViewPort, 15, 0x8, 0x0, 0xF );



/* Before we can open the window we need to give the NewWindow */
/* structure a pointer to the opened Custom Screen: */
my_new_window.Screen = my_screen;



/* We will now try to open the window: */
my_window = (struct Window *) OpenWindow( &my_new_window );

/* Have we opened the window succesfully? */
if(my_window == NULL)
{
  /* Could NOT open the Window! */

  /* Close the screen since we have opened it: */
  CloseScreen( my_screen );

  /* Close the Graphics Library since we have opened it: */
  CloseLibrary( GfxBase );

  /* Close the Intuition Library since we have opened it: */
  CloseLibrary( IntuitionBase );

  exit();
```

```
  }



  /* Tell Intuition to draw the Amiga Logo Image: */
  DrawImage( my_window->RPort, &amiga_logo, 3, 11 );



  /* Wait for 30 seconds: */
  Delay( 50 * 30);



  /* We should always close the windows we have opened before we leave: */
  CloseWindow( my_window );

  /* Remember that all windows connected to a screen must be closed */
  /* before you may close the screen! */
  CloseScreen( my_screen );



  /* Close the Graphics Library since we have opened it: */
  CloseLibrary( GfxBase );

  /* Close the Intuition Library since we have opened it: */
  CloseLibrary( IntuitionBase );

  /* THE END */
}
```

A.4 GADGETS

Example1

This program will open a normal window which is connected to the Workbench Screen. The window will use all System Gadgets, and will close first when the user has selected the System gadget Close window. (Same as Example3 in chapter 2 WINDOWS, except that we have added an IDCMP check on the Close window gadget.)

```
/* Example1                                                         */
/* This program will open a normal window which is connected to the    */
/* Workbench Screen. The window will use all System Gadgets, and will  */
/* close first when the user has selected the System gadget Close      */
/* window. (Same as Example3 in chapter 2 WINDOWS, except that we have */
/* added an IDCMP check on the Close window gadget.)                   */

/* Extra information:                                                  */
/* This program will quit first when the user has selected the Close     */
/* window gadget. To tell Intuition that we want the System gadget Close */
/* window to send us a message when the user selects it, we only need to */
/* set the CLOSEWINDOW flag in the IDCMPFlags field.                     */



#include <intuition/intuition.h>



struct IntuitionBase *IntuitionBase;



/* Declare a pointer to a Window structure: */
struct Window *my_window;

/* Declare and initialize your NewWindow structure: */
struct NewWindow my_new_window=
{
  50,            /* LeftEdge    x position of the window. */
  25,            /* TopEdge     y positio of the window. */
  200,           /* Width       200 pixels wide. */
  100,           /* Height      100 lines high. */
  0,             /* DetailPen   Text should be drawn with colour reg. 0 */
  1,             /* BlockPen    Blocks should be drawn with colour reg. 1 */
  CLOSEWINDOW,   /* IDCMPFlags  The window will give us a message if the */
                 /*             user has selected the Close window gad. */
  SMART_REFRESH| /* Flags       Intuition should refresh the window. */
  WINDOWCLOSE|   /*             Close Gadget. */
  WINDOWDRAG|    /*             Drag gadget. */
  WINDOWDEPTH|   /*             Depth arrange Gadgets. */
  WINDOWSIZING|  /*             Sizing Gadget. */
  ACTIVATE,      /*             The window should be Active when opened. */
  NULL,          /* FirstGadget No Custom gadgets. */
  NULL,          /* CheckMark   Use Intuition's default CheckMark. */
  "CLOSE ME",    /* Title       Title of the window. */
  NULL,          /* Screen      Connected to the Workbench Screen. */
  NULL,          /* BitMap      No Custom BitMap. */
  80,            /* MinWidth    We will not allow the window to become */
  30,            /* MinHeight   smaller than 80 x 30, and not bigger */
  300,           /* MaxWidth    than 300 x 200. */
  200,           /* MaxHeight */
  WBENCHSCREEN   /* Type        Connected to the Workbench Screen. */
};



main()
{
  /* Boolean variable used for the while loop: */
  BOOL close_me;
```

```
  /* Declare a variable in which we will store the IDCMP flag: */
  ULONG class;

  /* Declare a pointer to an IntuiMessage structure: */
  struct IntuiMessage *my_message;



  /* Before we can use Intuition we need to open the Intuition Library: */
  IntuitionBase = (struct IntuitionBase *)
    OpenLibrary( "intuition.library", 0 );

  if( IntuitionBase == NULL )
    exit(); /* Could NOT open the Intuition Library! */



  /* We will now try to open the window: */
  my_window = (struct Window *) OpenWindow( &my_new_window );

  /* Have we opened the window succesfully? */
  if(my_window == NULL)
  {
    /* Could NOT open the Window! */

    /* Close the Intuition Library since we have opened it: */
    CloseLibrary( IntuitionBase );

    exit();
  }



  /* We have opened the window, and everything seems to be OK. */



  close_me = FALSE;

  /* Stay in the while loop until the user has selected the Close window */
  /* gadget: */
  while( close_me == FALSE )
  {
    /* Wait until we have recieved a message: */
    Wait( 1 << my_window->UserPort->mp_SigBit );

    /* Collect the message: */
    my_message = (struct IntuiMessage *) GetMsg( my_window->UserPort );
    /* Ahhh, these paranoid compilers... */

    /* Have we collected the message sucessfully? */
    if(my_message)
    {
      /* After we have collected the message we can read it, and save any */
      /* important values which we maybe want to check later: */
      class = my_message->Class;

      /* After we have read it we reply as fast as possible: */
      /* REMEMBER! Do never try to read a message after you have replied! */
      /* Some other process has maybe changed it. */
      ReplyMsg( my_message );
      /* Check which IDCMP flag was sent: */
      if( class == CLOSEWINDOW )
        close_me=TRUE; /* The user selected the Close window gadget! */
    }
  }



  /* We should always close the windows we have opened before we leave: */
  CloseWindow( my_window );



  /* Close the Intuition Library since we have opened it: */
  CloseLibrary( IntuitionBase );

  /* THE END */
}
```

Example2

Same as Example1 except that we have added a Boolean gadget
with the text "PRESS ME".

```
/* Example2                                                      */
/* This program will open a normal window which is connected to the     */
/* Workbench Screen. The window will use all System Gadgets, and will   */
/* close first when the user has selected the System gadget Close       */
/* window. Inside the window we have put a Boolean gadget with the text */
/* "PRESS ME".                                                          */



#include <intuition/intuition.h>



struct IntuitionBase *IntuitionBase;



/* The coordinates for the box: */
SHORT my_points[]=
{
   0,  0, /* Start at position (0,0) */
  70,  0, /* Draw a line to the right to position (70,0) */
  70, 10, /* Draw a line down to position (70,10) */
   0, 10, /* Draw a line to the right to position (0,10) */
   0,  0  /* Finish of by drawing a line up to position (0,0) */
};

/* The Border structure: */
struct Border my_border=
{
  0, 0,        /* LeftEdge, TopEdge. */
  1,           /* FrontPen, colour register 1. */
  0,           /* BackPen, for the moment unused. */
  JAM1,        /* DrawMode, draw the lines with colour 1. */
  5,           /* Count, 5 pair of coordinates in the array. */
  my_points,   /* XY, pointer to the array with the coordinates. */
  NULL,        /* NextBorder, no other Border structures are connected. */
};



/* The text string: */
UBYTE my_string[]="PRESS ME";

/* The IntuiText structure: */
struct IntuiText my_text=
{
  1,         /* FrontPen, colour register 1. */
  0,         /* BackPen, colour register 0. */
  JAM1,      /* DrawMode, draw the characters with colour 1, do not */
             /* change the background. */
  4, 2,      /* LeftEdge, TopEdge. */
  NULL,      /* ITextFont, use default font. */
  my_string, /* IText, the text that will be printed. */
             /* (Remember my_text = &my_text[0].) */
  NULL,      /* NextText, no other IntuiText structures are connected. */
};



struct Gadget my_gadget=
{
  NULL,          /* NextGadget, no more gadgets in the list. */
  40,            /* LeftEdge, 40 pixels out. */
  20,            /* TopEdge, 20 lines down. */
  71,            /* Width, 71 pixels wide. */
  11,            /* Height, 11 pixels lines heigh. */
  GADGHCOMP,     /* Flags, when this gadget is highlighted, the gadget */
                 /* will be rendered in the complement colours. */
                 /* (Colour 0 (00) will be changed to colour 3 (11) */
                 /* (Colour 1 (01)           - " -           2 (10) */
                 /* (Colour 2 (10)           - " -           1 (01) */
                 /* (Colour 3 (11)           - " -           0 (00) */
  GADGIMMEDIATE| /* Activation, our program will recieve a message when */
  RELVERIFY,     /* the user has selected this gadget, and when the user */
                 /* has released it. */
  BOOLGADGET,    /* GadgetType, a Boolean gadget. */
  (APTR) &my_border, /* GadgetRender, a pointer to our Border structure. */
                 /* (Since Intuition does not know if this will be a */
                 /* pointer to a Border structure or an Image structure, */
                 /* Intuition expects an APTR (normal memory pointer). */
                 /* We will therefore have to calm down the compiler by */
                 /* doing some "casting".) */
  NULL,          /* SelectRender, NULL since we do not supply the gadget */
                 /* with an alternative image. (We complement the */
                 /* colours instead) */
  &my_text,      /* GadgetText, a pointer to our IntuiText structure. */
                 /* (See chapter 3 GRAPHICS for more information) */
  NULL,          /* MutualExclude, no mutual exclude. */
  NULL,          /* SpecialInfo, NULL since this is a Boolean gadget. */
                 /* (It is not a Proportional/String or Integer gdget) */
  0,             /* GadgetID, no id. */
  NULL           /* UserData, no user data connected to the gadget. */
};



/* Declare a pointer to a Window structure: */
struct Window *my_window;

/* Declare and initialize your NewWindow structure: */
struct NewWindow my_new_window=
{
  50,            /* LeftEdge    x position of the window. */
  25,            /* TopEdge     y positio of the window. */
  200,           /* Width       200 pixels wide. */
  100,           /* Height      100 lines high. */
  0,             /* DetailPen   Text should be drawn with colour reg. 0 */
  1,             /* BlockPen    Blocks should be drawn with colour reg. 1 */
  CLOSEWINDOW|   /* IDCMPFlags  The window will give us a message if the */
                 /*             user has selected the Close window gad, */
  GADGETDOWN|    /*             or a gadget has been pressed on, or */
  GADGETUP,      /*             a gadge has been released. */
  SMART_REFRESH| /* Flags       Intuition should refresh the window. */
  WINDOWCLOSE|   /*             Close Gadget. */
  WINDOWDRAG|    /*             Drag gadget. */
  WINDOWDEPTH|   /*             Depth arrange Gadgets. */
  WINDOWSIZING|  /*             Sizing Gadget. */
  ACTIVATE,      /*             The window should be Active when opened. */
  &my_gadget,    /* FirstGadget A pointer to my_gadget structure. */
  NULL,          /* CheckMark   Use Intuition's default CheckMark. */
  "TOUCH ME",    /* Title       Title of the window. */
  NULL,          /* Screen      Connected to the Workbench Screen. */
  NULL,          /* BitMap      No Custom BitMap. */
  140,           /* MinWidth    We will not allow the window to become */
  50,            /* MinHeight   smaller than 140 x 50, and not bigger */
  300,           /* MaxWidth    than 300 x 200. */
  200,           /* MaxHeight */
  WBENCHSCREEN   /* Type        Connected to the Workbench Screen. */
};



/***********************************************************************/
/* Extra information:                                                  */
/* You first need to decide what messages the gadgets should report.   */
/* In this case we told the Boolean gadget to send a message if the user */
/* pressed on it, and if the user released the gadget while still      */
/* pointing at it. (We sat the flags GADGIMMEDIATE and RELVERIFY)      */
/*                                                                     */
/* The important thing to remember is that we need to tell the window  */
/* what messages should be allowed to pass by. It was therefore we     */
/* needed to set the IDCMP flags GADGETUP and GADGETDOWN in the        */
/* IDCMPFlags field in the NewWindow structure.                        */
/***********************************************************************/



main()
{
  /* Boolean variable used for the while loop: */
  BOOL close_me;

  /* Declare a variable in which we will store the IDCMP flag: */
  ULONG class;

  /* Declare a pointer to an IntuiMessage structure: */
  struct IntuiMessage *my_message;



  /* Before we can use Intuition we need to open the Intuition Library: */
  IntuitionBase = (struct IntuitionBase *)
    OpenLibrary( "intuition.library", 0 );

  if( IntuitionBase == NULL )
    exit(); /* Could NOT open the Intuition Library! */



  /* We will now try to open the window: */
  my_window = (struct Window *) OpenWindow( &my_new_window );

  /* Have we opened the window succesfully? */
  if(my_window == NULL)
  {
    /* Could NOT open the Window! */

    /* Close the Intuition Library since we have opened it: */
    CloseLibrary( IntuitionBase );

    exit();
  }



  /* We have opened the window, and everything seems to be OK. */



  close_me = FALSE;

  /* Stay in the while loop until the user has selected the Close window */
  /* gadget: */
  while( close_me == FALSE )
  {
    /* Wait until we have recieved a message: */
    Wait( 1 << my_window->UserPort->mp_SigBit );

    /* Collect the message: */
    my_message = (struct IntuiMessage *) GetMsg( my_window->UserPort );

    /* Have we collected the message sucessfully? */
    if(my_message)
    {
      /* After we have collected the message we can read it, and save any */
      /* important values which we maybe want to check later: */
      class = my_message->Class;

      /* After we have read it we reply as fast as possible: */
      /* REMEMBER! Do never try to read a message after you have replied! */
      /* Some other process has maybe changed it. */
      ReplyMsg( my_message );

      /* Check which IDCMP flag was sent: */
      switch( class )
      {
        case CLOSEWINDOW:  /* The user selected the Close window gadget! */
               close_me=TRUE;
               break;

        case GADGETDOWN:   /* The user has pressed on the Boolean gadget. */
               printf("Down\n");
               break;

        case GADGETUP:     /* The user has released the Boolean gadget. */
               printf("Up\n");
               break;
      }
    }
  }



  /* We should always close the windows we have opened before we leave: */
  CloseWindow( my_window );



  /* Close the Intuition Library since we have opened it: */
  CloseLibrary( IntuitionBase );

  /* THE END */
}
```

Example3

Same as Example2 except that the on/off state of the gadget is toggled each time the user hits the gadget.

```
/* Example3                                                        */
/* This program will open a normal window which is connected to the     */
/* Workbench Screen. The window will use all System Gadgets, and will   */
/* close first when the user has selected the System gadget Close       */
/* window. Inside the window we have put a Boolean gadget with the text */
/* "PRESS ME". The on/off state of the gadget is toggled each time the  */
/* user hitts the gadget.                                               */



#include <intuition/intuition.h>



struct IntuitionBase *IntuitionBase;



/* The coordinates for the box: */
SHORT my_points[]=
{
   0,  0, /* Start at position (0,0) */
  70,  0, /* Draw a line to the right to position (70,0) */
  70, 10, /* Draw a line down to position (70,10) */
   0, 10, /* Draw a line to the right to position (0,10) */
   0,  0  /* Finish of by drawing a line up to position (0,0) */
};

/* The Border structure: */
struct Border my_border=
{
  0, 0,        /* LeftEdge, TopEdge. */
  1,           /* FrontPen, colour register 1. */
  0,           /* BackPen, for the moment unused. */
  JAM1,        /* DrawMode, draw the lines with colour 1. */
  5,           /* Count, 5 pair of coordinates in the array. */
  my_points,   /* XY, pointer to the array with the coordinates. */
  NULL,        /* NextBorder, no other Border structures are connected. */
};



/* The text string: */
UBYTE my_string[]="PRESS ME";

/* The IntuiText structure: */
struct IntuiText my_text=
{
  1,         /* FrontPen, colour register 1. */
  0,         /* BackPen, colour register 0. */
  JAM1,      /* DrawMode, draw the characters with colour 1, do not */
             /* change the background. */
  4, 2,      /* LeftEdge, TopEdge. */
  NULL,      /* ITextFont, use default font. */
  my_string, /* IText, the text that will be printed. */
             /* (Remember my_text = &my_text[0].) */
  NULL,      /* NextText, no other IntuiText structures are connected. */
};



struct Gadget my_gadget=
{
  NULL,          /* NextGadget, no more gadgets in the list. */
  40,            /* LeftEdge, 40 pixels out. */
  20,            /* TopEdge, 20 lines down. */
  71,            /* Width, 71 pixels wide. */
  11,            /* Height, 11 pixels lines heigh. */
  GADGHCOMP,     /* Flags, when this gadget is highlighted, the gadget */
                 /* will be rendered in the complement colours. */
                 /* (Colour 0 (00) will be changed to colour 3 (11) */
                 /* (Colour 1 (01)           - " -           2 (10) */
                 /* (Colour 2 (10)           - " -           1 (01) */
                 /* (Colour 3 (11)           - " -           0 (00) */
  GADGIMMEDIATE| /* Activation, our program will recieve a message when */
                 /* the user has selected this gadget. */
  TOGGLESELECT,  /* The on/off state of the gadget is toggled each time. */
  BOOLGADGET,    /* GadgetType, a Boolean gadget. */
  (APTR) &my_border, /* GadgetRender, a pointer to our Border structure. */
                 /* (Since Intuition does not know if this will be a */
                 /* pointer to a Border structure or an Image structure, */
                 /* Intuition expects an APTR (normal memory pointer). */
                 /* We will therefore have to calm down the compiler by */
                 /* doing some "casting".) */
  NULL,          /* SelectRender, NULL since we do not supply the gadget */
                 /* with an alternative image. (We complement the */
                 /* colours instead) */
  &my_text,      /* GadgetText, a pointer to our IntuiText structure. */
                 /* (See chapter 3 GRAPHICS for more information) */
  NULL,          /* MutualExclude, no mutual exclude. */
  NULL,          /* SpecialInfo, NULL since this is a Boolean gadget. */
                 /* (It is not a Proportional/String or Integer gdget) */
  0,             /* GadgetID, no id. */
  NULL           /* UserData, no user data connected to the gadget. */
};



/* Declare a pointer to a Window structure: */
struct Window *my_window;

/* Declare and initialize your NewWindow structure: */
struct NewWindow my_new_window=
{
  50,            /* LeftEdge    x position of the window. */
  25,            /* TopEdge     y positio of the window. */
  200,           /* Width       200 pixels wide. */
  100,           /* Height      100 lines high. */
  0,             /* DetailPen   Text should be drawn with colour reg. 0 */
  1,             /* BlockPen    Blocks should be drawn with colour reg. 1 */
  CLOSEWINDOW|   /* IDCMPFlags  The window will give us a message if the */
                 /*             user has selected the Close window gad, */
  GADGETDOWN,    /*             or a gadget has been pressed on. */
  SMART_REFRESH| /* Flags       Intuition should refresh the window. */
  WINDOWCLOSE|   /*             Close Gadget. */
  WINDOWDRAG|    /*             Drag gadget. */
  WINDOWDEPTH|   /*             Depth arrange Gadgets. */
  WINDOWSIZING|  /*             Sizing Gadget. */
  ACTIVATE,      /*             The window should be Active when opened. */
  &my_gadget,    /* FirstGadget A pointer to my_gadget structure. */
  NULL,          /* CheckMark   Use Intuition's default CheckMark. */
  "TOGGLE ME",   /* Title       Title of the window. */
  NULL,          /* Screen      Connected to the Workbench Screen. */
  NULL,          /* BitMap      No Custom BitMap. */
  140,           /* MinWidth    We will not allow the window to become */
  50,            /* MinHeight   smaller than 140 x 50, and not bigger */
  300,           /* MaxWidth    than 300 x 200. */
  200,           /* MaxHeight */
  WBENCHSCREEN   /* Type        Connected to the Workbench Screen. */
};



/***********************************************************************/
/* Extra information:                                                  */
/* You first need to decide what messages the gadgets should report.   */
/* In this case we told the Boolean gadget to send a message if the user */
/* pressed on it. (We sat the flag GADGIMMEDIATE) (The Close window    */
/* gadget will always send a message if someone has selected it.)      */
/* The important thing to remember is that we need to tell the window  */
/* what messages should be allowed to pass by. It was therefore we     */
/* needed to set the IDCMP flags GADGETDOWN and CLOSEWINDOW in the     */
/* NewWindow structure.                                                */
/***********************************************************************/



main()
{
  /* Boolean variable used for the while loop: */
  BOOL close_me;

  /* Declare a variable in which we will store the IDCMP flag: */
  ULONG class;

  /* Declare a pointer to an IntuiMessage structure: */
  struct IntuiMessage *my_message;



  /* Before we can use Intuition we need to open the Intuition Library: */
  IntuitionBase = (struct IntuitionBase *)
    OpenLibrary( "intuition.library", 0 );

  if( IntuitionBase == NULL )
    exit(); /* Could NOT open the Intuition Library! */



  /* We will now try to open the window: */
  my_window = (struct Window *) OpenWindow( &my_new_window );

  /* Have we opened the window succesfully? */
  if(my_window == NULL)
  {
    /* Could NOT open the Window! */

    /* Close the Intuition Library since we have opened it: */
    CloseLibrary( IntuitionBase );

    exit();
  }



  /* We have opened the window, and everything seems to be OK. */
```

```
  close_me = FALSE;

  /* Stay in the while loop until the user has selected the Close window */
  /* gadget: */
  while( close_me == FALSE )
  {
    /* Wait until we have recieved a message: */
    Wait( 1 << my_window->UserPort->mp_SigBit );

    /* Collect the message: */
    my_message = (struct IntuiMessage *) GetMsg( my_window->UserPort );

    /* Have we collected the message sucessfully? */
    if(my_message)
    {
      /* After we have collected the message we can read it, and save any */
      /* important values which we maybe want to check later: */
      class = my_message->Class;

      /* After we have read it we reply as fast as possible: */
      /* REMEMBER! Do never try to read a message after you have replied! */
      /* Some other process has maybe changed it. */
      ReplyMsg( my_message );

      /* Check which IDCMP flag was sent: */
      switch( class )
      {
        case CLOSEWINDOW:  /* The user selected the Close window gadget! */
               close_me=TRUE;
               break;

        case GADGETDOWN:   /* The user has pressed on the Boolean gadget. */
               printf("Hit\n");
               break;
      }
    }
  }



  /* We should always close the windows we have opened before we leave: */
  CloseWindow( my_window );



  /* Close the Intuition Library since we have opened it: */
  CloseLibrary( IntuitionBase );

  /* THE END */
}
```

Example4

This program will open a normal window which is connected to the Workbench Screen. The window will use all System Gadgets, and will close first when the user has selected the System gadget Close window. Inside the window we have put two Boolean gadgets with the text "GADGET 1" and "GADGET 2".

```
/* Example4                                                       */
/* This program will open a normal window which is connected to the   */
/* Workbench Screen. The window will use all System Gadgets, and will */
/* close first when the user has selected the System gadget Close     */
/* window. Inside the window we have put two Boolean gadgets with the */
/* text "GADGET 1" and "GADGET 2".                                     */



#include <intuition/intuition.h>



struct IntuitionBase *IntuitionBase;



/* The coordinates for the box: */
SHORT my_points[]=
{
   0,  0, /* Start at position (0,0) */
  70,  0, /* Draw a line to the right to position (70,0) */
  70, 10, /* Draw a line down to position (70,10) */
   0, 10, /* Draw a line to the right to position (0,10) */
   0,  0  /* Finish of by drawing a line up to position (0,0) */
};

/* The Border structure: */
struct Border my_border=
{
  0, 0,        /* LeftEdge, TopEdge. */
  1,           /* FrontPen, colour register 1. */
  0,           /* BackPen, for the moment unused. */
  JAM1,        /* DrawMode, draw the lines with colour 1. */
  5,           /* Count, 5 pair of coordinates in the array. */
  my_points,   /* XY, pointer to the array with the coordinates. */
  NULL,        /* NextBorder, no other Border structures are connected. */
};

/* We can use the same Border structure for both of the gadgets since */
/* they have the same size. */



/************/
/* GADGET 1 */
/************/

/* The first text string: */
UBYTE my_first_string[]="GADGET 1";

/* The IntuiText structure: */
struct IntuiText my_first_text=
{
  1,         /* FrontPen, colour register 1. */
  0,         /* BackPen, colour register 0. */
  JAM1,      /* DrawMode, draw the characters with colour 1, do not */
             /* change the background. */
  4, 2,      /* LeftEdge, TopEdge. */
  NULL,      /* ITextFont, use default font. */
  my_first_string, /* IText, the text that will be printed. */
             /* (Remember my_text = &my_text[0].) */
  NULL,      /* NextText, no other IntuiText structures are connected. */
};

struct Gadget my_first_gadget=
{
  NULL,          /* NextGadget, no more gadgets in the list. */
  40,            /* LeftEdge, 40 pixels out. */
  20,            /* TopEdge, 20 lines down. */
  71,            /* Width, 71 pixels wide. */
  11,            /* Height, 11 pixels lines heigh. */
  GADGHCOMP,     /* Flags, when this gadget is highlighted, the gadget */
                 /* will be rendered in the complement colours. */
                 /* (Colour 0 (00) will be changed to colour 3 (11) */
                 /* (Colour 1 (01)           - " -           2 (10) */
                 /* (Colour 2 (10)           - " -           1 (01) */
                 /* (Colour 3 (11)           - " -           0 (00) */
  GADGIMMEDIATE| /* Activation, our program will recieve a message when */
  RELVERIFY,     /* the user has selected this gadget, and when the user */
                 /* has released it. */
  BOOLGADGET,    /* GadgetType, a Boolean gadget. */
  (APTR) &my_border, /* GadgetRender, a pointer to our Border structure. */
                 /* (Since Intuition does not know if this will be a */
                 /* pointer to a Border structure or an Image structure, */
                 /* Intuition expects an APTR (normal memory pointer). */
                 /* We will therefore have to calm down the compiler by */
                 /* doing some "casting".) */
  NULL,          /* SelectRender, NULL since we do not supply the gadget */
                 /* with an alternative image. (We complement the */
                 /* colours instead) */
  &my_first_text,/* GadgetText, a pointer to our IntuiText structure. */
                 /* (See chapter 3 GRAPHICS for more information) */
  NULL,          /* MutualExclude, no mutual exclude. */
  NULL,          /* SpecialInfo, NULL since this is a Boolean gadget. */
                 /* (It is not a Proportional/String or Integer gdget) */
  0,             /* GadgetID, no id. */
  NULL           /* UserData, no user data connected to the gadget. */
};



/************/
/* GADGET 2 */
/************/

/* The second text string: */
UBYTE my_second_string[]="GADGET 2";

/* The IntuiText structure: */
struct IntuiText my_second_text=
{
  1,         /* FrontPen, colour register 1. */
  0,         /* BackPen, colour register 0. */
  JAM1,      /* DrawMode, draw the characters with colour 1, do not */
             /* change the background. */
  4, 2,      /* LeftEdge, TopEdge. */
  NULL,      /* ITextFont, use default font. */
  my_second_string, /* IText, the text that will be printed. */
             /* (Remember my_text = &my_text[0].) */
  NULL,      /* NextText, no other IntuiText structures are connected. */
};

struct Gadget my_second_gadget=
{
  &my_first_gadget, /* NextGadget, after this comes my_first_gadget. */
  150,            /* LeftEdge, 150 pixels out. */
  20,             /* TopEdge, 20 lines down. */
  71,             /* Width, 71 pixels wide. */
  11,             /* Height, 11 pixels lines heigh. */
  GADGHCOMP,      /* Flags, when this gadget is highlighted, the gadget */
                  /* will be rendered in the complement colours. */
                  /* (Colour 0 (00) will be changed to colour 3 (11) */
                  /* (Colour 1 (01)           - " -           2 (10) */
                  /* (Colour 2 (10)           - " -           1 (01) */
                  /* (Colour 3 (11)           - " -           0 (00) */
  GADGIMMEDIATE|  /* Activation, our program will recieve a message when */
  RELVERIFY,      /* the user has selected this gadget, and when the user */
                  /* has released it. */
  BOOLGADGET,     /* GadgetType, a Boolean gadget. */
  (APTR) &my_border, /* GadgetRender, a pointer to our Border structure. */
                  /* (Since Intuition does not know if this will be a */
                  /* pointer to a Border structure or an Image structure, */
                  /* Intuition expects an APTR (normal memory pointer). */
                  /* We will therefore have to calm down the compiler by */
                  /* doing some "casting".) */
  NULL,           /* SelectRender, NULL since we do not supply the gadget */
                  /* with an alternative image. (We complement the */
                  /* colours instead) */
  &my_second_text,/* GadgetText, a pointer to our IntuiText structure. */
                  /* (See chapter 3 GRAPHICS for more information) */
  NULL,           /* MutualExclude, no mutual exclude. */
  NULL,           /* SpecialInfo, NULL since this is a Boolean gadget. */
                  /* (It is not a Proportional/String or Integer gdget) */
  0,              /* GadgetID, no id. */
  NULL            /* UserData, no user data connected to the gadget. */
};



/* Declare a pointer to a Window structure: */
struct Window *my_window;

/* Declare and initialize your NewWindow structure: */
struct NewWindow my_new_window=
{
  50,            /* LeftEdge    x position of the window. */
  25,            /* TopEdge     y positio of the window. */
  320,           /* Width       320 pixels wide. */
  100,           /* Height      100 lines high. */
  0,             /* DetailPen   Text should be drawn with colour reg. 0 */
  1,             /* BlockPen    Blocks should be drawn with colour reg. 1 */
  CLOSEWINDOW|   /* IDCMPFlags  The window will give us a message if the */
                 /*             user has selected the Close window gad, */
  GADGETDOWN|    /*             or a gadget has been pressed on, or */
  GADGETUP,      /*             a gadge has been released. */
  SMART_REFRESH| /* Flags       Intuition should refresh the window. */
  WINDOWCLOSE|   /*             Close Gadget. */
  WINDOWDRAG|    /*             Drag gadget. */
  WINDOWDEPTH|   /*             Depth arrange Gadgets. */
  WINDOWSIZING|  /*             Sizing Gadget. */
  ACTIVATE,      /*             The window should be Active when opened. */
  &my_second_gadget, /* FirstGadget A pointer to my_second_gadget
                     /* structure. */
  NULL,          /* CheckMark   Use Intuition's default CheckMark. */
  "TOUCH ME",    /* Title       Title of the window. */
```

```
  NULL,          /* Screen      Connected to the Workbench Screen. */
  NULL,          /* BitMap      No Custom BitMap. */
  320,           /* MinWidth    We will not allow the window to become */
  50,            /* MinHeight   smaller than 320 x 50, and not bigger */
  640,           /* MaxWidth    than 640 x 200. */
  200,           /* MaxHeight */
  WBENCHSCREEN   /* Type        Connected to the Workbench Screen. */
};



main()
{
  /* Boolean variable used for the while loop: */
  BOOL close_me;


  /* Declare a variable in which we will store the IDCMP flag: */
  ULONG class;

  /* In this example we also need to store the address of the gadget */
  /* which sent us the mesage: */
  APTR address;

  /* we declare a memory pointer (APTR) called address. */


  /* Declare a pointer to an IntuiMessage structure: */
  struct IntuiMessage *my_message;



  /* Before we can use Intuition we need to open the Intuition Library: */
  IntuitionBase = (struct IntuitionBase *)
    OpenLibrary( "intuition.library", 0 );

  if( IntuitionBase == NULL )
    exit(); /* Could NOT open the Intuition Library! */



  /* We will now try to open the window: */
  my_window = (struct Window *) OpenWindow( &my_new_window );

  /* Have we opened the window succesfully? */
  if(my_window == NULL)
  {
    /* Could NOT open the Window! */

    /* Close the Intuition Library since we have opened it: */
    CloseLibrary( IntuitionBase );

    exit();
  }



  /* We have opened the window, and everything seems to be OK. */



  close_me = FALSE;
```

```
/* Stay in the while loop until the user has selected the Close window */
/* gadget: */
while( close_me == FALSE )
{
  /* Wait until we have recieved a message: */
  Wait( 1 << my_window->UserPort->mp_SigBit );

  /* Collect the message: */
  my_message = (struct IntuiMessage *) GetMsg( my_window->UserPort );

  /* Have we collected the message sucessfully? */
  if(my_message)
  {
    /* After we have collected the message we can read it, and save any */
    /* important values which we maybe want to check later: */
    class = my_message->Class;      /* Save the IDCMP flag. */
    address = my_message->IAddress; /* Save the address. */

    /* After we have read it we reply as fast as possible: */
    /* REMEMBER! Do never try to read a message after you have replied! */
    /* Some other process has maybe changed it. */
    ReplyMsg( my_message );

    /* Check which IDCMP flag was sent: */
    switch( class )
    {
      case CLOSEWINDOW:  /* The user selected the Close window gadget! */
             close_me=TRUE;
             break;

      case GADGETDOWN:   /* The user has pressed on one of the Boolean */
                         /* gadgets. We have now to check which: */
             if( address == (APTR) &my_first_gadget)
               printf("Gadget 1 Down\n");
             else
               printf("Gadget 2 Down\n");

             /* We need to do some "casting" here again since APTR is a */
             /* normal memory pointer, while &my_first_gadget is a */
             /* pointer to a Gadget structure. It is actually the same */
             /* thing but we need to explain this for the compiler. */

             break;

      case GADGETUP:     /* The user has released one of the Boolean */
                         /* gadgets. We have now to check which: */
             if( address == (APTR) &my_first_gadget)
               printf("Gadget 1 Up\n");
             else
               printf("Gadget 2 Up\n");
             break;
    }
  }
}



/* We should always close the windows we have opened before we leave: */
CloseWindow( my_window );
```

```
  /* Close the Intuition Library since we have opened it: */
  CloseLibrary( IntuitionBase );

  /* THE END */
}
```

Example5

This program will open a normal window which is connected to the Workbench Screen. The window will use all System Gadgets, and will close first when the user has selected the System gadget Close window. Inside the window we have put a Boolean gadget with two Image structures connected to it. Each time the user clicks on the gadget it will change images, lamp on/lamp off.

```
/* Example5                                                         */
/* This program will open a normal window which is connected to the     */
/* Workbench Screen. The window will use all System Gadgets, and will   */
/* close first when the user has selected the System gadget Close       */
/* window. Inside the window we have put a Boolean gadget with two      */
/* Image structures connected to it. Each time the user clicks on the   */
/* gadget it will change images, lamp on/lamp off.                      */



/* Since the program is using Intuition we need to include this file: */
#include <intuition/intuition.h>



struct IntuitionBase *IntuitionBase;



/* The Image data for the dark lamp: */
/* Remember that Image data must ALWAYS be placed in chip memory: */
USHORT chip lamp_off_data[84]=
{
  0x00FF,0x8000, /* Bitplane ZERO */
  0x0700,0x7000,
  0x18F0,0x0C00,
  0x27E0,0x0200,
  0x4800,0x0100,
  0x9100,0x4080,
  0x90AA,0x8080,
  0x8080,0x8080,
  0x4041,0x0100,
  0x2041,0x0200,
  0x1822,0x0C00,
  0x0622,0x3000,
  0x01FF,0xC000,
  0x0150,0x2000,
  0x0205,0x4000,
  0x0150,0x2000,
  0x0205,0x4000,
  0x0150,0x2000,
  0x0205,0x4000,
  0x01FF,0x8000,
  0x003C,0x0000,

  0x0000,0x0000, /* Bitplane ONE */
  0x00FF,0x8000,
  0x070F,0xF000,
  0x181F,0xFC00,
  0x37FF,0xFE00,
  0x6EFF,0xBF00,
  0x6F55,0x7F00,
  0x7F7F,0x7F00,
  0x3FBE,0xFE00,
  0x1FBE,0xFC00,
  0x07DD,0xF000,
  0x01DD,0xC000,
  0x0000,0x0000,
  0x00AF,0xC000,
  0x01FA,0x8000,
  0x00AF,0xC000,
  0x01FA,0x8000,
```

```
  0x00AF,0xC000,
  0x01FA,0x8000,
  0x0000,0x0000,
  0x0000,0x0000
};

/* The Image structure for the dark lamp: */
struct Image lamp_off=
{
  0, 0,          /* LeftEdge, TopEdge */
  25, 21,        /* Width, Height */
  2,             /* Depth */
  lamp_off_data, /* ImageData */
  0x03, 0x00,    /* PlanePick, PlaneOnOff */
  NULL           /* NextImage */
};



/* The Image data for the light lamp: */
/* Remember that Image data must ALWAYS be placed in chip memory: */
USHORT chip lamp_on_data[84]=
{
  0x00FF,0x8000, /* Bitplane ZERO */
  0x07FF,0xF000,
  0x1FFF,0xFC00,
  0x3FFF,0xFE00,
  0x7FFF,0xFF00,
  0xFFFF,0xFF80,
  0xFFFF,0xFF80,
  0xFFFF,0xFF80,
  0x7FFF,0xFF00,
  0x3FFF,0xFE00,
  0x1FFF,0xFC00,
  0x07FF,0xF000,
  0x01FF,0xC000,
  0x0150,0x2000,
  0x0205,0x4000,
  0x0150,0x2000,
  0x0205,0x4000,
  0x0150,0x2000,
  0x0205,0x4000,
  0x01FF,0x8000,
  0x003C,0x0000,

  0x0000,0x0000, /* Bitplane ONE */
  0x00FF,0x8000,
  0x070F,0xF000,
  0x181F,0xFC00,
  0x37FF,0xFE00,
  0x6CFF,0x9F00,
  0x6E00,0x3F00,
  0x7E7F,0x3F00,
  0x3F3E,0x7E00,
  0x1F3E,0x7C00,
  0x079C,0xF000,
  0x019C,0xC000,
  0x0000,0x0000,
  0x00AF,0xC000,
  0x01FA,0x8000,
  0x00AF,0xC000,
  0x01FA,0x8000,
  0x00AF,0xC000,
  0x01FA,0x8000,
  0x0000,0x0000,
  0x0000,0x0000
};

/* The Image structure for the light lamp: */
struct Image lamp_on=
{
  0, 0,         /* LeftEdge, TopEdge */
  25, 21,       /* Width, Height */
  2,            /* Depth */
  lamp_on_data, /* ImageData */
  0x03, 0x00,   /* PlanePick, PlaneOnOff */
  NULL          /* NextImage */
};



struct Gadget my_gadget=
{
  NULL,          /* NextGadget, no more gadgets in the list. */
  40,            /* LeftEdge, 40 pixels out. */
  20,            /* TopEdge, 20 lines down. */
  25,            /* Width, 25 pixels wide. */
  21,            /* Height, 21 pixels lines heigh. */
  GADGHIMAGE|    /* Flags, display an alternative image when selected. */
  GADGIMAGE,     /* The gadget should be rendered as an Image. */
  GADGIMMEDIATE| /* Activation, our program will recieve a message when */
                 /* the user has selected this gadget. */
  TOGGLESELECT,  /* The on/off state of the gadget is toggled each time. */
  BOOLGADGET,    /* GadgetType, a Boolean gadget. */
  (APTR) &lamp_off, /* GadgetRender, a pointer to our unselected Image. */
                 /* (Since Intuition does not know if this will be a */
                 /* pointer to a Border structure or an Image structure, */
                 /* Intuition expects an APTR (normal memory pointer). */
                 /* We will therefore have to calm down the compiler by */
                 /* doing some "casting". We tell the compiler that */
                 /* the pointer to the Image structure is the same thing */
                 /* as a memory pointer (APTR). */
  (APTR) &lamp_on, /* SelectRender, a pointer to the alternative image. */
  NULL,          /* GadgetText, no text connected to this gadget. */
  NULL,          /* MutualExclude, no mutual exclude. */
  NULL,          /* SpecialInfo, NULL since this is a Boolean gadget. */
                 /* (It is not a Proportional/String or Integer gadget) */
  NULL           /* UserData, no user data connected to the gadget. */
};



/* Declare a pointer to a Window structure: */
struct Window *my_window;

/* Declare and initialize your NewWindow structure: */
struct NewWindow my_new_window=
{
  50,            /* LeftEdge    x position of the window. */
  25,            /* TopEdge     y positio of the window. */
  200,           /* Width       200 pixels wide. */
  100,           /* Height      100 lines high. */
  0,             /* DetailPen   Text should be drawn with colour reg. 0 */
  1,             /* BlockPen    Blocks should be drawn with colour reg. 1 */
```

```
  CLOSEWINDOW|   /* IDCMPFlags  The window will give us a message if the */
                 /*             user has selected the Close window gad, */
  GADGETDOWN,    /*             or a gadget has been pressed on. */
  SMART_REFRESH| /* Flags       Intuition should refresh the window. */
  WINDOWCLOSE|   /*             Close Gadget. */
  WINDOWDRAG|    /*             Drag gadget. */
  WINDOWDEPTH|   /*             Depth arrange Gadgets. */
  WINDOWSIZING|  /*             Sizing Gadget. */
  ACTIVATE,      /*             The window should be Active when opened. */
  &my_gadget,    /* FirstGadget A pointer to my_gadget structure. */
  NULL,          /* CheckMark   Use Intuition's default CheckMark. */
  "ENLIGHTEN ME",/* Title       Title of the window. */
  NULL,          /* Screen      Connected to the Workbench Screen. */
  NULL,          /* BitMap      No Custom BitMap. */
  140,           /* MinWidth    We will not allow the window to become */
  50,            /* MinHeight   smaller than 140 x 50, and not bigger */
  300,           /* MaxWidth    than 300 x 200. */
  200,           /* MaxHeight */
  WBENCHSCREEN   /* Type        Connected to the Workbench Screen. */
};



main()
{
  /* Boolean variable used for the while loop: */
  BOOL close_me;

  /* Declare a variable in which we will store the IDCMP flag: */
  ULONG class;

  /* Declare a pointer to an IntuiMessage structure: */
  struct IntuiMessage *my_message;



  /* Before we can use Intuition we need to open the Intuition Library: */
  IntuitionBase = (struct IntuitionBase *)
    OpenLibrary( "intuition.library", 0 );

  if( IntuitionBase == NULL )
    exit(); /* Could NOT open the Intuition Library! */



  /* We will now try to open the window: */
  my_window = (struct Window *) OpenWindow( &my_new_window );

  /* Have we opened the window succesfully? */
  if(my_window == NULL)
  {
    /* Could NOT open the Window! */

    /* Close the Intuition Library since we have opened it: */
    CloseLibrary( IntuitionBase );

    exit();
  }



  /* We have opened the window, and everything seems to be OK. */
  close_me = FALSE;

  /* Stay in the while loop until the user has selected the Close window */
  /* gadget: */
  while( close_me == FALSE )
  {
    /* Wait until we have recieved a message: */
    Wait( 1 << my_window->UserPort->mp_SigBit );

    /* Collect the message: */
    my_message = (struct IntuiMessage *) GetMsg( my_window->UserPort );

    /* Have we collected the message sucessfully? */
    if(my_message)
    {
      /* After we have collected the message we can read it, and save any */
      /* important values which we maybe want to check later: */
      class = my_message->Class;

      /* After we have read it we reply as fast as possible: */
      /* REMEMBER! Do never try to read a message after you have replied! */
      /* Some other process has maybe changed it. */
      ReplyMsg( my_message );

      /* Check which IDCMP flag was sent: */
      switch( class )
      {
        case CLOSEWINDOW:  /* The user selected the Close window gadget! */
               close_me=TRUE;
               break;

        case GADGETDOWN:   /* The user has pressed on the Boolean gadget. */
               /* Is the lamp on? */
               /* We check if the SELECTED bit is set: */
               if(my_gadget.Flags & SELECTED)
                 printf("Lamp: ON\n");
               else
printf("Lamp: OFF\n");

               break;
      }
    }
  }



  /* We should always close the windows we have opened before we leave: */
  CloseWindow( my_window );



  /* Close the Intuition Library since we have opened it: */
  CloseLibrary( IntuitionBase );

  /* THE END */
}
```

Example6

This program will open a normal window which is connected to the Workbench Screen. The window will use all System Gadgets, and will close first when the user has selected the System gadget Close window. Inside the window we have put a Boolean gadget with a connecting mask. The gadget will only be highlighted when the user selects this gadget while pointing inside the specified (masked) area.

```
/* Example6                                                     */
/* This program will open a normal window which is connected to the   */
/* Workbench Screen. The window will use all System Gadgets, and will */
/* close first when the user has selected the System gadget Close     */
/* window. Inside the window we have put a Boolean gadget with a      */
/* connecting mask. The gadget will only be highlighted when the user */
/* selects this gadget while pointing inside the specified (masked)   */
/* area.                                                              */



/* If your program is using Intuition you should include intuition.h: */
#include <intuition/intuition.h>



struct IntuitionBase *IntuitionBase;



/* Image data for the gadget: */
USHORT chip my_image_data[32]=
{
   0xFFFF,0xFFFF, /* Bitplane ZERO */
   0xFFF8,0x1FFF,
   0xFFE0,0x07FF,
   0xFF80,0x01FF,
   0xFE00,0x007F,
   0xF800,0x001F,
   0xE000,0x0007,
   0x8000,0x0001,
   0x8000,0x0001,
   0xE000,0x0007,
   0xF800,0x001F,
   0xFE00,0x007F,
   0xFF80,0x01FF,
   0xFFE0,0x07FF,
   0xFFF8,0x1FFF,
   0xFFFF,0xFFFF
};

/* Image structure for the gadget: */
struct Image my_image=
{
  0, 0,          /* LeftEdge, TopEdge */
  32, 16,        /* Width, Height */
  1,             /* Depth */
  my_image_data, /* ImageData */
  0x01, 0x00,    /* PlanePick, PlaneOnOff */
  NULL           /* NextImage */
};


     UWORD chip my_mask[32]=
{
   0x0000,0x0000, /* Bitplane ZERO */
   0x0007,0xE000,
   0x001F,0xF800,
   0x007F,0xFE00,
   0x01FF,0xFF80,
   0x07FF,0xFFE0,
   0x1FFF,0xFFF8,
   0x7FFF,0xFFFE,
   0x7FFF,0xFFFE,
   0x1FFF,0xFFF8,
   0x07FF,0xFFE0,
   0x01FF,0xFF80,
   0x007F,0xFE00,
   0x001F,0xF800,
   0x0007,0xE000,
   0x0000,0x0000
};

/* The BoolInfo structure fot the gadget: */
struct BoolInfo my_bool_info=
{
          BOOLMASK,  /* Flags, for the moment this is the only flag you may use. */
          my_mask,   /* Mask, pointer to our bit mask. Only when the user clicks */
                     /* inside the small area of the gadget it will be selected, */
                                                         /* and only that area
will be highlighted. */
                                                         /* Remember! The width
and height of the mask data must */
                                                         /* be the same as the
width and height of the gadget. */
  0           /* Reserved, set this variable to 0 for the moment. */
};

/* The Gadget structure: */
struct Gadget my_gadget=
{
  NULL,          /* NextGadget, no more gadgets in the list. */
  40,            /* LeftEdge, 40 pixels out. */
  20,            /* TopEdge, 20 lines down. */
  32,            /* Width, 32 pixels wide. */
  16,            /* Height, 16 pixels lines heigh. */
  GADGHCOMP|     /* Flags, complement the colours when selected. */
  GADGIMAGE,     /* Render the gadget with an Image structure. */
  GADGIMMEDIATE| /* Activation, our program will recieve a message when */
  RELVERIFY|     /* the user has selected this gadget, and when the user */
                 /* has released it. */
  BOOLEXTEND,    /* This gadget has an BoolInfo connected to it. */
  BOOLGADGET,    /* GadgetType, a Boolean gadget. */
  (APTR) &my_image, /* GadgetRender, a pointer to our Image structure. */
  NULL,          /* SelectRender, NULL since we do not supply the gadget */
                 /* with an alternative image. (We complement the */
                 /* colours instead) */
  NULL,          /* GadgetText, no text connected to the gadget. */
                 /* (See chapter 3 GRAPHICS for more information) */
  NULL,          /* MutualExclude, no mutual exclude. */
  (APTR) &my_bool_info, /* SpecialInfo, pointer to the BoolInfo str. */
  0,             /* GadgetID, no id. */
  NULL           /* UserData, no user data connected to the gadget. */
};



/* Declare a pointer to a Window structure: */
struct Window *my_window;

/* Declare and initialize your NewWindow structure: */
struct NewWindow my_new_window=
{
  50,            /* LeftEdge    x position of the window. */
  25,            /* TopEdge     y positio of the window. */
  200,           /* Width       200 pixels wide. */
  100,           /* Height      100 lines high. */
  0,             /* DetailPen   Text should be drawn with colour reg. 0 */
  1,             /* BlockPen    Blocks should be drawn with colour reg. 1 */
  CLOSEWINDOW|   /* IDCMPFlags  The window will give us a message if the */
                 /*             user has selected the Close window gad, */
  GADGETDOWN|    /*             or a gadget has been pressed on, or */
  GADGETUP,      /*             a gadge has been released. */
  SMART_REFRESH| /* Flags       Intuition should refresh the window. */
  WINDOWCLOSE|   /*             Close Gadget. */
  WINDOWDRAG|    /*             Drag gadget. */
  WINDOWDEPTH|   /*             Depth arrange Gadgets. */
  WINDOWSIZING|  /*             Sizing Gadget. */
  ACTIVATE,      /*             The window should be Active when opened. */
  &my_gadget,    /* FirstGadget A pointer to my_gadget structure. */
  NULL,          /* CheckMark   Use Intuition's default CheckMark. */
  "TOUCH ME",    /* Title       Title of the window. */
  NULL,          /* Screen      Connected to the Workbench Screen. */
  NULL,          /* BitMap      No Custom BitMap. */
  140,           /* MinWidth    We will not allow the window to become */
  50,            /* MinHeight   smaller than 140 x 50, and not bigger */
  300,           /* MaxWidth    than 300 x 200. */
  200,           /* MaxHeight */
  WBENCHSCREEN   /* Type        Connected to the Workbench Screen. */
};



main()
{
  /* Boolean variable used for the while loop: */
  BOOL close_me;

  /* Declare a variable in which we will store the IDCMP flag: */
  ULONG class;

  /* Declare a pointer to an IntuiMessage structure: */
  struct IntuiMessage *my_message;



  /* Before we can use Intuition we need to open the Intuition Library: */
  IntuitionBase = (struct IntuitionBase *)
    OpenLibrary( "intuition.library", 0 );

  if( IntuitionBase == NULL )
    exit(); /* Could NOT open the Intuition Library! */



  /* We will now try to open the window: */
  my_window = (struct Window *) OpenWindow( &my_new_window );

  /* Have we opened the window succesfully? */
  if(my_window == NULL)
  {
    /* Could NOT open the Window! */

    /* Close the Intuition Library since we have opened it: */
    CloseLibrary( IntuitionBase );

    exit();
  }



  /* We have opened the window, and everything seems to be OK. */



  close_me = FALSE;

  /* Stay in the while loop until the user has selected the Close window */
  /* gadget: */
  while( close_me == FALSE )
  {
    /* Wait until we have recieved a message: */
    Wait( 1 << my_window->UserPort->mp_SigBit );

    /* Collect the message: */
    my_message = (struct IntuiMessage *) GetMsg( my_window->UserPort );

    /* Have we collected the message sucessfully? */
    if(my_message)
    {
      /* After we have collected the message we can read it, and save any */
      /* important values which we maybe want to check later: */
      class = my_message->Class;

      /* After we have read it we reply as fast as possible: */
      /* REMEMBER! Do never try to read a message after you have replied! */
      /* Some other process has maybe changed it. */
      ReplyMsg( my_message );

      /* Check which IDCMP flag was sent: */
      switch( class )
      {
        case CLOSEWINDOW:  /* The user selected the Close window gadget! */
               close_me=TRUE;
               break;

        case GADGETDOWN:   /* The user has pressed on the Boolean gadget. */
               printf("Down\n");
               break;

        case GADGETUP:     /* The user has released the Boolean gadget. */
               printf("Up\n");
               break;
      }
    }
  }



  /* We should always close the windows we have opened before we leave: */
  CloseWindow( my_window );



  /* Close the Intuition Library since we have opened it: */
  CloseLibrary( IntuitionBase );

  /* THE END */
}
```

Example7

This program will open a normal window which is connected to the Workbench Screen. The window will use all System Gadgets, and will close first when the user has selected the System gadget Close window. Inside the window we have put a String gadget.

```
/* Example7                                                     */
/* This program will open a normal window which is connected to the   */
/* Workbench Screen. The window will use all System Gadgets, and will */
/* close first when the user has selected the System gadget Close     */
/* window. Inside the window we have put a String gadget.             */


#include <intuition/intuition.h>


struct IntuitionBase *IntuitionBase;


/* THE STRING GADGET's STRUCTURES: */

/* The coordinates for the box: */
SHORT my_points[]=
{
   -7, -4, /* Start at position (-7, -4) */
  200, -4, /* Draw a line to the right to position (200,-4) */
  200, 11, /* Draw a line down to position (200,11) */
   -7, 11, /* Draw a line to the right to position (-7,11) */
   -7, -4  /* Finish of by drawing a line up to position (-7,-4) */
};

/* The Border structure: */
struct Border my_border=
{
  0, 0,        /* LeftEdge, TopEdge. */
  1,           /* FrontPen, colour register 1. */
  0,           /* BackPen, for the moment unused. */
  JAM1,        /* DrawMode, draw the lines with colour 1. */
  5,           /* Count, 5 pair of coordinates in the array. */
  my_points,   /* XY, pointer to the array with the coordinates. */
  NULL,        /* NextBorder, no other Border structures are connected. */
};


/* The IntuiText structure: */
struct IntuiText my_text=
{
  1,         /* FrontPen, colour register 1. */
  0,         /* BackPen, colour register 0. */
  JAM1,      /* DrawMode, draw the characters with colour 1, do not */
             /* change the background. */
  -53, 0,    /* LeftEdge, TopEdge. */
  NULL,      /* ITextFont, use default font. */
  "Text:",   /* IText, the text that will be printed. */
  NULL,      /* NextText, no other IntuiText structures. */
};


UBYTE my_buffer[50]; /* 50 characters including the NULL-sign. */
UBYTE my_undo_buffer[50]; /* Must be at least as big as my_buffer. */



struct StringInfo my_string_info=
{
  my_buffer,       /* Buffer, pointer to a null-terminated string. */
  my_undo_buffer,  /* UndoBuffer, pointer to a null-terminated string. */
                   /* (Remember my_buffer is equal to &my_buffer[0]) */
  0,                /* BufferPos, initial position of the cursor. */
  50,               /* MaxChars, 50 characters + null-sign ('\0'). */
  0,                /* DispPos, first character in the string should be */
                    /* first character in the display. */

  /* Intuition initializes and maintaines these variables: */

  0,                /* UndoPos */
  0,                /* NumChars */
  0,                /* DispCount */
  0, 0,             /* CLeft, CTop */
  NULL,             /* LayerPtr */
  NULL,             /* LongInt */
  NULL,             /* AltKeyMap */
};


struct Gadget my_gadget=
{
  NULL,           /* NextGadget, no more gadgets in the list. */
  68,             /* LeftEdge, 68 pixels out. */
  30,             /* TopEdge, 30 lines down. */
  198,            /* Width, 198 pixels wide. */
  8,              /* Height, 8 pixels lines heigh. */
  GADGHCOMP,      /* Flags, draw the select box in the complement */
                  /* colours. Note: it actually only the cursor which */
                  /* will be drawn in the complement colours (yellow). */
                  /* If you set the flag GADGHNONE the cursor will not be */
                  /* highlighted, and the user will therefore not be able */
                  /* to see it. */
  GADGIMMEDIATE| /* Activation, our program will recieve a message when */
  RELVERIFY,      /* the user has selected this gadget, and when the user */
                  /* has released it. */
  STRGADGET,      /* GadgetType, a String gadget. */
  (APTR) &my_border, /* GadgetRender, a pointer to our Border structure. */
  NULL,           /* SelectRender, NULL since we do not supply the gadget */
                  /* with an alternative image. */
  &my_text,       /* GadgetText, a pointer to our IntuiText structure. */
  NULL,           /* MutualExclude, no mutual exclude. */
  (APTR) &my_string_info, /* SpecialInfo, a pointer to a StringInfo str. */
  0,              /* GadgetID, no id. */
  NULL            /* UserData, no user data connected to the gadget. */
};



/* Declare a pointer to a Window structure: */
struct Window *my_window;

/* Declare and initialize your NewWindow structure: */
struct NewWindow my_new_window=
{
  50,             /* LeftEdge    x position of the window. */
  25,             /* TopEdge     y positio of the window. */
  320,            /* Width       320 pixels wide. */
  100,            /* Height      100 lines high. */
  0,              /* DetailPen   Text should be drawn with colour reg. 0 */
  1,              /* BlockPen    Blocks should be drawn with colour reg. 1 */
  CLOSEWINDOW|    /* IDCMPFlags  The window will give us a message if the */
                  /*             user has selected the Close window gad, */
  GADGETDOWN|     /*             or a gadget has been pressed on, or */
  GADGETUP,       /*             a gadge has been released. */
  SMART_REFRESH| /* Flags       Intuition should refresh the window. */
  WINDOWCLOSE|   /*             Close Gadget. */
  WINDOWDRAG|    /*             Drag gadget. */
  WINDOWDEPTH|   /*             Depth arrange Gadgets. */
  WINDOWSIZING|  /*             Sizing Gadget. */
  ACTIVATE,      /*             The window should be Active when opened. */
  &my_gadget,    /* FirstGadget A pointer to the String gadget. */
  NULL,          /* CheckMark   Use Intuition's default CheckMark. */
  "String Window",/* Title       Title of the window. */
  NULL,          /* Screen      Connected to the Workbench Screen. */
  NULL,          /* BitMap      No Custom BitMap. */
  320,           /* MinWidth    We will not allow the window to become */
  50,            /* MinHeight   smaller than 320 x 50, and not bigger */
  640,           /* MaxWidth    than 640 x 200. */
  200,           /* MaxHeight */
  WBENCHSCREEN   /* Type        Connected to the Workbench Screen. */
};


main()
{
  /* Boolean variable used for the while loop: */
  BOOL close_me;

  /* Declare a variable in which we will store the IDCMP flag: */
  ULONG class;

  /* Declare a pointer to an IntuiMessage structure: */
  struct IntuiMessage *my_message;



  /* Put some text into the my_buffer string: */
  strcpy( my_buffer, "Some text" );



  /* Before we can use Intuition we need to open the Intuition Library: */
  IntuitionBase = (struct IntuitionBase *)
    OpenLibrary( "intuition.library", 0 );

  if( IntuitionBase == NULL )
    exit(); /* Could NOT open the Intuition Library! */


  /* We will now try to open the window: */
  my_window = (struct Window *) OpenWindow( &my_new_window );

  /* Have we opened the window succesfully? */
  if(my_window == NULL)
  {
    /* Could NOT open the Window! */

    /* Close the Intuition Library since we have opened it: */
    CloseLibrary( IntuitionBase );

    exit();
  }



  /* We have opened the window, and everything seems to be OK. */
```

```

  close_me = FALSE;

  /* Stay in the while loop until the user has selected the Close window */
  /* gadget: */
  while( close_me == FALSE )
  {
    /* Wait until we have recieved a message: */
    Wait( 1 << my_window->UserPort->mp_SigBit );

    /* Collect the message: */
    my_message = (struct IntuiMessage *) GetMsg( my_window->UserPort );

    /* Have we collected the message sucessfully? */
    if(my_message)
    {
      /* After we have collected the message we can read it, and save any */
      /* important values which we maybe want to check later: */
      class = my_message->Class;      /* Save the IDCMP flag. */

      /* After we have read it we reply as fast as possible: */
      /* REMEMBER! Do never try to read a message after you have replied! */
      /* Some other process has maybe changed it. */
      ReplyMsg( my_message );

      /* Check which IDCMP flag was sent: */
      switch( class )
      {
        case CLOSEWINDOW:  /* The user selected the Close window gadget! */
               close_me=TRUE;
               break;

        case GADGETDOWN:   /* The user has selected the String gadget: */
                           /* (Clicked inside the select box) */
               printf("String gadget selected.\n");
               break;

        case GADGETUP:     /* The user has released the String gadget: */
                           /* (Pressed ENTER or RETURN) */
               printf("String gadget released.\n");
               break;
      }
    }
  }


  /* Print out the final string: */
  printf("String: %s\n", my_string_info.Buffer);



  /* We should always close the windows we have opened before we leave: */
  CloseWindow( my_window );



  /* Close the Intuition Library since we have opened it: */
  CloseLibrary( IntuitionBase );

  /* THE END */
}
```

Example8

Same as Example7 except that it is an Integer gadget.

```c
/* Example8                                                     */
/* This program will open a normal window which is connected to the   */
/* Workbench Screen. The window will use all System Gadgets, and will */
/* close first when the user has selected the System gadget Close     */
/* window. Inside the window we have put an Integer gadget.           */


#include <intuition/intuition.h>



struct IntuitionBase *IntuitionBase;



/* THE INTEGER GADGET's STRUCTURES: */

/* The coordinates for the box: */
SHORT my_points[]=
{
   -7, -4, /* Start at position (-7, -4) */
  200, -4, /* Draw a line to the right to position (200,-4) */
  200, 11, /* Draw a line down to position (200,11) */
   -7, 11, /* Draw a line to the right to position (-7,11) */
   -7, -4  /* Finish of by drawing a line up to position (-7,-4) */
};

/* The Border structure: */
struct Border my_border=
{
  0, 0,        /* LeftEdge, TopEdge. */
  1,           /* FrontPen, colour register 1. */
  0,           /* BackPen, for the moment unused. */
  JAM1,        /* DrawMode, draw the lines with colour 1. */
  5,           /* Count, 5 pair of coordinates in the array. */
  my_points,   /* XY, pointer to the array with the coordinates. */
  NULL,        /* NextBorder, no other Border structures are connected. */
};



/* The IntuiText structure: */
struct IntuiText my_text=
{
  1,         /* FrontPen, colour register 1. */
  0,         /* BackPen, colour register 0. */
  JAM1,      /* DrawMode, draw the characters with colour 1, do not */
             /* change the background. */
  -37, 0,    /* LeftEdge, TopEdge. */
  NULL,      /* ITextFont, use default font. */
  "Nr:",     /* IText, the text that will be printed. */
  NULL,      /* NextText, no other IntuiText structures. */
};



UBYTE my_buffer[25]; /* 25 characters including the NULL-sign. */
UBYTE my_undo_buffer[25]; /* Must be at least as big as my_buffer. */



struct StringInfo my_string_info=
{
  my_buffer,       /* Buffer, pointer to a null-terminated string. */
  my_undo_buffer,  /* UndoBuffer, pointer to a null-terminated string. */
                   /* (Remember my_buffer is equal to &my_buffer[0]) */
  0,               /* BufferPos, initial position of the cursor. */
  25,              /* MaxChars, 25 characters + null-sign ('\0'). */
  0,               /* DispPos, first character in the string should be */
                   /* first character in the display. */

  /* Intuition initializes and maintaines these variables: */

  0,               /* UndoPos */
  0,               /* NumChars */
  0,               /* DispCount */
  0, 0,            /* CLeft, CTop */
  NULL,            /* LayerPtr */
  NULL,            /* LongInt */
  NULL,            /* AltKeyMap */
};


struct Gadget my_gadget=
{
  NULL,          /* NextGadget, no more gadgets in the list. */
  68,            /* LeftEdge, 68 pixels out. */
  30,            /* TopEdge, 30 lines down. */
  198,           /* Width, 198 pixels wide. */
  8,             /* Height, 8 pixels lines heigh. */
  GADGHCOMP,     /* Flags, draw the select box in the complement */
                 /* colours. Note: it actually only the cursor which */
                 /* will be drawn in the complement colours (yellow). */
                 /* If you set the flag GADGHNONE the cursor will not be */
                 /* highlighted, and the user will therefore not be able */
                 /* to see it. */
  GADGIMMEDIATE| /* Activation, our program will recieve a message when */
  RELVERIFY|     /* the user has selected this gadget, and when the user */
                 /* has released it. */
  LONGINT,       /* An Integer gadget. */
  STRGADGET,     /* GadgetType, a String gadget. */
  (APTR) &my_border, /* GadgetRender, a pointer to our Border structure. */
  NULL,          /* SelectRender, NULL since we do not supply the gadget */
                 /* with an alternative image. */
  &my_text,      /* GadgetText, a pointer to our IntuiText structure. */
  NULL,          /* MutualExclude, no mutual exclude. */
  (APTR) &my_string_info, /* SpecialInfo, a pointer to a StringInfo str. */
  0,             /* GadgetID, no id. */
  NULL           /* UserData, no user data connected to the gadget. */
};



/* Declare a pointer to a Window structure: */
struct Window *my_window;

/* Declare and initialize your NewWindow structure: */
struct NewWindow my_new_window=
{
  50,            /* LeftEdge    x position of the window. */
  25,            /* TopEdge     y positio of the window. */
  320,           /* Width       320 pixels wide. */
  100,           /* Height      100 lines high. */
  0,             /* DetailPen   Text should be drawn with colour reg. 0 */
  1,             /* BlockPen    Blocks should be drawn with colour reg. 1 */
  CLOSEWINDOW|   /* IDCMPFlags  The window will give us a message if the */
                 /*             user has selected the Close window gad, */
  GADGETDOWN|    /*             or a gadget has been pressed on, or */
  GADGETUP,      /*             a gadge has been released. */
  SMART_REFRESH| /* Flags       Intuition should refresh the window. */
  WINDOWCLOSE|   /*             Close Gadget. */
  WINDOWDRAG|    /*             Drag gadget. */
  WINDOWDEPTH|   /*             Depth arrange Gadgets. */
  WINDOWSIZING|  /*             Sizing Gadget. */
  ACTIVATE,      /*             The window should be Active when opened. */
  &my_gadget,    /* FirstGadget A pointer to the String gadget. */
  NULL,          /* CheckMark   Use Intuition's default CheckMark. */
  "Integer Window",/* Title       Title of the window. */
  NULL,          /* Screen      Connected to the Workbench Screen. */
  NULL,          /* BitMap      No Custom BitMap. */
  320,           /* MinWidth    We will not allow the window to become */
  50,            /* MinHeight   smaller than 320 x 50, and not bigger */
  640,           /* MaxWidth    than 640 x 200. */
  200,           /* MaxHeight */
  WBENCHSCREEN   /* Type        Connected to the Workbench Screen. */
};


main()
{
  /* Boolean variable used for the while loop: */
  BOOL close_me;

  /* Declare a variable in which we will store the IDCMP flag: */
  ULONG class;

  /* Declare a pointer to an IntuiMessage structure: */
  struct IntuiMessage *my_message;


  /* Put an integer value in the string: */
  /* This is very important! */
  strcpy( my_buffer, "0" );


  /* Before we can use Intuition we need to open the Intuition Library: */
  IntuitionBase = (struct IntuitionBase *)
    OpenLibrary( "intuition.library", 0 );

  if( IntuitionBase == NULL )
    exit(); /* Could NOT open the Intuition Library! */


  /* We will now try to open the window: */
  my_window = (struct Window *) OpenWindow( &my_new_window );

  /* Have we opened the window succesfully? */
  if(my_window == NULL)
  {
    /* Could NOT open the Window! */

    /* Close the Intuition Library since we have opened it: */
    CloseLibrary( IntuitionBase );

    exit();
  }


  /* We have opened the window, and everything seems to be OK. */


  close_me = FALSE;

  /* Stay in the while loop until the user has selected the Close window */
  /* gadget: */
  while( close_me == FALSE )
  {
    /* Wait until we have recieved a message: */
    Wait( 1 << my_window->UserPort->mp_SigBit );

    /* Collect the message: */
    my_message = (struct IntuiMessage *) GetMsg( my_window->UserPort );

    /* Have we collected the message sucessfully? */
    if(my_message)
    {
      /* After we have collected the message we can read it, and save any */
      /* important values which we maybe want to check later: */
      class = my_message->Class;      /* Save the IDCMP flag. */

      /* After we have read it we reply as fast as possible: */
      /* REMEMBER! Do never try to read a message after you have replied! */
      /* Some other process has maybe changed it. */
      ReplyMsg( my_message );

      /* Check which IDCMP flag was sent: */
      switch( class )
      {
        case CLOSEWINDOW:  /* The user selected the Close window gadget! */
               close_me=TRUE;
               break;

        case GADGETDOWN:   /* The user has selected the Integer gadget: */
                           /* (Clicked inside the select box) */
               printf("Integer gadget selected.\n");
               break;

        case GADGETUP:     /* The user has released the Integer gadget: */
                           /* (Pressed ENTER or RETURN) */
               printf("Integer gadget released.\n");
               /* Print out the integer value: */
               printf("Nr: %d\n\n", my_string_info.LongInt);
               break;
      }
    }
  }


  /* We should always close the windows we have opened before we leave: */
  CloseWindow( my_window );



  /* Close the Intuition Library since we have opened it: */
  CloseLibrary( IntuitionBase );

  /* THE END */
}
```

Example9

Same as Example7 except that it is a Proportional gadget.

```
/* Example9                                                     */
/* This program will open a normal window which is connected to the   */
/* Workbench Screen. The window will use all System Gadgets, and will */
/* close first when the user has selected the System gadget Close     */
/* window. Inside the window we have put a Proportional gadget.       */



#include <intuition/intuition.h>



struct IntuitionBase *IntuitionBase;



/* THE PROPORTIONAL GADGET's STRUCTURES: */

/* The IntuiText structure: */
struct IntuiText my_text=
{
  1,         /* FrontPen, colour register 1. */
  0,         /* BackPen, colour register 0. */
  JAM1,      /* DrawMode, draw the characters with colour 1, do not */
             /* change the background. */
  -65, 2,    /* LeftEdge, TopEdge. */
  NULL,      /* ITextFont, use default font. */
  "Volume:", /* IText, the text that will be printed. */
  NULL,      /* NextText, no other IntuiText structures. */
};


/* We need to declare an Image structure for the knob, but since */
/* Intuition will take care of the size etc of the knob, we do not need */
/* to initialize the Image structure: */
struct Image my_image;


struct PropInfo my_prop_info=
{
  FREEHORIZ|      /* Flags, the knob should be moved horizontally, and */
  AUTOKNOB,       /* Intuition should take care of the knob image. */
  0,              /* HorizPot, start position of the knob. */
  0,              /* VertPot, 0 since we will not move the knob hor. */
  MAXBODY * 1/64, /* HorizBody, 64 steps. */
  0,              /* VertBody, 0 since we will not move the knob hor. */

  /* These variables are initialized and maintained by Intuition: */

  0,              /* CWidth */
  0,              /* CHeight */
  0, 0,           /* HPotRes, VPotRes */
  0,              /* LeftBorder */
  0               /* TopBorder */
};


struct Gadget my_gadget=
{
  NULL,            /* NextGadget, no more gadgets in the list. */
  80,              /* LeftEdge, 80 pixels out. */
  30,              /* TopEdge, 30 lines down. */
  200,             /* Width, 200 pixels wide. */
  12,              /* Height, 12 pixels lines heigh. */
  GADGHCOMP,       /* Flags, complement the colours. */
  GADGIMMEDIATE|   /* Activation, our program will recieve a message */
  RELVERIFY,       /* when the user has selected this gadget, and when */
                   /* the user has released it. */
  PROPGADGET,      /* GadgetType, a Proportional gadget. */
  (APTR) &my_image,/* GadgetRender, a pointer to our Image structure. */
                   /* (Intuition will take care of the knob image) */
                   /* (See chapter 3 GRAPHICS for more information) */
  NULL,            /* SelectRender, NULL since we do not supply the */
                   /* gadget with an alternative image. */
  &my_text,        /* GadgetText, volume. */
  NULL,            /* MutualExclude, no mutual exclude. */
  (APTR) &my_prop_info, /* SpecialInfo, pointer to a PropInfo structure. */
  0,               /* GadgetID, no id. */
  NULL             /* UserData, no user data connected to the gadget. */
};



/* Declare a pointer to a Window structure: */
struct Window *my_window;

/* Declare and initialize your NewWindow structure: */
struct NewWindow my_new_window=
{
  50,            /* LeftEdge    x position of the window. */
  25,            /* TopEdge     y positio of the window. */
  320,           /* Width       320 pixels wide. */
  100,           /* Height      100 lines high. */
  0,             /* DetailPen   Text should be drawn with colour reg. 0 */
  1,             /* BlockPen    Blocks should be drawn with colour reg. 1 */
  CLOSEWINDOW|   /* IDCMPFlags  The window will give us a message if the */
                 /*             user has selected the Close window gad, */
  GADGETDOWN|    /*             or a gadget has been pressed on, or */
  GADGETUP,      /*             a gadge has been released. */
  SMART_REFRESH| /* Flags       Intuition should refresh the window. */
  WINDOWCLOSE|   /*             Close Gadget. */
  WINDOWDRAG|    /*             Drag gadget. */
  WINDOWDEPTH|   /*             Depth arrange Gadgets. */
  WINDOWSIZING|  /*             Sizing Gadget. */
  ACTIVATE,      /*             The window should be Active when opened. */
  &my_gadget,    /* FirstGadget A pointer to the String gadget. */
  NULL,          /* CheckMark   Use Intuition's default CheckMark. */
  "Proportional Window", /* Title Title of the window. */
  NULL,          /* Screen      Connected to the Workbench Screen. */
  NULL,          /* BitMap      No Custom BitMap. */
  320,           /* MinWidth    We will not allow the window to become */
  50,            /* MinHeight   smaller than 320 x 50, and not bigger */
  640,           /* MaxWidth    than 640 x 200. */
  200,           /* MaxHeight */
  WBENCHSCREEN   /* Type        Connected to the Workbench Screen. */
};



main()
{
  /* Boolean variable used for the while loop: */
  BOOL close_me;
```

```
/* Declare a variable in which we will store the IDCMP flag: */
ULONG class;

/* Declare a pointer to an IntuiMessage structure: */
struct IntuiMessage *my_message;



/* Before we can use Intuition we need to open the Intuition Library: */
IntuitionBase = (struct IntuitionBase *)
  OpenLibrary( "intuition.library", 0 );

if( IntuitionBase == NULL )
  exit(); /* Could NOT open the Intuition Library! */



/* We will now try to open the window: */
my_window = (struct Window *) OpenWindow( &my_new_window );

/* Have we opened the window succesfully? */
if(my_window == NULL)
{
  /* Could NOT open the Window! */

  /* Close the Intuition Library since we have opened it: */
  CloseLibrary( IntuitionBase );

  exit();
}



/* We have opened the window, and everything seems to be OK. */



close_me = FALSE;

/* Stay in the while loop until the user has selected the Close window */
/* gadget: */
while( close_me == FALSE )
{
  /* Wait until we have recieved a message: */
  Wait( 1 << my_window->UserPort->mp_SigBit );

  /* We have now recieved one or more messages. */

  /* Since we may recieve several messages we stay in the while loop */
  /* and collect, save, reply and execute the messages until there is */
  /* a pause: */
  while(my_message=(struct IntuiMessage *)GetMsg( my_window->UserPort))
  {
    /* GetMsg will return a pointer to a message if there was one, */
    /* else it returns NULL. We will therefore stay in this while loop */
    /* as long as there are some messages waiting in the port. */

    /* After we have collected the message we can read it, and save */
    /* any important values which we maybe want to check later: */
    class = my_message->Class;      /* Save the IDCMP flag. */

    /* After we have read it we reply as fast as possible: */
```

```
      /* REMEMBER! Do never try to read a message after you have replied! */
      /* Some other process has maybe changed it. */
      ReplyMsg( my_message );

      /* Check which IDCMP flag was sent: */
      switch( class )
      {
        case CLOSEWINDOW:  /* The user selected the Close window gadget! */
               close_me=TRUE;
               break;

        case GADGETDOWN:   /* The user has selected the Prop. gadget: */
               printf("Proportional gadget selected.\n");
               break;

        case GADGETUP:     /* The user has released the Prop. gadget: */
               printf("Proportional gadget released.\n");
               break;
      }
    }
    printf("Volume= %1.0f\n\n", (float) my_prop_info.HorizPot/MAXPOT*64);
  }

  /* We should always close the windows we have opened before we leave: */
  CloseWindow( my_window );



  /* Close the Intuition Library since we have opened it: */
  CloseLibrary( IntuitionBase );

  /* THE END */
}

/************************************************************************/
/* EXTRA INFORMATION:                                                   */
/* We will recieve a message (GADGETDOWN) when the user selects the     */
/* knob, and one message (GADGETUP) when the user releases the knob. If  */
/* the user on the other hand clicks inside the container (not on the    */
/* knob) we will recieve both a GADGETDOWN and a GADGETUP message at the */
/* same time.                                                            */
/* It is because of that we need to have a while loop which collects the */
/* messages once one or more has arrived. We can not as before just wait */
/* and then collect one message, since there may be more in the queue.   */
/************************************************************************/
```

Example10

Same as Example9 except that the Proportional gadget uses a custom image knob.

```
/* Example10                                                      */
/* This program will open a normal window which is connected to the   */
/* Workbench Screen. The window will use all System Gadgets, and will */
/* close first when the user has selected the System gadget Close     */
/* window. Inside the window we have put a Proportional gadget which  */
/* uses a custom image knob.                                          */



#include <intuition/intuition.h>



struct IntuitionBase *IntuitionBase;



/* THE PROPORTIONAL GADGET's STRUCTURES: */

/* The IntuiText structure: */
struct IntuiText my_text=
{
  1,         /* FrontPen, colour register 1. */
  0,         /* BackPen, colour register 0. */
  JAM1,      /* DrawMode, draw the characters with colour 1, do not */
             /* change the background. */
  -65, 8,    /* LeftEdge, TopEdge. */
  NULL,      /* ITextFont, use default font. */
  "Volume:", /* IText, the text that will be printed. */
  NULL,      /* NextText, no other IntuiText structures. */
};


/* Image data for the knob: */
/* Remember that Image data must ALWAYS be in chip memory! */
USHORT chip my_knob_data[80]=
{
  0x01E0,0x0000, /* Bitplane ZERO */
  0x03F0,0x0000,
  0x03F0,0x0000,
  0x03F0,0x0000,
  0x03F0,0x0000,
  0x03F0,0x0000,
  0x03F0,0x0000,
  0x07F8,0x0000,
  0x7BF7,0x8000,
  0x83F0,0x4000,
  0x83F0,0x4000,
  0x7BF7,0x8000,
  0x07F8,0x0000,
  0x03F0,0x0000,
  0x03F0,0x0000,
  0x03F0,0x0000,
  0x03F0,0x0000,
  0x03F0,0x0000,
  0x03F0,0x0000,
  0x01E0,0x0000,

  0x0000,0x0000, /* Bitplane ONE */
  0x01E0,0x0000,
  0x01E0,0x0000,
  0x01E0,0x0000,
  0x01E0,0x0000,
  0x01E0,0x0000,
  0x01E0,0x0000,
  0x01E0,0x0000,
  0x05E8,0x0000,
  0x7DEF,0x8000,
  0x7DEF,0x8000,
  0x05E8,0x0000,
  0x01E0,0x0000,
  0x01E0,0x0000,
  0x01E0,0x0000,
  0x01E0,0x0000,
  0x01E0,0x0000,
  0x01E0,0x0000,
  0x01E0,0x0000,
  0x0000,0x0000
};

/* The Image structure for the knob: */
struct Image my_knob=
{
  0, 0,         /* LeftEdge, TopEdge */
  18, 20,       /* Width, Height */
  2,            /* Depth */
        my_knob_data, /* ImageData */
  0x03, 0x00,   /* PlanePick, PlaneOnOff */
  NULL          /* NextImage */
};


struct PropInfo my_prop_info=
{
  FREEHORIZ,      /* Flags, the knob should be moved horizontally. */
  0,              /* HorizPot, start position of the knob. */
  0,              /* VertPot, 0 since we will not move the knob hor. */
  MAXBODY * 1/64, /* HorizBody, 64 steps. */
  0,              /* VertBody, 0 since we will not move the knob hor. */

  /* These variables are initialized and maintained by Intuition: */

  0,              /* CWidth */
  0,              /* CHeight */
  0, 0,           /* HPotRes, VPotRes */
  0,              /* LeftBorder */
  0               /* TopBorder */
};


struct Gadget my_gadget=
{
  NULL,            /* NextGadget, no more gadgets in the list. */
  80,              /* LeftEdge, 80 pixels out. */
  30,              /* TopEdge, 30 lines down. */
  200,             /* Width, 200 pixels wide. */
  24,              /* Height, 24 pixels lines heigh. */
  GADGHNONE,       /* Flags, no highlightning. */
  GADGIMMEDIATE|   /* Activation, our program will recieve a message */
  RELVERIFY,       /* when the user has selected this gadget, and when */
                   /* the user has released it. */
  PROPGADGET,      /* GadgetType, a Proportional gadget. */
  (APTR) &my_knob, /* GadgetRender, a pointer to our knob Image str. */
  NULL,            /* SelectRender, NULL since we do not supply the */
```

```
                    /* gadget with an alternative image. */
  &my_text,         /* GadgetText, volume. */
  NULL,             /* MutualExclude, no mutual exclude. */
  (APTR) &my_prop_info, /* SpecialInfo, pointer to a PropInfo structure. */
  0,                /* GadgetID, no id. */
  NULL              /* UserData, no user data connected to the gadget. */
};



/* Declare a pointer to a Window structure: */
struct Window *my_window;

/* Declare and initialize your NewWindow structure: */
struct NewWindow my_new_window=
{
  50,             /* LeftEdge    x position of the window. */
  25,             /* TopEdge     y positio of the window. */
  320,            /* Width       320 pixels wide. */
  100,            /* Height      100 lines high. */
  0,              /* DetailPen   Text should be drawn with colour reg. 0 */
  1,              /* BlockPen    Blocks should be drawn with colour reg. 1 */
  CLOSEWINDOW|    /* IDCMPFlags  The window will give us a message if the */
                  /*             user has selected the Close window gad, */
  GADGETDOWN|     /*             or a gadget has been pressed on, or */
  GADGETUP,       /*             a gadge has been released. */
  SMART_REFRESH|  /* Flags       Intuition should refresh the window. */
  WINDOWCLOSE|    /*             Close Gadget. */
  WINDOWDRAG|     /*             Drag gadget. */
  WINDOWDEPTH|    /*             Depth arrange Gadgets. */
  WINDOWSIZING|   /*             Sizing Gadget. */
  ACTIVATE,       /*             The window should be Active when opened. */
  &my_gadget,     /* FirstGadget A pointer to the String gadget. */
  NULL,           /* CheckMark   Use Intuition's default CheckMark. */
  "Proportional Window", /* Title Title of the window. */
  NULL,           /* Screen      Connected to the Workbench Screen. */
  NULL,           /* BitMap      No Custom BitMap. */
  320,            /* MinWidth    We will not allow the window to become */
  60,             /* MinHeight   smaller than 320 x 60, and not bigger */
  640,            /* MaxWidth    than 640 x 200. */
  200,            /* MaxHeight */
  WBENCHSCREEN    /* Type        Connected to the Workbench Screen. */
};



main()
{
  /* Boolean variable used for the while loop: */
  BOOL close_me;

  /* Declare a variable in which we will store the IDCMP flag: */
  ULONG class;

  /* Declare a pointer to an IntuiMessage structure: */
  struct IntuiMessage *my_message;



  /* Before we can use Intuition we need to open the Intuition Library: */
  IntuitionBase = (struct IntuitionBase *)
    OpenLibrary( "intuition.library", 0 );
```

```
if( IntuitionBase == NULL )
  exit(); /* Could NOT open the Intuition Library! */



/* We will now try to open the window: */
my_window = (struct Window *) OpenWindow( &my_new_window );

/* Have we opened the window succesfully? */
if(my_window == NULL)
{
  /* Could NOT open the Window! */

  /* Close the Intuition Library since we have opened it: */
  CloseLibrary( IntuitionBase );

  exit();
}



/* We have opened the window, and everything seems to be OK. */



close_me = FALSE;

/* Stay in the while loop until the user has selected the Close window */
/* gadget: */
while( close_me == FALSE )
{
  /* Wait until we have recieved a message: */
  Wait( 1 << my_window->UserPort->mp_SigBit );

  /* We have now recieved one or more messages. */

  /* Since we may recieve several messages we stay in the while loop */
  /* and collect, save, reply and execute the messages until there is */
  /* a pause: */
  while(my_message=(struct IntuiMessage *)GetMsg( my_window->UserPort))
  {
    /* GetMsg will return a pointer to a message if there was one, */
    /* else it returns NULL. We will therefore stay in this while loop */
    /* as long as there are some messages waiting in the port. */

    /* After we have collected the message we can read it, and save */
    /* any important values which we maybe want to check later: */
    class = my_message->Class;      /* Save the IDCMP flag. */

    /* After we have read it we reply as fast as possible: */
    /* REMEMBER! Do never try to read a message after you have replied! */
    /* Some other process has maybe changed it. */
    ReplyMsg( my_message );

    /* Check which IDCMP flag was sent: */
    switch( class )
    {
      case CLOSEWINDOW:  /* The user selected the Close window gadget! */
             close_me=TRUE;
             break;
```

```
        case GADGETDOWN:   /* The user has selected the Prop. gadget: */
               printf("Proportional gadget selected.\n");
               break;

        case GADGETUP:     /* The user has released the Prop. gadget: */
               printf("Proportional gadget released.\n");
               break;
      }
    }
    printf("Volume= %1.0f\n\n", (float) my_prop_info.HorizPot/MAXPOT*64);
  }

  /* We should always close the windows we have opened before we leave: */
  CloseWindow( my_window );



  /* Close the Intuition Library since we have opened it: */
  CloseLibrary( IntuitionBase );

  /* THE END */
}

/***************************************************************************/
/* EXTRA INFORMATION:                                                      */
/* We will recieve a message (GADGETDOWN) when the user selects the       */
/* knob, and one message (GADGETUP) when the user releases the knob. If   */
/* the user on the other hand clicks inside the container (not on the     */
/* knob) we will recieve both a GADGETDOWN and a GADGETUP message at the */
/* same time.                                                              */
/* It is because of that we need to have a while loop which collects the */
/* messages once one or more has arrived. We can not as before just wait */
/* and then collect one message, since there may be more in the queue.   */
/***************************************************************************/
```

Example11

This program will open a normal window which is connected to the Workbench Screen. The window will use all System Gadgets, and will close first when the user has selected the System gadget Close window. Inside the window we have put a Proportional gadget where the knob can be moved both horizontally and vertically.

```
/* Example11                                                      */
/* This program will open a normal window which is connected to the   */
/* Workbench Screen. The window will use all System Gadgets, and will */
/* close first when the user has selected the System gadget Close     */
/* window. Inside the window we have put a Proportional gadget where  */
/* the knob can be moved both horizontally and vertically.            */



#include <intuition/intuition.h>



struct IntuitionBase *IntuitionBase;



/* THE PROPORTIONAL GADGET's STRUCTURES: */

/* We need to declare an Image structure for the knob, but since */
/* Intuition will take care of the size etc of the knob, we do not need */
/* to initialize the Image structure: */
struct Image my_image;


struct PropInfo my_prop_info=
{
  FREEHORIZ|      /* Flags, the knob should be able to movew both */
  FREEVERT|       /* horizontally and vertically. */
  AUTOKNOB,       /* Intuition should take care of the knob image. */
  0,              /* HorizPot, start position of the knob. */
  0,              /* VertPot, start position of the knob. */
  MAXBODY * 1/32, /* HorizBody, 32 steps. */
  MAXBODY * 1/10, /* VertBody, 10 steps. */

  /* These variables are initialized and maintained by Intuition: */

  0,              /* CWidth */
  0,              /* CHeight */
  0, 0,           /* HPotRes, VPotRes */
  0,              /* LeftBorder */
  0               /* TopBorder */
};


struct Gadget my_gadget=
{
  NULL,            /* NextGadget, no more gadgets in the list. */
  10,              /* LeftEdge, 10 pixels out. */
  20,              /* TopEdge, 20 lines down. */
  -20,             /* Width, always 20 pixels less than the wind. size. */
  -40,             /* Height, always 40 lines less than the wind. size. */
  GADGHCOMP|       /* Flags, complement the colours. */
  GRELWIDTH|       /* Width describes the size relative to the window. */
  GRELHEIGHT,      /* Height describes the size relative to the window*/
  GADGIMMEDIATE|   /* Activation, our program will recieve a message */
  RELVERIFY,       /* when the user has selected this gadget, and when */
                   /* the user has released it. */
  PROPGADGET,      /* GadgetType, a Proportional gadget. */
  (APTR) &my_image,/* GadgetRender, a pointer to our Image structure. */
                   /* (Intuition will take care of the knob image) */
                   /* (See chapter 3 GRAPHICS for more information) */
  NULL,            /* SelectRender, NULL since we do not supply the */
                   /* gadget with an alternative image. */
  NULL,            /* GadgetText, no text. */
  NULL,            /* MutualExclude, no mutual exclude. */
  (APTR) &my_prop_info, /* SpecialInfo, pointer to a PropInfo structure. */
  0,               /* GadgetID, no id. */
  NULL             /* UserData, no user data connected to the gadget. */
};



/* Declare a pointer to a Window structure: */
struct Window *my_window;

/* Declare and initialize your NewWindow structure: */
struct NewWindow my_new_window=
{
  50,            /* LeftEdge    x position of the window. */
  25,            /* TopEdge     y positio of the window. */
  320,           /* Width       320 pixels wide. */
  100,           /* Height      100 lines high. */
  0,             /* DetailPen   Text should be drawn with colour reg. 0 */
  1,             /* BlockPen    Blocks should be drawn with colour reg. 1 */
  CLOSEWINDOW|   /* IDCMPFlags  The window will give us a message if the */
                 /*             user has selected the Close window gad, */
  GADGETDOWN|    /*             or a gadget has been pressed on, or */
  GADGETUP,      /*             a gadge has been released. */
  SMART_REFRESH| /* Flags       Intuition should refresh the window. */
  WINDOWCLOSE|   /*             Close Gadget. */
  WINDOWDRAG|    /*             Drag gadget. */
  WINDOWDEPTH|   /*             Depth arrange Gadgets. */
  WINDOWSIZING|  /*             Sizing Gadget. */
  ACTIVATE,      /*             The window should be Active when opened. */
  &my_gadget,    /* FirstGadget A pointer to the String gadget. */
  NULL,          /* CheckMark   Use Intuition's default CheckMark. */
  "Proportional Window", /* Title Title of the window. */
  NULL,          /* Screen      Connected to the Workbench Screen. */
  NULL,          /* BitMap      No Custom BitMap. */
  100,           /* MinWidth    We will not allow the window to become */
  100,           /* MinHeight   smaller than 100 x 100, and not bigger */
  640,           /* MaxWidth    than 640 x 200. */
  200,           /* MaxHeight */
  WBENCHSCREEN   /* Type        Connected to the Workbench Screen. */
};



main()
{
  /* Boolean variable used for the while loop: */
  BOOL close_me;

  /* Declare a variable in which we will store the IDCMP flag: */
  ULONG class;

  /* Declare a pointer to an IntuiMessage structure: */
  struct IntuiMessage *my_message;



  /* Before we can use Intuition we need to open the Intuition Library: */
  IntuitionBase = (struct IntuitionBase *)
```

```
    OpenLibrary( "intuition.library", 0 );

  if( IntuitionBase == NULL )
    exit(); /* Could NOT open the Intuition Library! */



  /* We will now try to open the window: */
  my_window = (struct Window *) OpenWindow( &my_new_window );

  /* Have we opened the window succesfully? */
  if(my_window == NULL)
  {
    /* Could NOT open the Window! */

    /* Close the Intuition Library since we have opened it: */
    CloseLibrary( IntuitionBase );

    exit();
  }



  /* We have opened the window, and everything seems to be OK. */



  close_me = FALSE;

  /* Stay in the while loop until the user has selected the Close window */
  /* gadget: */
  while( close_me == FALSE )
  {
    /* Wait until we have recieved a message: */
    Wait( 1 << my_window->UserPort->mp_SigBit );

    /* We have now recieved one or more messages. */

    /* Since we may recieve several messages we stay in the while loop */
    /* and collect, save, reply and execute the messages until there is */
    /* a pause: */
    while(my_message=(struct IntuiMessage *)GetMsg( my_window->UserPort))
    {
      /* GetMsg will return a pointer to a message if there was one, */
      /* else it returns NULL. We will therefore stay in this while loop */
      /* as long as there are some messages waiting in the port. */

      /* After we have collected the message we can read it, and save */
      /* any important values which we maybe want to check later: */
      class = my_message->Class;      /* Save the IDCMP flag. */

      /* After we have read it we reply as fast as possible: */
      /* REMEMBER! Do never try to read a message after you have replied! */
      /* Some other process has maybe changed it. */
      ReplyMsg( my_message );

      /* Check which IDCMP flag was sent: */
      switch( class )
      {
        case CLOSEWINDOW:  /* The user selected the Close window gadget! */
               close_me=TRUE;
               break;

        case GADGETDOWN:   /* The user has selected the Prop. gadget: */
               printf("Proportional gadget selected.\n");
               break;

        case GADGETUP:     /* The user has released the Prop. gadget: */
               printf("Proportional gadget released.\n");
               break;
      }
    }
    printf("Hor= %1.0f\n", (float) my_prop_info.HorizPot / MAXPOT * 32);
    printf("Ver= %1.0f\n\n", (float) my_prop_info.VertPot / MAXPOT * 10);
  }

  /* We should always close the windows we have opened before we leave: */
  CloseWindow( my_window );



  /* Close the Intuition Library since we have opened it: */
  CloseLibrary( IntuitionBase );

  /* THE END */
}
```

Example12

This program will open a SuperBitmap window which is connected to the Workbench Screen. The window will use all System Gadgets, and will close first when the user has selected the System gadget Close window. Inside the window we have put two Proportional gadgets, one on the right side, and one at the bottom. With help of these two gadgets, the user can move around the BitMap.

This example is for experienced programmers only, since it uses some functions etc which we have not discussed yet. I have, however, included it here since it is a good example on how you can combine Proportional gadgets with SuperBitmap windows.

```
/* Example12                                                          */
/* This program will open a SuperBitmap window which is connected to the */
/* Workbench Screen. The window will use all System Gadgets, and will    */
/* close first when the user has selected the System gadget Close        */
/* window. Inside the window we have put two Proportional gadgets, one   */
/* on the right side, and one at the bottom. With help of these two      */
/* gadgets, the user can move around the BitMap.                         */
/*                                                                       */
/* This example is for experienced programmers only, and uses some       */
/* functions etc which we have not discussed yet. I have, however,       */
/* included it here since it is a good example on how you can combine    */
/* Proportional gadgets with SuperBitmap windows.                        */



#include <intuition/intuition.h>



#define WIDTH      320
#define MAX_WIDTH  640
#define HEIGHT     128
#define MAX_HEIGHT 256
#define DEPTH        2 /* 4 colours. */



/* Tell the C compiler that the function draw_some_boxes will return: */
void draw_some_boxes(); /* Return nothing (void). */



/* Declare three pointers to the three libraries we are going to open: */
struct IntuitionBase *IntuitionBase;
struct GfxBase *GfxBase;
struct LayersBase *LayersBase;



/***********************************************/
/* THE RIGHT PROPORTIONAL GADGET's STRUCTURES: */
/***********************************************/

/* We need to declare an Image structure for the knob, but since */
/* Intuition will take care of the size etc of the knob, we do not need */
/* to initialize the Image structure: */
struct Image my_right_image;

struct PropInfo my_right_prop_info=
{
  FREEVERT|       /* Flags, the knob should be moved vertically, and */
  AUTOKNOB,       /* Intuition should take care of the knob image. */
  0,              /* HorizPot, 0 since we will not move the knob hor. */
  0,              /* VertPot, start position of the knob. */
  0,              /* HorizBody. 0 since we will not move the knob hor. */
  MAXBODY * HEIGHT / MAX_HEIGHT, /* VertBody. */

  /* These variables are initialized and maintained by Intuition: */

  0,              /* CWidth */
  0,              /* CHeight */
  0, 0,           /* HPotRes, VPotRes */
  0,              /* LeftBorder */
  0               /* TopBorder */
};

struct Gadget my_right_gadget=
{
  NULL,            /* NextGadget, no more gadgets in the list. */
  -15,             /* LeftEdge, 15 pixels out from the right side. */
    9,             /* TopEdge, 9 lines down. */
   16,             /* Width, 16 pixels wide. */
  -17,             /* Height, 17 lines less than the heigh of the wind. */
  GADGHCOMP|       /* Flags, complement the colours when act. */
  GRELRIGHT|       /* LeftEdge relative to the right border. */
  GRELHEIGHT,      /* Height relative to the height of the window. */
  GADGIMMEDIATE|   /* Activation, our program will recieve a message */
  RELVERIFY|       /* when the user has selected this gadget, and when */
                   /* the user has released it. We will also recieve a */
  FOLLOWMOUSE,     /* message when the mouse moves while this gadget is */
                   /* activated. */
  PROPGADGET|      /* GadgetType, a Proportional gadget. */
  GZZGADGET,       /* Put the gadget in the Outer window. */
  (APTR) &my_right_image, /* GadgetRender, the knob's Image structure. */
  NULL,            /* SelectRender, NULL since we do not supply the */
                   /* gadget with an alternative image. */
  NULL,            /* GadgetText, no text. */
  NULL,            /* MutualExclude, no mutual exclude. */
  (APTR) &my_right_prop_info, /* SpecialInfo, our PropInfo structure. */
  0,               /* GadgetID, no id. */
  NULL             /* UserData, no user data connected to the gadget. */
};



/************************************************/
/* THE BOTTOM PROPORTIONAL GADGET's STRUCTURES: */
/************************************************/

/* We need to declare an Image structure for the knob, but since */
/* Intuition will take care of the size etc of the knob, we do not need */
/* to initialize the Image structure: */
struct Image my_bottom_image;

struct PropInfo my_bottom_prop_info=
{
  FREEHORIZ|      /* Flags, the knob should be moved horizontally, and */
  AUTOKNOB,       /* Intuition should take care of the knob image. */
  0,              /* HorizPot, start position of the knob. */
  0,              /* VertPot, 0 since we will not move the knob ver. */
  MAXBODY * WIDTH / MAX_WIDTH, /* HorizBody. */
  0,              /* VertBody, 0 since we will not move the knob ver. */

  /* These variables are initialized and maintained by Intuition: */

  0,              /* CWidth */
  0,              /* CHeight */
  0, 0,           /* HPotRes, VPotRes */
  0,              /* LeftBorder */
  0               /* TopBorder */
};

struct Gadget my_bottom_gadget=
{
  &my_right_gadget,/* NextGadget, no more gadgets in the list. */
    1,             /* LeftEdge, 1 pixel out from the left side. */
   -8,             /* TopEdge, 8 lines above the bottom border. */
  -15,             /* Width, 15 pixels less wide than the window. */
    9,             /* Height, 9 lines heigh. */
  GADGHCOMP|       /* Flags, complement the colours when act. */
  GRELBOTTOM|      /* TopEdge relative to the bottom border. */
  GRELWIDTH,       /* Width relative to the width of the window. */
  GADGIMMEDIATE|   /* Activation, our program will recieve a message */
  RELVERIFY|       /* when the user has selected this gadget, and when */
                   /* the user has released it. We will also recieve a */
  FOLLOWMOUSE|     /* message when the mouse moves while this gadget is */
                   /* activated. */
  BOTTOMBORDER,    /* Make the bottom border of the window big enough */
                   /* for this gadge. */
  PROPGADGET|      /* GadgetType, a Proportional gadget. */
  GZZGADGET,       /* Put the gadget in the Outer window. */
  (APTR) &my_bottom_image,/* GadgetRender, the knob's Image structure. */
  NULL,            /* SelectRender, NULL since we do not supply the */
                   /* gadget with an alternative image. */
  NULL,            /* GadgetText, no text. */
  NULL,            /* MutualExclude, no mutual exclude. */
  (APTR) &my_bottom_prop_info, /* SpecialInfo, our PropInfo structure. */
  0,               /* GadgetID, no id. */
  NULL             /* UserData, no user data connected to the gadget. */
};



/*****************************/
/* OPEN A SUPERBITMAP WINDOW: */
/*****************************/

/***********************************************************/
/* 1. Declare and initialize a NewWindow structure with your */
/*    requirements:                                         */
/***********************************************************/

/* Declare a pointer to a Window structure: */
struct Window *my_window;

/* Declare and initialize your NewWindow structure: */
struct NewWindow my_new_window=
{
  10,            /* LeftEdge    x position of the window. */
  30,            /* TopEdge     y positio of the window. */
  WIDTH,         /* Width       200 pixels wide. */
  HEIGHT,        /* Height      dsfsafsadfdsafsad50 lines high. */
  0,             /* DetailPen   Text should be drawn with colour reg. 0 */
  1,             /* BlockPen    Blocks should be drawn with colour reg. 1 */
  CLOSEWINDOW|   /* IDCMPFlags  The window will give us a message if the */
                 /*             user has selected the Close window gad, */
  GADGETDOWN|    /*             or a gadget has been pressed on, or */
  GADGETUP|      /*             a gadge has been released, or */
  NEWSIZE|       /*             the user has changed the size or */
  MOUSEMOVE,     /*             the mouse moved while a gadget was act. */
  SUPER_BITMAP|  /* Flags       SuperBitMap. (No refreshing necessary) */
  GIMMEZEROZERO| /*             It is also a Gimmezerozero window. */
  WINDOWCLOSE|   /*             Close Gadget. */
  WINDOWDRAG|    /*             Drag gadget. */
  WINDOWDEPTH|   /*             Depth arrange Gadgets. */
  WINDOWSIZING|  /*             Sizing Gadget. */
```

```
  ACTIVATE,      /*             The window should be Active when opened. */
  &my_bottom_gadget, /* FirstGadget  Pointer to the first gadget. */
  NULL,          /* CheckMark   Use Intuition's default CheckMark (v). */
  "SuperBitMap", /* Title       Title of the window. */
  NULL,          /* Screen      Connected to the Workbench Screen. */
  NULL,          /* BitMap      We will change this later. */
  50,            /* MinWidth    We will not allow the window to become */
  50,            /* MinHeight   smaller than 50 x 50, and not bigger */
  MAX_WIDTH,     /* MaxWidth    than MAX_WIDTH x MAX_HEIGHT. */
  MAX_HEIGHT,    /* MaxHeight */
  WBENCHSCREEN   /* Type        Connected to the Workbench Screen. */
};



/**********************************/
/* 2. Declare a BitMap structure: */
/**********************************/

struct BitMap my_bitmap;



main()
{
  /* Boolean variable used for the while loop: */
  BOOL close_me;

  BOOL fix_window;

  int x, y;
  int new_x, new_y;
  int delta_x, delta_y;

  /* Declare two pointers which the ScrollLayer() function needs: */
  struct Layer *my_layer;
  struct Layer_Info *my_layer_info;

  /* Declare a variable in which we will store the IDCMP flag: */
  ULONG class;

  /* Declare a pointer to an IntuiMessage structure: */
  struct IntuiMessage *my_message;

  /* Variable used for the loops: */
  int loop;



  /* Before we can use Intuition we need to open the Intuition Library: */
  IntuitionBase = (struct IntuitionBase *)
    OpenLibrary( "intuition.library", 0 );

  if( IntuitionBase == NULL )
    exit(); /* Could NOT open the Intuition Library! */



  /* Before we can use the function AllocRaster() etc we need to open */
  /* the graphics Library. (See chapter "Amiga C" for more information) */
  GfxBase = (struct GfxBase *)
    OpenLibrary( "graphics.library", 0);
```

```
if( GfxBase == NULL )
{
  /* Could NOT open the Graphics Library! */

  /* Close the Intuition Library since we have opened it: */
  CloseLibrary( IntuitionBase );

  exit();
}


/* Before we can use the function ScrollLayer() etc we need to open */
/* the layers Library. (See chapter "Amiga C" for more information) */
LayersBase = (struct LayersBase *)
  OpenLibrary( "layers.library", 0);

if( LayersBase == NULL )
{
  /* Could NOT open the Layers Library! */

  /* Close the Graphics Library since we have opened it: */
  CloseLibrary( GfxBase );

  /* Close the Intuition Library since we have opened it: */
  CloseLibrary( IntuitionBase );

  exit();
}



/**********************************************************/
/* 3. Initialize your own BitMap by calling the function: */
/**********************************************************/

InitBitMap( &my_bitmap, DEPTH, MAX_WIDTH, MAX_HEIGHT );

/* &my_bitmap: A pointer to the my_bitmap structure. */
/* DEPTH:      Number of bitplanes to use. */
/* MAX_WIDTH:  The width of the BitMap. */
/* MAX_HEIGHT: The height of the BitMap. */



/**********************************************/
/* 4. Allocate display memory for the BitMap: */
/**********************************************/

for( loop=0; loop < DEPTH; loop++)
  if((my_bitmap.Planes[loop] = (PLANEPTR)
    AllocRaster( MAX_WIDTH, MAX_HEIGHT )) == NULL )
  {
    /* PANIC! Not enough memory */

    /* Deallocate the display memory, Bitplan by Bitplan. */
    for( loop=0; loop < DEPTH; loop++)
      if( my_bitmap.Planes[loop] ) /* Deallocate this Bitplan? */
        FreeRaster( my_bitmap.Planes[loop], MAX_WIDTH, MAX_HEIGHT );

    /* Close the Layers Library since we have opened it: */
    CloseLibrary( LayersBase );
```

```
    /* Close the Graphics Library since we have opened it: */
    CloseLibrary( GfxBase );

    /* Close the Intuition Library since we have opened it: */
    CloseLibrary( IntuitionBase );

    exit();
  }


/***************************/
/* 5. Clear all Bitplanes: */
/***************************/

for( loop=0; loop < DEPTH; loop++)
  BltClear( my_bitmap.Planes[loop], RASSIZE( MAX_WIDTH, MAX_HEIGHT ), 0);

/* The memory we allocated for the Bitplanes, is normaly "dirty", and */
/* therefore needs cleaning. We can here use the Blitter to clear the */
/* memory since it is the fastest way to do it, and the easiest. */
/* RASSIZE is a macro which calculates memory size for a Bitplane of */
/* the size WIDTH x HEIGHT. We will later go into more details about */
/* these functions etc, so do not worry about them... yet. */



/*******************************************************************/
/* 6. Make sure the NewWindow's BitMap pointer is pointing to your */
/*    BitMap structure:                                            */
/*******************************************************************/

my_new_window.BitMap=&my_bitmap;



/***************************************/
/* 7. At last you can open the window: */
/***************************************/

my_window = (struct Window *) OpenWindow( &my_new_window );

/* Have we opened the window succesfully? */
if(my_window == NULL)
{
  /* Could NOT open the Window! */

  /* Deallocate the display memory, Bitplan by Bitplan. */
  for( loop=0; loop < DEPTH; loop++)
    if( my_bitmap.Planes[loop] ) /* Deallocate this Bitplan? */
      FreeRaster( my_bitmap.Planes[loop], MAX_WIDTH, MAX_HEIGHT );

  /* Close the Layers Library since we have opened it: */
  CloseLibrary( LayersBase );

  /* Close the Graphics Library since we have opened it: */
  CloseLibrary( GfxBase );

  /* Close the Intuition Library since we have opened it: */
  CloseLibrary( IntuitionBase );
  exit();
}



/* We have opened the window, and everything seems to be OK. */



/* Initialize the two pointers which will be used by the ScrollLayer */
/* function: */
my_layer_info=&(my_window->WScreen->LayerInfo);
my_layer=my_window->RPort->Layer;



/* We will now draw some boxes in different colours: */
draw_some_boxes();



/* We can for the moment see the top left corner of the BitMap: */
x=0;
y=0;

/* The window does not need to be redrawn: */
fix_window=FALSE;

/* The user wants to run the program for the momnt. */
close_me = FALSE;

/* Stay in the while loop until the user has selected the Close window */
/* gadget: */
while( close_me == FALSE )
{
  /* Wait until we have recieved a message: */
  Wait( 1 << my_window->UserPort->mp_SigBit );

  /* We have now recieved one or more messages. */

  /* Since we may recieve several messages we stay in the while loop */
  /* and collect, save, reply and execute the messages until there is */
  /* a pause: */
  while(my_message=(struct IntuiMessage *)GetMsg( my_window->UserPort))
  {
    /* GetMsg will return a pointer to a message if there was one, */
    /* else it returns NULL. We will therefore stay in this while loop */
    /* as long as there are some messages waiting in the port. */

    /* After we have collected the message we can read it, and save */
    /* any important values which we maybe want to check later: */
    class = my_message->Class;      /* Save the IDCMP flag. */

    /* After we have read it we reply as fast as possible: */
    /* REMEMBER! Do never try to read a message after you have replied! */
    /* Some other process has maybe changed it. */
    ReplyMsg( my_message );

    /* Check which IDCMP flag was sent: */
    switch( class )
    {
      case CLOSEWINDOW:  /* The user selected the Close window gadget! */
```

```
            close_me=TRUE;
            break;

     case MOUSEMOVE:    /* The user moved the mouse while one of the */
                        /* Proportional gadgets was activated: */
            fix_window=TRUE; /* Redraw the display. */
            break;

     case NEWSIZE:      /* The user has resized the window: */
            /* Change size of the knobs: */
            ModifyProp
            (
              &my_right_gadget,           /* Pointer to the gadget. */
              my_window,                  /* Pointer to the window. */
              NULL,                       /* Not a requester gadget. */
              my_right_prop_info.Flags,   /* Flags, no change. */
              0,                          /* HorizPot */
              my_right_prop_info.VertPot, /* VertPot, no change. */
              0,                          /* HorizBody */
                                          /* VertBody: */
              (ULONG) MAXBODY*my_window->Height/MAX_HEIGHT
            );
            ModifyProp
            (
              &my_bottom_gadget,            /* Pointer to the gadget. */
              my_window,                    /* Pointer to the window. */
              NULL,                         /* Not a req. gadget. */
              my_bottom_prop_info.Flags,    /* Flags, no change. */
              my_bottom_prop_info.HorizPot, /* HorizPot, no change. */
              0,                            /* VertPot */
                                            /* HorizBody: */
              (ULONG) MAXBODY*my_window->Width/MAX_WIDTH,
              0                             /* VertBody: */
            );
            fix_window=TRUE; /* Redraw the display. */
            break;

     case GADGETDOWN:   /* The user has selected one of the gadgets: */
            fix_window=TRUE; /* Redraw the display. */
            break;

     case GADGETUP:     /* The user has released one of the gadgets: */
            fix_window=TRUE; /* Redraw the display. */
            break;
   }
 }



/* Should we update the window's display? */
if(fix_window)
{
  fix_window=FALSE;

  /* Calculate what part of the BitMap we should display: */
  new_x= (MAX_WIDTH - my_bottom_prop_info.HorizBody / (float) MAXBODY
         * MAX_WIDTH) * my_bottom_prop_info.HorizPot / (float) MAXPOT;

  new_y= (MAX_HEIGHT - my_right_prop_info.VertBody / (float) MAXBODY
         * MAX_HEIGHT) * my_right_prop_info.VertPot / (float) MAXPOT;

  delta_x=new_x-x;
```

```
      delta_y=new_y-y;

      x=new_x;
      y=new_y;

      ScrollLayer( my_layer_info, my_layer, delta_x, delta_y );
    }
  }



  /********************************************************************/
  /* 8. Do not forget to close the window, AND deallocate the display */
  /*    memory:                                                       */
  /********************************************************************/

  /* We should always close the windows we have opened before we leave: */
  CloseWindow( my_window );

  /* Deallocate the display memory, Bitplan by Bitplan. */
  for( loop=0; loop < DEPTH; loop++)
    if( my_bitmap.Planes[loop] ) /* Deallocate this Bitplan? */
      FreeRaster( my_bitmap.Planes[loop], MAX_WIDTH, MAX_HEIGHT );



  /* Close the Layers Library since we have opened it: */
  CloseLibrary( LayersBase );

  /* Close the Graphics Library since we have opened it: */
  CloseLibrary( GfxBase );

  /* Close the Intuition Library since we have opened it: */
  CloseLibrary( IntuitionBase );

  /* THE END */
}



/* This function draws some coloured boxes: */
/* Returns nothing. */
void draw_some_boxes()
{
  int x, y;
  UBYTE colour;


  colour=1; /* Set colour to 1, white. */

  /* Set Draw Mode to normal: */
  SetDrMd( my_window->RPort, JAM1 );

  for(x=0; x < MAX_WIDTH/40-2; x++)
    for(y=0; y < MAX_HEIGHT/20-2; y++)
    {
      /* New colour to draw with */
      SetAPen( my_window->RPort, colour );

      colour++;

      /* If colour is bigger than 3 (Orange) we change colour to 1 */
      /* (white) again. (The boxes will therefore be drawn with the */
      /* colours white, black, orange: */
      if(colour > 3)
        colour=1;

      Move( my_window->RPort, x*40+40, y*20+20 ); /* Top left corner. */
      Draw( my_window->RPort, x*40+72, y*20+20 ); /* Out to the right. */
      Draw( my_window->RPort, x*40+72, y*20+36 ); /* Down. */
      Draw( my_window->RPort, x*40+40, y*20+36 ); /* Back to the left. */
      Draw( my_window->RPort, x*40+40, y*20+20 ); /* Up again. */
    }
}
```

A.5 REQUESTERS

Example1

This example opens a Simple requester by calling the function AutoRequest. It displays a message "This is a very simple requester!", and has only one gadget connected to it (on the right side of the requester) with the text "OK".

```c
/* Example1                                                          */
/* This example opens a Simple requester by calling the function     */
/* AutoRequest. It displays a message "This is a very simple         */
/* requester!", and has one gadget connected to it (on the right side) */
/* with the text "OK".                                               */



#include <intuition/intuition.h>



struct IntuitionBase *IntuitionBase;



/* The body text for the requester: */
struct IntuiText my_body_text=
{
  0,       /* FrontPen, colour 0 (blue). */
  0,       /* BackPen, not used since JAM1. */
  JAM1,    /* DrawMode, do not change the background. */
  15,      /* LedtEdge, 15 pixels out. */
  5,       /* TopEdge, 5 lines down. */
  NULL,    /* ITextFont, default font. */
  "This is a very simple requester!", /* IText, the text . */
  NULL,    /* NextText, no more IntuiText structures link. */
};

/* The OK text: */
struct IntuiText my_ok_text=
{
  0,       /* FrontPen, colour 0 (blue). */
  0,       /* BackPen, not used since JAM1. */
  JAM1,    /* DrawMode, do not change the background. */
  6,       /* LedtEdge, 6 pixels out. */
  3,       /* TopEdge, 3 lines down. */
  NULL,    /* ITextFont, default font. */
  "OK",    /* IText, the text that will be printed. */
  NULL,    /* NextText, no more IntuiText structures link. */
};



main()
{
  /* Before we can use Intuition we need to open the Intuition Library: */
  IntuitionBase = (struct IntuitionBase *)
    OpenLibrary( "intuition.library", 0 );

  if( IntuitionBase == NULL )
    exit(); /* Could NOT open the Intuition Library! */



  AutoRequest(NULL, &my_body_text, NULL, &my_ok_text, NULL, NULL, 320, 72);

  /*********************************************************************/
  /* NULL,              no pointer to a window structure.               */
  /* &my_body_text,     pointer to a IntuiText str. cont. the body text  */
  /* NULL,              no gadget on the right side.                    */
  /* &my_ok_text,       pointer to a IntuiText str. cont. the neg. text  */
```

```c
  /* NULL,              no gadget on the right side.                    */
  /* NULL,              IDCMP flags which will satisfy the negative gad. */
  /* 320,               Width, 320 pixels wide.                         */
  /* 72,                Height, 72 lines high.                          */
  /*                                                                    */
  /* Intuition will automatically set the IDCMP flag RELVERIFY for both  */
  /* of the gadgets, so we do not need to set any IDCMP flags if we do   */
  /* not want to.                                                       */
  /*                                                                    */
  /* The requester will look like this:                                 */
  /*                                                                    */
  /* ---------------------------------------                            */
  /* | System Request ================[*][*]                            */
  /* ---------------------------------------                            */
  /* | This is a very simple requester! |  |                            */
  /* |                                  |  |                            */
  /* |                                  |  |                            */
  /* |                           ------ |  |                            */
  /* |                           | OK | |  |                            */
  /* |                           ------ |  |                            */
  /* ------------------------------------[*]                            */
  /*                                                                    */
  /**********************************************************************/



  /* Close the Intuition Library since we have opened it: */
  CloseLibrary( IntuitionBase );

  /* THE END */
}
```

Example2

Same as Example1, except that the requester displays a message "Do you really want to quit?", and allows the user to choose between "Yes" and "No". The program will continue to reopen the requester until the user has chosen "Yes".

```
/* Example2                                                         */
/* This example opens a Simple requester by calling the function        */
/* AutoRequest. It displays a message "Do you really want to quit?",    */
/* and allows the user to choose between "Yes" and "No". The program    */
/* will continue to open the requester until the user has chosen "Yes". */



#include <intuition/intuition.h>



struct IntuitionBase *IntuitionBase;



/* The body text for the requester: */
struct IntuiText my_body_text=
{
  0,       /* FrontPen, colour 0 (blue). */
  0,       /* BackPen, not used since JAM1. */
  JAM1,    /* DrawMode, do not change the background. */
  15,      /* LedtEdge, 15 pixels out. */
  5,       /* TopEdge, 5 lines down. */
  NULL,    /* ITextFont, default font. */
  "Do you really want to quit?", /* IText, the text that will be printed. */
  NULL,    /* NextText, no more IntuiText structures link. */
};

/* The positive text: */
/* (Printed inside the left gadget) */
struct IntuiText my_positive_text=
{
  0,       /* FrontPen, colour 0 (blue). */
  0,       /* BackPen, not used since JAM1. */
  JAM1,    /* DrawMode, do not change the background. */
  6,       /* LedtEdge, 6 pixels out. */
  3,       /* TopEdge, 3 lines down. */
  NULL,    /* ITextFont, default font. */
  "Yes",   /* IText, the text that will be printed. */
  NULL,    /* NextText, no more IntuiText structures link. */
};

/* The negative text: */
/* (Printed inside the right gadget) */
struct IntuiText my_negative_text=
{
  0,       /* FrontPen, colour 0 (blue). */
  0,       /* BackPen, not used since JAM1. */
  JAM1,    /* DrawMode, do not change the background. */
  6,       /* LedtEdge, 6 pixels out. */
  3,       /* TopEdge, 3 lines down. */
  NULL,    /* ITextFont, default font. */
  "No",    /* IText, the text that will be printed. */
  NULL,    /* NextText, no more IntuiText structures link. */
};



main()
{
  /* Before we can use Intuition we need to open the Intuition Library: */
  IntuitionBase = (struct IntuitionBase *)
    OpenLibrary( "intuition.library", 0 );

  if( IntuitionBase == NULL )
    exit(); /* Could NOT open the Intuition Library! */



  while( !AutoRequest( NULL, &my_body_text, &my_positive_text,
                       &my_negative_text, NULL, NULL, 320, 72) );

  /*********************************************************************/
  /* NULL,              no pointer to a window structure.              */
  /* &my_body_text,     pointer to a IntuiText str. cont. the body text */
  /* &my_positive_text, pointer to a IntuiText str. cont. the pos. text */
  /* &my_negative_text, pointer to a IntuiText str. cont. the neg. text */
  /* NULL,              IDCMP flags which will satisfy the positive gad. */
  /* NULL,              IDCMP flags which will satisfy the negative gad. */
  /* 320,               Width, 320 pixels wide.                        */
  /* 72,                Height, 72 lines high.                         */
  /*                                                                   */
  /* Intuition will automatically set the IDCMP flag RELVERIFY for both */
  /* of the gadgets, so we do not need to set any IDCMP flags if we do  */
  /* not want to.                                                      */
  /*                                                                   */
  /* while( !AutoRequest(...) );                                       */
  /* Since AutoRequest returns TRUE ("Yes") or FALSE ("No") we neggate  */
  /* it (!), and can then use the statement in a while loop. As long as */
  /* the user selects the "No" gadget AutoRequest returns FALSE which   */
  /* is changed into TRUE, and we stay in the while loop. When the user, */
  /* on the other hand, selects the "Yes" gadget AutoRequest() returns  */
  /* TRUE, changed into FALSE, and we leave the while loop.            */
  /*                                                                   */
  /* The requester will look like this:                                */
  /*                                                                   */
  /* ---------------------------------------                           */
  /* | System Request ================[*][*]                           */
  /* ---------------------------------------                           */
  /* | Do you really want to quit?      |  |                           */
  /* |                                  |  |                           */
  /* |                                  |  |                           */
  /* | -------                   ------ |  |                           */
  /* | | Yes |                   | No | |  |                           */
  /* | -------                   ------ |  |                           */
  /* ----------------------------------[*]                             */
  /*                                                                   */
  /*********************************************************************/



  /* Close the Intuition Library since we have opened it: */
  CloseLibrary( IntuitionBase );

  /* THE END */
}
```

Example3

Same as Example1, except that this requester displays a message "Insert a disk in any drive!", and allows the user to choose between "Yes" and "No". The program will continue to reopen the requester until the user has chosen "Yes" or inserted a disk.

```
/* Example3                                                          */
/* This example opens a Simple requester by calling the function     */
/* AutoRequest. It displays a message "Insert a disk in any drive!", */
/* and allows the user to choose between "Yes" and "No". The program */
/* will continue to open the requester until the user has chosen "Yes", */
/* or the user has inserted a disk.                                  */



#include <intuition/intuition.h>



struct IntuitionBase *IntuitionBase;



/* The "body" text for the requester: */
struct IntuiText my_body_text=
{
  0,       /* FrontPen, colour 0 (blue). */
  0,       /* BackPen, not used since JAM1. */
  JAM1,    /* DrawMode, do not change the background. */
  15,      /* LedtEdge, 15 pixels out. */
  5,       /* TopEdge, 5 lines down. */
  NULL,    /* ITextFont, default font. */
  "Insert a disk in any drive!", /* IText, the body text. */
  NULL,    /* NextText, no more IntuiText structures link. */
};

/* The positive text: */
/* (Printed inside the left gadget) */
struct IntuiText my_positive_text=
{
  0,       /* FrontPen, colour 0 (blue). */
  0,       /* BackPen, not used since JAM1. */
  JAM1,    /* DrawMode, do not change the background. */
  6,       /* LedtEdge, 6 pixels out. */
  3,       /* TopEdge, 3 lines down. */
  NULL,    /* ITextFont, default font. */
  "Yes",   /* IText, the text that will be printed. */
  NULL,    /* NextText, no more IntuiText structures link. */
};

/* The negative text: */
/* (Printed inside the right gadget) */
struct IntuiText my_negative_text=
{
  0,       /* FrontPen, colour 0 (blue). */
  0,       /* BackPen, not used since JAM1. */
  JAM1,    /* DrawMode, do not change the background. */
  6,       /* LedtEdge, 6 pixels out. */
  3,       /* TopEdge, 3 lines down. */
  NULL,    /* ITextFont, default font. */
  "No",    /* IText, the text that will be printed. */
  NULL,    /* NextText, no more IntuiText structures link. */
};



main()
{
  /* Before we can use Intuition we need to open the Intuition Library: */
  IntuitionBase = (struct IntuitionBase *)
    OpenLibrary( "intuition.library", 0 );

  if( IntuitionBase == NULL )
    exit(); /* Could NOT open the Intuition Library! */



  while( !AutoRequest( NULL, &my_body_text, &my_positive_text,
                       &my_negative_text, DISKINSERTED, NULL, 320, 72) );

  /*********************************************************************/
  /* NULL,              no pointer to a window structure.               */
  /* &my_body_text,     pointer to a IntuiText str. cont. the body text  */
  /* &my_positive_text, pointer to a IntuiText str. cont. the pos. text  */
  /* &my_negative_text, pointer to a IntuiText str. cont. the neg. text  */
  /* DISKINSERTED,      IDCMP flags which will satisfy the positive gad. */
  /* NULL,              IDCMP flags which will satisfy the negative gad. */
  /* 320,               Width, 320 pixels wide.                          */
  /* 72,                Height, 72 lines high.                           */
  /*                                                                     */
  /* Intuition will automatically set the IDCMP flag RELVERIFY for both  */
  /* of the gadgets, so we do not need to set the DISKINSERTED flag for  */
  /* the "positive" gadget.                                              */
  /*                                                                     */
  /* while( !AutoRequest(...) );                                         */
  /* Since AutoRequest returns TRUE ("Yes") or FALSE ("No") we neggate   */
  /* it (!), and can then use the statement in a while loop. As long as  */
  /* the user selects the "No" gadget AutoRequest returns FALSE which    */
  /* is changed into TRUE, and we stay in the while loop. When the user, */
  /* on the other hand, selects the "Yes" gadget, or inserts a disk,     */
  /* AutoRequest() returns TRUE, changed into FALSE, and we leave the    */
  /* while loop.                                                         */
  /*                                                                     */
  /* The requester will look like this:                                  */
  /*                                                                     */
  /* ---------------------------------------                             */
  /* | System Request ================[*][*]                             */
  /* ---------------------------------------                             */
  /* | Insert a disk in any drive!      |  |                             */
  /* |                                  |  |                             */
  /* |                                  |  |                             */
  /* | -------                   ------ |  |                             */
  /* | | Yes |                   | No | |  |                             */
  /* | -------                   ------ |  |                             */
  /* ------------------------------------[*]                             */
  /*                                                                     */
  /*********************************************************************/



  /* Close the Intuition Library since we have opened it: */
  CloseLibrary( IntuitionBase );

  /* THE END */
}
```

Example4

This program will open a normal window which is connected to the Workbench Screen. The window will use all System Gadgets, and will close first when the user has selected the System gadget Close window. Inside the window we have activated an Application requester with a connecting gadget. The requester will first be satisfied when the user has selected the gadget, and will then be deactivated. The window can now be closed.

```
/* Example4                                                        */
/* This program will open a normal window which is connected to the     */
/* Workbench Screen. The window will use all System Gadgets, and will   */
/* close first when the user has selected the System gadget Close       */
/* window. Inside the window we have activated an Application requester */
/* with a connecting gadget. The requester will first be satisfied when */
/* the user has selected the gadget, and will then be deactivated. The  */
/* window can now be closed.                                            */



#include <intuition/intuition.h>



struct IntuitionBase *IntuitionBase;



/***************/
/* THE GADGET: */
/***************/

/* The coordinates for the box: */
SHORT gadget_border_points[]=
{
   0,  0, /* Start at position (0,0) */
  70,  0, /* Draw a line to the right to position (70,0) */
  70, 10, /* Draw a line down to position (70,10) */
   0, 10, /* Draw a line to the right to position (0,10) */
   0,  0  /* Finish of by drawing a line up to position (0,0) */
};

/* The Border structure: */
struct Border gadget_border=
{
  0, 0,        /* LeftEdge, TopEdge. */
  1,           /* FrontPen, colour register 1. */
  0,           /* BackPen, for the moment unused. */
  JAM1,        /* DrawMode, draw the lines with colour 1. */
  5,           /* Count, 5 pair of coordinates in the array. */
  gadget_border_points, /* XY, pointer to the array with the coord. */
  NULL,        /* NextBorder, no other Border structures are connected. */
};

/* The IntuiText structure: */
struct IntuiText gadget_text=
{
  1,         /* FrontPen, colour register 1. */
  0,         /* BackPen, colour register 0. */
  JAM1,      /* DrawMode, draw the characters with colour 1, do not */
             /* change the background. */
  4, 2,      /* LeftEdge, TopEdge. */
  NULL,      /* ITextFont, use default font. */
  "PRESS ME",/* IText, the text that will be printed. */
  NULL,      /* NextText, no other IntuiText structures are connected. */
};

struct Gadget requester_gadget=
{
  NULL,          /* NextGadget, no more gadgets in the list. */
  40,            /* LeftEdge, 40 pixels out. */
  20,            /* TopEdge, 20 lines down. */
  71,            /* Width, 71 pixels wide. */
  11,            /* Height, 11 pixels lines heigh. */
  GADGHCOMP,     /* Flags, when this gadget is highlighted, the gadget */
                 /* will be rendered in the complement colours. */
                 /* (Colour 0 (00) will be changed to colour 3 (11) */
                 /* (Colour 1 (01)           - " -           2 (10) */
                 /* (Colour 2 (10)           - " -           1 (01) */
                 /* (Colour 3 (11)           - " -           0 (00) */
  GADGIMMEDIATE| /* Activation, our program will recieve a message when */
  RELVERIFY|     /* the user has selected this gadget, and when the user */
                 /* has released it. */
  ENDGADGET,     /* When the user has selected this gadget, the */
                 /* requester is satisfied, and is deactivated. */
                 /* IMPORTANT! At least one gadget per requester */
                 /* must have the flag ENDGADGET set. If not, the */
                 /* requester would never be deactivated! */

  BOOLGADGET|    /* GadgetType, a Boolean gadget which is connected to */
  REQGADGET,     /* a requester. IMPORTANT! Every gadget which is */
                 /* connectd to a requester must have the REQGADGET flsg */
                 /* set in the GadgetType field. */
  (APTR) &gadget_border, /* GadgetRender, a pointer to our Border struc. */
  NULL,          /* SelectRender, NULL since we do not supply the gadget */
                 /* with an alternative image. (We complement the */
                 /* colours instead) */
  &gadget_text,  /* GadgetText, a pointer to our IntuiText structure. */
                 /* (See chapter 3 GRAPHICS for more information) */
  NULL,          /* MutualExclude, no mutual exclude. */
  NULL,          /* SpecialInfo, NULL since this is a Boolean gadget. */
                 /* (It is not a Proportional/String or Integer gdget) */
  0,             /* GadgetID, no id. */
  NULL           /* UserData, no user data connected to the gadget. */
};

/***********************************************************************/
/* Important notice:                                                   */
/* Remember that every gadget which is connected to a requester must   */
/* have the flag REQGADGET set in the GadgetType field. Remember also  */
/* that at least one gadget per requester must have the ENDGADGET flag */
/* set in the Activation field.                                        */
/***********************************************************************/



/************************************/
/* THE BORDER AROUND THE REQUESTER: */
/************************************/

/* The coordinates for the box around the requester: */
SHORT requester_border_points[]=
{
    0,  0, /* Start at position (0,0) */
  319,  0, /* Draw a line to the right to position (319,0) */
  319, 99, /* Draw a line down to position (319,99) */
    0, 99, /* Draw a line to the right to position (319,99) */
    0,  0  /* Finish of by drawing a line up to position (0,0) */
};

/* The Border structure for the requester: */
struct Border requester_border=
{
  0, 0,        /* LeftEdge, TopEdge. */
  1,           /* FrontPen, colour register 1. */
  0,           /* BackPen, for the moment unused. */
  JAM1,        /* DrawMode, draw the lines with colour 1. */
  5,           /* Count, 5 pair of coordinates in the array. */
  requester_border_points, /* XY, pointer to the array with the coord. */
  NULL,        /* NextBorder, no other Border structures are connected. */
};



/**********************************/
/* THE TEXT INSIDE THE REQUESTER: */
/**********************************/

/* The IntuiText structure used to print some text inside the requester: */
struct IntuiText requester_text=
{
  1,         /* FrontPen, colour register 1. */
  0,         /* BackPen, unused since JAM1. */
  JAM1,      /* DrawMode, draw the characters with colour 1, do not */
             /* change the background. */
  4, 2,      /* LeftEdge, TopEdge. */
  NULL,      /* ITextFont, use default font. */
  "This is the requester!", /* IText, the text that will be printed. */
  NULL,      /* NextText, no other IntuiText structures are connected. */
};



struct Requester my_requester=
{
  NULL,              /* OlderRequester, used by Intuition. */
  40, 20,            /* LeftEdge, TopEdge, 40 pixels out, 20 lines down. */
  320, 100,          /* Width, Height, 320 pixels wide, 100 lines high. */
  0, 0,              /* RelLeft, RelTop, Since POINTREL flag is not set, */
                     /* Intuition ignores these values. */
  &requester_gadget, /* ReqGadget, pointer to the first gadget. */
  &requester_border, /* ReqBorder, pointer to a Border structure. */
  &requester_text,   /* ReqText, pointer to a IntuiText structure. */
  NULL,              /* Flags, no flags set. */
  3,                 /* BackFill, draw everything on an orange backgr. */
  NULL,              /* ReqLayer, used by Intuition. Set to NULL. */
  NULL,              /* ReqPad1, used by Intuition. Set to NULL. */
  NULL,              /* ImageBMap, no predrawn Bitmap. Set to NULL. */
                     /*           (The PREDRAWN flag was not set) */
  NULL,              /* RWindow, used by Intuition. Set to NULL. */
  NULL               /* ReqPad2, used by Intuition. Set to NULL. */
};


/* Declare a pointer to a Window structure: */
struct Window *my_window;

/* Declare and initialize your NewWindow structure: */
struct NewWindow my_new_window=
{
  0,             /* LeftEdge    x position of the window. */
  0,             /* TopEdge     y positio of the window. */
  640,           /* Width       640 pixels wide. */
  200,           /* Height      200 lines high. */
  0,             /* DetailPen   Text should be drawn with colour reg. 0 */
  1,             /* BlockPen    Blocks should be drawn with colour reg. 1 */
  CLOSEWINDOW|   /* IDCMPFlags  The window will give us a message if the */
                 /*             user has selected the Close window gad, */
  GADGETDOWN|    /*             or a gadget has been pressed on, or */
  GADGETUP,      /*             a gadge has been released. */
  SMART_REFRESH| /* Flags       Intuition should refresh the window. */
  WINDOWCLOSE|   /*             Close Gadget. */
  WINDOWDRAG|    /*             Drag gadget. */
  WINDOWDEPTH|   /*             Depth arrange Gadgets. */
  WINDOWSIZING|  /*             Sizing Gadget. */
  ACTIVATE,      /*             The window should be Active when opened. */
  NULL,          /* FirstGadget No gadget connected to this window. */
  NULL,          /* CheckMark   Use Intuition's default CheckMark. */
  "The Window",  /* Title       Title of the window. */
  NULL,          /* Screen      Connected to the Workbench Screen. */
  NULL,          /* BitMap      No Custom BitMap. */
  140,           /* MinWidth    We will not allow the window to become */
  50,            /* MinHeight   smaller than 140 x 50, and not bigger */
  300,           /* MaxWidth    than 300 x 200. */
  200,           /* MaxHeight */
  WBENCHSCREEN   /* Type        Connected to the Workbench Screen. */
};



main()
{
  /* Boolean variable used for the while loop: */
  BOOL close_me;

  /* Declare a variable in which we will store the IDCMP flag: */
  ULONG class;

  /* Declare a pointer to an IntuiMessage structure: */
  struct IntuiMessage *my_message;

  /* We use this variable to check if the requester has ben activated */
  /* or not: */
  BOOL result;


  /* Before we can use Intuition we need to open the Intuition Library: */
  IntuitionBase = (struct IntuitionBase *)
    OpenLibrary( "intuition.library", 0 );

  if( IntuitionBase == NULL )
    exit(); /* Could NOT open the Intuition Library! */



  /* We will now try to open the window: */
  my_window = (struct Window *) OpenWindow( &my_new_window );

  /* Have we opened the window succesfully? */
  if(my_window == NULL)
  {
    /* Could NOT open the Window! */

    /* Close the Intuition Library since we have opened it: */
    CloseLibrary( IntuitionBase );

    exit();
}



/* We have opened the window, and everything seems to be OK. */



/* We will now try to activate the requester: */
result=Request( &my_requester, my_window );

if( !result )  /* !result is the same thing as result==FALSE */
{
  /* Intuition could not activate the requester! */
  /* In this case we do not need to quit since it does not matter if */
  /* the requester was activated or not. I just wanted to show how */
  /* you can check if you have opened or not the requester. */

  printf("Could not activate the requester!\n");
}
else
{
  /* Intuition could open the requester! */
  printf("Try to close the window!\n");
}



close_me = FALSE;

/* Stay in the while loop until the user has selected the Close window */
/* gadget. However, in this example the user first need to deactivate */
/* the requester before he can select the Close window gadget: */
while( !close_me )
{
  /* Wait until we have recieved a message: */
  Wait( 1 << my_window->UserPort->mp_SigBit );

  /* As long as we collect messages sucessfully: */
  while(my_message=(struct IntuiMessage *) GetMsg(my_window->UserPort))
  {
    /* After we have collected the message we can read it, and save any */
    /* important values which we maybe want to check later: */
    class = my_message->Class;

    /* After we have read it we reply as fast as possible: */
    /* REMEMBER! Do never try to read a message after you have replied! */
    /* Some other process has maybe changed it. */
    ReplyMsg( my_message );

    /* Check which IDCMP flag was sent: */
    switch( class )
    {
      case CLOSEWINDOW:  /* The user selected the Close window gadget! */

             /* It is first when the requester has been satisfied, the */
             /* user can close the window. Remember, do never close a */
             /* window if there are any active requester in it. */

             close_me=TRUE;
             break;

      case GADGETDOWN:   /* The user has pressed on a gadget. */
             /* Since there exist only one "nomal" gadget, we do not */
             /* need to check which gadget was selected. */

             printf("Down\n");
             break;

      case GADGETUP:     /* The user has released a gadget. */
             /* Since there exist only one "nomal" gadget, we do not */
             /* need to check which gadget was released. */

             printf("Up\n");

             /* Once the user releases this gadget the requester will */
             /* be satisfied and deactivated. The user can from now on */
             /* select the Close window gadget. */

             printf("Requester satisfied!\n");
             printf("You may now close the window!\n");
             break;
    }
  }
}



/* We should always close the windows we have opened before we leave: */
CloseWindow( my_window );



/* Close the Intuition Library since we have opened it: */
CloseLibrary( IntuitionBase );

/* THE END */
}
```

Example5

Same as Example4, except that the requester is first activated when the user double-clicks on the right mouse button. This example shows how to create a Double-menu requester, and how to monitor the IDCMP flags REQSET and REQCLEAR.

```
/* Example5                                                         */
/* This program will open a normal window which is connected to the     */
/* Workbench Screen. The window will use all System Gadgets, and will   */
/* close first when the user has selected the System gadget Close       */
/* window. Whenever the user double-clicks on the right mouse button,   */
/* a Double-menue requester is activated. This example also shows how   */
/* to use the IDCMP flags REQSET and REQCLEAR.                          */



#include <intuition/intuition.h>



struct IntuitionBase *IntuitionBase;



/***************/
/* THE GADGET: */
/***************/

/* The coordinates for the box: */
SHORT gadget_border_points[]=
{
   0,  0, /* Start at position (0,0) */
  70,  0, /* Draw a line to the right to position (70,0) */
  70, 10, /* Draw a line down to position (70,10) */
   0, 10, /* Draw a line to the right to position (0,10) */
   0,  0  /* Finish of by drawing a line up to position (0,0) */
};

/* The Border structure: */
struct Border gadget_border=
{
  0, 0,        /* LeftEdge, TopEdge. */
  1,           /* FrontPen, colour register 1. */
  0,           /* BackPen, for the moment unused. */
  JAM1,        /* DrawMode, draw the lines with colour 1. */
  5,           /* Count, 5 pair of coordinates in the array. */
  gadget_border_points, /* XY, pointer to the array with the coord. */
  NULL,        /* NextBorder, no other Border structures are connected. */
};

/* The IntuiText structure: */
struct IntuiText gadget_text=
{
  1,         /* FrontPen, colour register 1. */
  0,         /* BackPen, colour register 0. */
  JAM1,      /* DrawMode, draw the characters with colour 1, do not */
             /* change the background. */
  4, 2,      /* LeftEdge, TopEdge. */
  NULL,      /* ITextFont, use default font. */
  "PRESS ME",/* IText, the text that will be printed. */
  NULL,      /* NextText, no other IntuiText structures are connected. */
};

struct Gadget requester_gadget=
{
  NULL,          /* NextGadget, no more gadgets in the list. */
  40,            /* LeftEdge, 40 pixels out. */
  20,            /* TopEdge, 20 lines down. */
  71,            /* Width, 71 pixels wide. */
  11,            /* Height, 11 pixels lines heigh. */
  GADGHCOMP,     /* Flags, when this gadget is highlighted, the gadget */
                 /* will be rendered in the complement colours. */
                 /* (Colour 0 (00) will be changed to colour 3 (11) */
                 /* (Colour 1 (01)           - " -           2 (10) */
                 /* (Colour 2 (10)           - " -           1 (01) */
                 /* (Colour 3 (11)           - " -           0 (00) */
  GADGIMMEDIATE| /* Activation, our program will recieve a message when */
  RELVERIFY|     /* the user has selected this gadget, and when the user */
                 /* has released it. */
  ENDGADGET,     /* When the user has selected this gadget, the */
                 /* requester is satisfied, and is deactivated. */
                 /* IMPORTANT! At least one gadget per requester */
                 /* must have the flag ENDGADGET set. If not, the */
                 /* requester would never be deactivated! */

  BOOLGADGET|    /* GadgetType, a Boolean gadget which is connected to */
  REQGADGET,     /* a requester. IMPORTANT! Every gadget which is */
                 /* connectd to a requester must have the REQGADGET flsg */
                 /* set in the GadgetType field. */
  (APTR) &gadget_border, /* GadgetRender, a pointer to our Border struc. */
  NULL,          /* SelectRender, NULL since we do not supply the gadget */
                 /* with an alternative image. (We complement the */
                 /* colours instead) */
  &gadget_text,  /* GadgetText, a pointer to our IntuiText structure. */
                 /* (See chapter 3 GRAPHICS for more information) */
  NULL,          /* MutualExclude, no mutual exclude. */
  NULL,          /* SpecialInfo, NULL since this is a Boolean gadget. */
                 /* (It is not a Proportional/String or Integer gdget) */
  0,             /* GadgetID, no id. */
  NULL           /* UserData, no user data connected to the gadget. */
};

/**********************************************************************/
/* Important notice:                                                  */
/* Remember that every gadget which is connected to a requester must   */
/* have the flag REQGADGET set in the GadgetType field. Remember also  */
/* that at least one gadget per requester must have the ENDGADGET flag */
/* set in the Activation field.                                       */
/**********************************************************************/



/************************************/
/* THE BORDER AROUND THE REQUESTER: */
/************************************/

/* The coordinates for the box around the requester: */
SHORT requester_border_points[]=
{
    0,  0, /* Start at position (0,0) */
  319,  0, /* Draw a line to the right to position (319,0) */
  319, 99, /* Draw a line down to position (319,99) */
    0, 99, /* Draw a line to the right to position (319,99) */
    0,  0  /* Finish of by drawing a line up to position (0,0) */
};

/* The Border structure for the requester: */
struct Border requester_border=
{
  0, 0,        /* LeftEdge, TopEdge. */
  1,          /* FrontPen, colour register 1. */
  0,          /* BackPen, for the moment unused. */
  JAM1,       /* DrawMode, draw the lines with colour 1. */
  5,          /* Count, 5 pair of coordinates in the array. */
  requester_border_points, /* XY, pointer to the array with the coord. */
  NULL,       /* NextBorder, no other Border structures are connected. */
};



/**********************************/
/* THE TEXT INSIDE THE REQUESTER: */
/**********************************/

/* The IntuiText structure used to print some text inside the requester: */
struct IntuiText requester_text=
{
  1,        /* FrontPen, colour register 1. */
  0,        /* BackPen, unused since JAM1. */
  JAM1,     /* DrawMode, draw the characters with colour 1, do not */
            /* change the background. */
  4, 2,     /* LeftEdge, TopEdge. */
  NULL,     /* ITextFont, use default font. */
  "This is the requester!", /* IText, the text that will be printed. */
  NULL,     /* NextText, no other IntuiText structures are connected. */
};



/* Note:                                                             */
/* This is the structure for the Double-menu requester, but as you have */
/* maybe noticed, it is exactly the same as a normal requester struc.   */
/* The diffrence is that we call the function SetDMRequest() instead    */
/* of calling the function Request().                                   */

struct Requester my_requester=
{
  NULL,              /* OlderRequester, used by Intuition. */
  40, 20,            /* LeftEdge, TopEdge, 40 pixels out, 20 lines down. */
  320, 100,          /* Width, Height, 320 pixels wide, 100 lines high. */
  0, 0,              /* RelLeft, RelTop, Since POINTREL flag is not set, */
                     /* Intuition ignores these values. */
  &requester_gadget, /* ReqGadget, pointer to the first gadget. */
  &requester_border, /* ReqBorder, pointer to a Border structure. */
  &requester_text,   /* ReqText, pointer to a IntuiText structure. */
  NULL,              /* Flags, no flags set. */
  3,                 /* BackFill, draw everything on an orange backgr. */
  NULL,              /* ReqLayer, used by Intuition. Set to NULL. */
  NULL,              /* ReqPad1, used by Intuition. Set to NULL. */
  NULL,              /* ImageBMap, no predrawn Bitmap. Set to NULL. */
                     /*            (The PREDRAWN flag was not set) */
  NULL,              /* RWindow, used by Intuition. Set to NULL. */
  NULL               /* ReqPad2, used by Intuition. Set to NULL. */
};



/* Declare a pointer to a Window structure: */
struct Window *my_window;

/* Declare and initialize your NewWindow structure: */
struct NewWindow my_new_window=
{
  0,             /* LeftEdge    x position of the window. */
  0,             /* TopEdge     y positio of the window. */
  640,           /* Width       640 pixels wide. */
  200,           /* Height      200 lines high. */
  0,             /* DetailPen   Text should be drawn with colour reg. 0 */
  1,             /* BlockPen    Blocks should be drawn with colour reg. 1 */
  CLOSEWINDOW|   /* IDCMPFlags  The window will give us a message if the */
                 /*             user has selected the Close window gad, */
  GADGETDOWN|    /*             or a gadget has been pressed on, or */
  GADGETUP|      /*             a gadge has been released. */
  REQSET|        /*             We will also recieve a message when the */
  REQCLEAR,      /*             user has activated and deactivated a req. */
  SMART_REFRESH| /* Flags       Intuition should refresh the window. */
  WINDOWCLOSE|   /*             Close Gadget. */
  WINDOWDRAG|    /*             Drag gadget. */
  WINDOWDEPTH|   /*             Depth arrange Gadgets. */
  WINDOWSIZING|  /*             Sizing Gadget. */
  ACTIVATE,      /*             The window should be Active when opened. */
  NULL,          /* FirstGadget No gadget connected to this window. */
  NULL,          /* CheckMark   Use Intuition's default CheckMark. */
  "The Fantastic Window!",      /* Title       Title of the window. */
  NULL,          /* Screen      Connected to the Workbench Screen. */
  NULL,          /* BitMap      No Custom BitMap. */
  140,           /* MinWidth    We will not allow the window to become */
  50,            /* MinHeight   smaller than 140 x 50, and not bigger */
  300,           /* MaxWidth    than 300 x 200. */
  200,           /* MaxHeight */
  WBENCHSCREEN   /* Type        Connected to the Workbench Screen. */
};

/* Note:                                                     */
/* Since we want to know when the user selects and deselects the */
/* DMRequester, we set the IDCMP flags REQSET and REQCLEAR.      */



main()
{
  /* Boolean variable used for the while loop: */
  BOOL close_me;

  /* Declare a variable in which we will store the IDCMP flag: */
  ULONG class;

  /* Declare a pointer to an IntuiMessage structure: */
  struct IntuiMessage *my_message;

  /* We use this variable to check if Intuition could enable the user */
  /* to bring up the requester whenever he/she wants: */
  BOOL result;



  /* Before we can use Intuition we need to open the Intuition Library: */
  IntuitionBase = (struct IntuitionBase *)
    OpenLibrary( "intuition.library", 0 );

  if( IntuitionBase == NULL )
    exit(); /* Could NOT open the Intuition Library! */
```

```
/* We will now try to open the window: */
my_window = (struct Window *) OpenWindow( &my_new_window );

/* Have we opened the window succesfully? */
if(my_window == NULL)
{
  /* Could NOT open the Window! */

  /* Close the Intuition Library since we have opened it: */
  CloseLibrary( IntuitionBase );

  exit();
}



/* We have opened the window, and everything seems to be OK. */



/* We will now try to set the Double-menu requester: */
result=SetDMRequest( my_window, &my_requester );

if( !result )  /* !result is the same thing as result==FALSE */
{
  /* Intuition could not set the Double-menu requester! */

  printf("Could not set the Double-menu requester!\n");
}
else
{
  /* OK */
  printf("Try to double-click on the right mouse button!\n\n");
}


close_me = FALSE;

/* Stay in the while loop until the user has selected the Close window */
/* gadget: */
while( !close_me )
{
  /* Wait until we have recieved a message: */
  Wait( 1 << my_window->UserPort->mp_SigBit );

  /* As long as we collect messages sucessfully: */
  while(my_message=(struct IntuiMessage *) GetMsg(my_window->UserPort))
  {
    /* After we have collected the message we can read it, and save any */
    /* important values which we maybe want to check later: */
    class = my_message->Class;

    /* After we have read it we reply as fast as possible: */
    /* REMEMBER! Do never try to read a message after you have replied! */
    /* Some other process has maybe changed it. */
    ReplyMsg( my_message );

    /* Check which IDCMP flag was sent: */
    switch( class )
    {
      case CLOSEWINDOW:  /* The user selected the Close window gadget! */
```

```
             close_me=TRUE;
             break;

      case GADGETDOWN:   /* The user has pressed on a gadget. */
             /* Since there exist only one "nomal" gadget, we do not */
             /* need to check which gadget was selected. */

             printf("Gadget down\n");
             break;

      case GADGETUP:     /* The user has released a gadget. */
             /* Since there exist only one "nomal" gadget, we do not */
             /* need to check which gadget was released. */

             /* Once we recieve this message, the requester will be */
             /* satisfied, and therefore deactivated. We will */
             /* therefore also recieve a REQCLEAR message. */

             printf("Gadget up\n");
             break;

      case REQSET:       /* Requester activated. */
             printf("Requester activated!\n");
             printf("You can not close the window now.\n");
             break;

      case REQCLEAR:     /* Requester deactivated. */
             printf("Requester deactivated!\n");
             printf("You can close the window now.\n\n");
             break;
    }
  }
}



/* We should always close the windows we have opened before we leave: */
CloseWindow( my_window );



/* Close the Intuition Library since we have opened it: */
CloseLibrary( IntuitionBase );

/* THE END */
}
```

Example6

Same as Example5, except that whenever the user double-clicks on the right mouse button, we will receive a REQVERIFY message, and first when we have replied, will the requester be activated. This example shows how to use the REQVERIFY flag.

```
/* Example6                                                        */
/* This program will open a normal window which is connected to the     */
/* Workbench Screen. The window will use all System Gadgets, and will   */
/* close first when the user has selected the System gadget Close       */
/* window. Whenever the user double-clicks on the right mouse button,   */
/* we will recieve a REQVERIFY message, and first when we have replied, */
/* will the requester be activated.                                     */



#include <intuition/intuition.h>



struct IntuitionBase *IntuitionBase;



/***************/
/* THE GADGET: */
/***************/

/* The coordinates for the box: */
SHORT gadget_border_points[]=
{
   0,  0, /* Start at position (0,0) */
  70,  0, /* Draw a line to the right to position (70,0) */
  70, 10, /* Draw a line down to position (70,10) */
   0, 10, /* Draw a line to the right to position (0,10) */
   0,  0  /* Finish of by drawing a line up to position (0,0) */
};

/* The Border structure: */
struct Border gadget_border=
{
  0, 0,        /* LeftEdge, TopEdge. */
  1,           /* FrontPen, colour register 1. */
  0,           /* BackPen, for the moment unused. */
  JAM1,        /* DrawMode, draw the lines with colour 1. */
  5,           /* Count, 5 pair of coordinates in the array. */
  gadget_border_points, /* XY, pointer to the array with the coord. */
  NULL,        /* NextBorder, no other Border structures are connected. */
};

/* The IntuiText structure: */
struct IntuiText gadget_text=
{
  1,         /* FrontPen, colour register 1. */
  0,         /* BackPen, colour register 0. */
  JAM1,      /* DrawMode, draw the characters with colour 1, do not */
             /* change the background. */
  4, 2,      /* LeftEdge, TopEdge. */
  NULL,      /* ITextFont, use default font. */
  "PRESS ME",/* IText, the text that will be printed. */
  NULL,      /* NextText, no other IntuiText structures are connected. */
};

struct Gadget requester_gadget=
{
  NULL,          /* NextGadget, no more gadgets in the list. */
  40,            /* LeftEdge, 40 pixels out. */
  20,            /* TopEdge, 20 lines down. */
  71,            /* Width, 71 pixels wide. */
  11,            /* Height, 11 pixels lines heigh. */
  GADGHCOMP,     /* Flags, when this gadget is highlighted, the gadget */
                 /* will be rendered in the complement colours. */
                 /* (Colour 0 (00) will be changed to colour 3 (11) */
                 /* (Colour 1 (01)           - " -           2 (10) */
                 /* (Colour 2 (10)           - " -           1 (01) */
                 /* (Colour 3 (11)           - " -           0 (00) */
  GADGIMMEDIATE| /* Activation, our program will recieve a message when */
  RELVERIFY|     /* the user has selected this gadget, and when the user */
                 /* has released it. */
  ENDGADGET,     /* When the user has selected this gadget, the */
                 /* requester is satisfied, and is deactivated. */
                 /* IMPORTANT! At least one gadget per requester */
                 /* must have the flag ENDGADGET set. If not, the */
                 /* requester would never be deactivated! */

  BOOLGADGET|    /* GadgetType, a Boolean gadget which is connected to */
  REQGADGET,     /* a requester. IMPORTANT! Every gadget which is */
                 /* connectd to a requester must have the REQGADGET flsg */
                 /* set in the GadgetType field. */
  (APTR) &gadget_border, /* GadgetRender, a pointer to our Border struc. */
  NULL,          /* SelectRender, NULL since we do not supply the gadget */
                 /* with an alternative image. (We complement the */
                 /* colours instead) */
  &gadget_text,  /* GadgetText, a pointer to our IntuiText structure. */
                 /* (See chapter 3 GRAPHICS for more information) */
  NULL,          /* MutualExclude, no mutual exclude. */
  NULL,          /* SpecialInfo, NULL since this is a Boolean gadget. */
                 /* (It is not a Proportional/String or Integer gdget) */
  0,             /* GadgetID, no id. */
  NULL           /* UserData, no user data connected to the gadget. */
};

/**********************************************************************/
/* Important notice:                                                  */
/* Remember that every gadget which is connected to a requester must  */
/* have the flag REQGADGET set in the GadgetType field. Remember also */
/* that at least one gadget per requester must have the ENDGADGET flag */
/* set in the Activation field.                                       */
/**********************************************************************/



/************************************/
/* THE BORDER AROUND THE REQUESTER: */
/************************************/

/* The coordinates for the box around the requester: */
SHORT requester_border_points[]=
{
    0,  0, /* Start at position (0,0) */
  319,  0, /* Draw a line to the right to position (319,0) */
  319, 99, /* Draw a line down to position (319,99) */
    0, 99, /* Draw a line to the right to position (319,99) */
    0,  0  /* Finish of by drawing a line up to position (0,0) */
};

/* The Border structure for the requester: */
struct Border requester_border=
{
  0, 0,        /* LeftEdge, TopEdge. */
  1,           /* FrontPen, colour register 1. */
  0,           /* BackPen, for the moment unused. */
  JAM1,        /* DrawMode, draw the lines with colour 1. */
  5,           /* Count, 5 pair of coordinates in the array. */
  requester_border_points, /* XY, pointer to the array with the coord. */
  NULL,        /* NextBorder, no other Border structures are connected. */
};



/**********************************/
/* THE TEXT INSIDE THE REQUESTER: */
/**********************************/

/* The IntuiText structure used to print some text inside the requester: */
struct IntuiText requester_text=
{
  1,         /* FrontPen, colour register 1. */
  0,         /* BackPen, unused since JAM1. */
  JAM1,      /* DrawMode, draw the characters with colour 1, do not */
             /* change the background. */
  4, 2,      /* LeftEdge, TopEdge. */
  NULL,      /* ITextFont, use default font. */
  "This is the requester!", /* IText, the text that will be printed. */
  NULL,      /* NextText, no other IntuiText structures are connected. */
};



/* Note:                                                              */
/* This is the structure for the Double-menu requester, but as you have */
/* maybe noticed, it is exactly the same as a normal requester struc.   */
/* The diffrence is that we call the function SetDMRequest() instead    */
/* of calling the function Request().                                   */

struct Requester my_requester=
{
  NULL,              /* OlderRequester, used by Intuition. */
  40, 20,            /* LeftEdge, TopEdge, 40 pixels out, 20 lines down. */
  320, 100,          /* Width, Height, 320 pixels wide, 100 lines high. */
  0, 0,              /* RelLeft, RelTop, Since POINTREL flag is not set, */
                     /* Intuition ignores these values. */
  &requester_gadget, /* ReqGadget, pointer to the first gadget. */
  &requester_border, /* ReqBorder, pointer to a Border structure. */
  &requester_text,   /* ReqText, pointer to a IntuiText structure. */
  NULL,              /* Flags, no flags set. */
  3,                 /* BackFill, draw everything on an orange backgr. */
  NULL,              /* ReqLayer, used by Intuition. Set to NULL. */
  NULL,              /* ReqPad1, used by Intuition. Set to NULL. */
  NULL,              /* ImageBMap, no predrawn Bitmap. Set to NULL. */
                     /*            (The PREDRAWN flag was not set) */
  NULL,              /* RWindow, used by Intuition. Set to NULL. */
  NULL               /* ReqPad2, used by Intuition. Set to NULL. */
};



/* Declare a pointer to a Window structure: */
struct Window *my_window;

/* Declare and initialize your NewWindow structure: */
struct NewWindow my_new_window=
{
  0,              /* LeftEdge    x position of the window. */
  0,              /* TopEdge     y positio of the window. */
  640,            /* Width       640 pixels wide. */
  200,            /* Height      200 lines high. */
  0,              /* DetailPen   Text should be drawn with colour reg. 0 */
  1,              /* BlockPen    Blocks should be drawn with colour reg. 1 */
  CLOSEWINDOW|    /* IDCMPFlags  The window will give us a message if the */
                  /*             user has selected the Close window gad, */
  GADGETDOWN|     /*             or a gadget has been pressed on, or */
  GADGETUP|       /*             a gadge has been released. */
  REQSET|         /*             We will also recieve a message when the */
  REQCLEAR|       /*             user has activated and deactivated a req. */
  REQVERIFY,      /*             When a requester is activated we will */
                  /*             recieve a message, and the requester */
                  /*             will be activated first when we have */
                  /*             replied. */
  SMART_REFRESH| /* Flags       Intuition should refresh the window. */
  WINDOWCLOSE|    /*             Close Gadget. */
  WINDOWDRAG|     /*             Drag gadget. */
  WINDOWDEPTH|    /*             Depth arrange Gadgets. */
  WINDOWSIZING|   /*             Sizing Gadget. */
  ACTIVATE,       /*             The window should be Active when opened. */
  NULL,           /* FirstGadget No gadget connected to this window. */
  NULL,           /* CheckMark   Use Intuition's default CheckMark. */
  "The Fantastic Window!",      /* Title       Title of the window. */
  NULL,           /* Screen      Connected to the Workbench Screen. */
  NULL,           /* BitMap      No Custom BitMap. */
  140,            /* MinWidth    We will not allow the window to become */
  50,             /* MinHeight   smaller than 140 x 50, and not bigger */
  300,            /* MaxWidth    than 300 x 200. */
  200,            /* MaxHeight */
  WBENCHSCREEN    /* Type        Connected to the Workbench Screen. */
};

/* Note:                                                        */
/* Since we want to know when the user selects and deselects the */
/* DMRequester, we set the IDCMP flags REQSET and REQCLEAR.      */
/* We have also set the flag REQVERIFY which enable us to finish */
/* of something before the requester is activated. Note that     */
/* everything, even the cursor, is halted while Intuition is     */
/* waiting on our reply.                                         */



main()
{
  /* Boolean variable used for the while loop: */
  BOOL close_me;

  /* Declare a variable in which we will store the IDCMP flag: */
  ULONG class;

  /* Declare a pointer to an IntuiMessage structure: */
  struct IntuiMessage *my_message;

  /* We use this variable to check if Intuition could enable the user */
  /* to bring up the requester whenever he/she wants: */
  BOOL result;
```

```
/* Before we can use Intuition we need to open the Intuition Library: */
IntuitionBase = (struct IntuitionBase *)
  OpenLibrary( "intuition.library", 0 );

if( IntuitionBase == NULL )
  exit(); /* Could NOT open the Intuition Library! */



/* We will now try to open the window: */
my_window = (struct Window *) OpenWindow( &my_new_window );

/* Have we opened the window succesfully? */
if(my_window == NULL)
{
  /* Could NOT open the Window! */

  /* Close the Intuition Library since we have opened it: */
  CloseLibrary( IntuitionBase );

  exit();
}



/* We have opened the window, and everything seems to be OK. */



/* We will now try to set the Double-menu requester: */
result=SetDMRequest( my_window, &my_requester );

if( !result )  /* !result is the same thing as result==FALSE */
{
  /* Intuition could not set the Double-menu requester! */

  printf("Could not set the Double-menu requester!\n");
}
else
{
  /* OK */
  printf("Try to double-click on the right mouse button!\n\n");
}


close_me = FALSE;

/* Stay in the while loop until the user has selected the Close window */
/* gadget: */
while( !close_me )
{
  /* Wait until we have recieved a message: */
  Wait( 1 << my_window->UserPort->mp_SigBit );

  /* As long as we collect messages sucessfully: */
  while(my_message=(struct IntuiMessage *) GetMsg(my_window->UserPort))
  {
    /* After we have collected the message we can read it, and save any */
    /* important values which we maybe want to check later: */
    class = my_message->Class;

    /* We will do a little check on the IDCMP (class) flag before we */
```

```
    /* reply: (We want to check if the REQVERIFY flag was sent) */

    if( class == REQVERIFY )
    {
      /* The user is trying to activate the requester. */

      printf("We have recieved a REQVERIFY message, and the requester ");
      printf("will be activated\nfirst when we have replied. ");
      printf("We take a little pause...\n\n");

      printf("5 seconds left\n");

      /* Wait 1 seconds: */
      Delay(50);
      printf("4 seconds left\n");

      /* Wait 1 seconds: */
      Delay(50);
      printf("3 seconds left\n");

      /* Wait 1 seconds: */
      Delay(50);
      printf("2 seconds left\n");

      /* Wait 1 seconds: */
      Delay(50);
      printf("1 second left\n");

      /* Wait 1 seconds: */
      Delay(50);
      printf("OK!\n\n");
    }



    /* After we have read it we reply as fast as possible: */
    /* REMEMBER! Do never try to read a message after you have replied! */
    /* Some other process has maybe changed it. */
    ReplyMsg( my_message );

    /* Check which IDCMP flag was sent: */
    switch( class )
    {
      case CLOSEWINDOW:  /* The user selected the Close window gadget! */
             close_me=TRUE;
             break;

      case GADGETDOWN:   /* The user has pressed on a gadget. */
             /* Since there exist only one "nomal" gadget, we do not */
             /* need to check which gadget was selected. */

             printf("Gadget down\n");
             break;

      case GADGETUP:     /* The user has released a gadget. */
             /* Since there exist only one "nomal" gadget, we do not */
             /* need to check which gadget was released. */

             /* Once we recieve this message, the requester will be */
             /* satisfied, and therefore deactivated. We will */
             /* therefore also recieve a REQCLEAR message. */
```

```
              printf("Gadget up\n");
              break;

       case REQSET:       /* Requester activated. */
              printf("Requester activated!\n");
              printf("You can not close the window now.\n");
              break;

       case REQCLEAR:     /* Requester deactivated. */
              printf("Requester deactivated!\n");
              printf("You can close the window now.\n\n");
              break;
     }
   }
  }



  /* We should always close the windows we have opened before we leave: */
  CloseWindow( my_window );



  /* Close the Intuition Library since we have opened it: */
  CloseLibrary( IntuitionBase );

  /* THE END */
}
```

Example7

This program will open a normal window which is connected to the Workbench Screen. The window will use all System Gadgets, and will close first when the user has selected the System gadget Close window. Inside the window we have activated an Application requester with three connecting gadgets. Two are Boolean gadgets ("OK and "CANCEL"), and one is a String gadget.

```
/* Example7                                                         */
/* This program will open a normal window which is connected to the     */
/* Workbench Screen. The window will use all System Gadgets, and will   */
/* close first when the user has selected the System gadget Close       */
/* window. Inside the window we have activated an Application requester */
/* with three connecting gadgets. Two are Boolean gadgets ("OK and      */
/* "CANCEL"), and one is a String gadget.                               */



#include <intuition/intuition.h>



struct IntuitionBase *IntuitionBase;



/***********************************/
/* THE STRING GADGET's STRUCTURES: */
/***********************************/

/* The coordinates for the box around the string gadget: */
SHORT string_border_points[]=
{
   -7, -4, /* Start at position (-7, -4) */
  200, -4, /* Draw a line to the right to position (200,-4) */
  200, 11, /* Draw a line down to position (200,11) */
   -7, 11, /* Draw a line to the left to position (-7,11) */
   -7, -4  /* Finish of by drawing a line up to position (-7,-4) */
};

/* The Border structure for the string gadget: */
struct Border string_border=
{
  0, 0,                 /* LeftEdge, TopEdge. */
  1,                    /* FrontPen, colour register 1. */
  0,                    /* BackPen, for the moment unused. */
  JAM1,                 /* DrawMode, draw the lines with colour 1. */
  5,                    /* Count, 5 pair of coordinates in the array. */
  string_border_points, /* XY, pointer to the array with the coordinates. */
  NULL,                 /* NextBorder, no other Border structures. */
};



/* The IntuiText structure for the string gadget: */
struct IntuiText string_text=
{
  1,         /* FrontPen, colour register 1. (white) */
  0,         /* BackPen, not used since JAM1. */
  JAM1,      /* DrawMode, draw the characters with colour 1, and do not */
             /* bother about the background. */
  -53, 0,    /* LeftEdge, TopEdge. */
  NULL,      /* ITextFont, use default font. */
  "Name:",   /* IText, the text that will be printed. */
  NULL,      /* NextText, no other IntuiText structures. */
};



UBYTE my_buffer[50]; /* 50 characters including the NULL-sign. */
```

```
UBYTE my_undo_buffer[50]; /* Must be at least as big as my_buffer. */



struct StringInfo string_info=
{
  my_buffer,       /* Buffer, pointer to a null-terminated string. */
  my_undo_buffer,  /* UndoBuffer, pointer to a null-terminated string. */
                   /* (Remember my_buffer is equal to &my_buffer[0]) */
  0,               /* BufferPos, initial position of the cursor. */
  50,              /* MaxChars, 50 characters + null-sign ('\0'). */
  0,               /* DispPos, first character in the string should be */
                   /* first character in the display. */

  /* Intuition initializes and maintaines these variables: */

  0,               /* UndoPos */
  0,               /* NumChars */
  0,               /* DispCount */
  0, 0,            /* CLeft, CTop */
  NULL,            /* LayerPtr */
  NULL,            /* LongInt */
  NULL,            /* AltKeyMap */
};


struct Gadget string_gadget=
{
  NULL,          /* NextGadget, no more gadgets in the list. */
  68,            /* LeftEdge, 68 pixels out. */
  26,            /* TopEdge, 26 lines down. */
  198,           /* Width, 198 pixels wide. */
  8,             /* Height, 8 pixels lines heigh. */
  GADGHCOMP,     /* Flags, draw the select box in the complement */
                 /* colours. Note: it actually only the cursor which */
                 /* will be drawn in the complement colours (yellow). */
                 /* If you set the flag GADGHNONE the cursor will not be */
                 /* highlighted, and the user will therefore not be able */
                 /* to see it. */
  GADGIMMEDIATE| /* Activation, our program will recieve a message when */
  RELVERIFY,     /* the user has selected this gadget, and when the user */
                 /* has released it. */
  STRGADGET|     /* GadgetType, a String gadget which is connected to */
  REQGADGET,     /* a requester. IMPORTANT! Every gadget which is */
                 /* connectd to a requester must have the REQGADGET flsg */
                 /* set in the GadgetType field. */
  (APTR) &string_border, /* GadgetRender, a pointer to our Border struc. */
  NULL,          /* SelectRender, NULL since we do not supply the gadget */
                 /* with an alternative image. */
  &string_text,  /* GadgetText, a pointer to our IntuiText structure. */
  NULL,          /* MutualExclude, no mutual exclude. */
  (APTR) &string_info, /* SpecialInfo, a pointer to a StringInfo str. */
  0,             /* GadgetID, no id. */
  NULL           /* UserData, no user data connected to the gadget. */
};



/*******************************/
/* THE OK GADGET's STRUCTURES: */
/*******************************/
```

```
/* The coordinates for the OK box: */
SHORT ok_border_points[]=
{
   0,  0, /* Start at position (0,0) */
  22,  0, /* Draw a line to the right to position (22,0) */
  22, 10, /* Draw a line down to position (22,10) */
   0, 10, /* Draw a line to the left to position (0,10) */
   0,  0  /* Finish of by drawing a line up to position (0,0) */
};

/* The Border structure: */
struct Border ok_border=
{
  0, 0,        /* LeftEdge, TopEdge. */
  1,           /* FrontPen, colour register 1. */
  0,           /* BackPen, for the moment unused. */
  JAM1,        /* DrawMode, draw the lines with colour 1. */
  5,           /* Count, 5 pair of coordinates in the array. */
  ok_border_points, /* XY, pointer to the array with the coord. */
  NULL,        /* NextBorder, no other Border structures are connected. */
};

/* The IntuiText structure: */
struct IntuiText ok_text=
{
  1,      /* FrontPen, colour register 1. */
  0,      /* BackPen, not used since JAM1. */
  JAM1,   /* DrawMode, draw the characters with colour 1, do not */
          /* change the background. */
  4, 2,   /* LeftEdge, TopEdge. */
  NULL,   /* ITextFont, use default font. */
  "OK",   /* IText, the text that will be printed. */
  NULL,   /* NextText, no other IntuiText structures are connected. */
};

struct Gadget ok_gadget=
{
  &string_gadget,/* NextGadget, linked to the string gadget. */
  14,            /* LeftEdge, 14 pixels out. */
  47,            /* TopEdge, 47 lines down. */
  23,            /* Width, 23 pixels wide. */
  11,            /* Height, 11 pixels lines heigh. */
  GADGHCOMP,     /* Flags, when this gadget is highlighted, the gadget */
                 /* will be rendered in the complement colours. */
                 /* (Colour 0 (00) will be changed to colour 3 (11) */
                 /* (Colour 1 (01)           - " -           2 (10) */
                 /* (Colour 2 (10)           - " -           1 (01) */
                 /* (Colour 3 (11)           - " -           0 (00) */
  GADGIMMEDIATE| /* Activation, our program will recieve a message when */
  RELVERIFY|     /* the user has selected this gadget, and when the user */
                 /* has released it. */
  ENDGADGET,     /* When the user has selected this gadget, the */
                 /* requester is satisfied, and is deactivated. */
                 /* IMPORTANT! At least one gadget per requester */
                 /* must have the flag ENDGADGET set. If not, the */
                 /* requester would never be deactivated! */

  BOOLGADGET|    /* GadgetType, a Boolean gadget which is connected to */
  REQGADGET,     /* a requester. IMPORTANT! Every gadget which is */
                 /* connectd to a requester must have the REQGADGET flsg */
                 /* set in the GadgetType field. */
  (APTR) &ok_border, /* GadgetRender, a pointer to our Border struc. */
  NULL,          /* SelectRender, NULL since we do not supply the gadget */
                 /* with an alternative image. (We complement the */
                 /* colours instead) */
  &ok_text,      /* GadgetText, a pointer to our IntuiText structure. */
                 /* (See chapter 3 GRAPHICS for more information) */
  NULL,          /* MutualExclude, no mutual exclude. */
  NULL,          /* SpecialInfo, NULL since this is a Boolean gadget. */
                 /* (It is not a Proportional/String or Integer gdget) */
  0,             /* GadgetID, no id. */
  NULL           /* UserData, no user data connected to the gadget. */
};



/***********************************/
/* THE CANCEL GADGET's STRUCTURES: */
/***********************************/

/* The coordinates for the CANCEL box: */
SHORT cancel_border_points[]=
{
   0,  0, /* Start at position (0,0) */
  54,  0, /* Draw a line to the right to position (54,0) */
  54, 10, /* Draw a line down to position (54,10) */
   0, 10, /* Draw a line to the left to position (0,10) */
   0,  0  /* Finish of by drawing a line up to position (0,0) */
};

/* The Border structure: */
struct Border cancel_border=
{
  0, 0,        /* LeftEdge, TopEdge. */
  1,           /* FrontPen, colour register 1. */
  0,           /* BackPen, for the moment unused. */
  JAM1,        /* DrawMode, draw the lines with colour 1. */
  5,           /* Count, 5 pair of coordinates in the array. */
  cancel_border_points, /* XY, pointer to the array with the coord. */
  NULL,        /* NextBorder, no other Border structures are connected. */
};

/* The IntuiText structure: */
struct IntuiText cancel_text=
{
  1,        /* FrontPen, colour register 1. */
  0,        /* BackPen, not used since JAM1. */
  JAM1,     /* DrawMode, draw the characters with colour 1, do not */
            /* change the background. */
  4, 2,     /* LeftEdge, TopEdge. */
  NULL,     /* ITextFont, use default font. */
  "CANCEL", /* IText, the text that will be printed. */
  NULL,     /* NextText, no other IntuiText structures are connected. */
};

struct Gadget cancel_gadget=
{
  &ok_gadget,    /* NextGadget, linked to the OK gadget. */
  214,           /* LeftEdge, 214 pixels out. */
  47,            /* TopEdge, 47 lines down. */
  55,            /* Width, 55 pixels wide. */
  11,            /* Height, 11 pixels lines heigh. */
  GADGHCOMP,     /* Flags, when this gadget is highlighted, the gadget */
```

```
                /* will be rendered in the complement colours. */
                /* (Colour 0 (00) will be changed to colour 3 (11) */
                /* (Colour 1 (01)          - " -          2 (10) */
                /* (Colour 2 (10)          - " -          1 (01) */
                /* (Colour 3 (11)          - " -          0 (00) */
  GADGIMMEDIATE| /* Activation, our program will recieve a message when */
  RELVERIFY|     /* the user has selected this gadget, and when the user */
                 /* has released it. */
  ENDGADGET,     /* When the user has selected this gadget, the */
                 /* requester is satisfied, and is deactivated. */
                 /* IMPORTANT! At least one gadget per requester */
                 /* must have the flag ENDGADGET set. If not, the */
                 /* requester would never be deactivated! */

  BOOLGADGET|    /* GadgetType, a Boolean gadget which is connected to */
  REQGADGET,     /* a requester. IMPORTANT! Every gadget which is */
                 /* connectd to a requester must have the REQGADGET flsg */
                 /* set in the GadgetType field. */
  (APTR) &cancel_border, /* GadgetRender, a pointer to our Border struc. */
  NULL,          /* SelectRender, NULL since we do not supply the gadget */
                 /* with an alternative image. (We complement the */
                 /* colours instead) */
  &cancel_text,  /* GadgetText, a pointer to our IntuiText structure. */
                 /* (See chapter 3 GRAPHICS for more information) */
  NULL,          /* MutualExclude, no mutual exclude. */
  NULL,          /* SpecialInfo, NULL since this is a Boolean gadget. */
                 /* (It is not a Proportional/String or Integer gdget) */
  0,             /* GadgetID, no id. */
  NULL           /* UserData, no user data connected to the gadget. */
};



/***********************************************************************/
/* Note:                                                               */
/* Remember that every gadget which is connected to a requester must   */
/* have the flag REQGADGET set in the GadgetType field. Remember also  */
/* that at least one gadget per requester must have the ENDGADGET flag */
/* set in the Activation field.                                        */
/* In this example we have three gadgets connected to the requester.   */
/* All of them has the REQGADGET flag set, and the OK and CANCEL gadget */
/* has also the ENDGADGET flag set.                                    */
/***********************************************************************/



/************************************/
/* THE BORDER AROUND THE REQUESTER: */
/************************************/

/* The coordinates for the box around the requester: */
SHORT requester_border_points[]=
{
    0,  0, /* Start at position (0,0) */
  282,  0, /* Draw a line to the right. */
  282, 64, /* Draw a line down. */
    0, 64, /* Draw a line to the left. */
    0,  0  /* Finish of by drawing a line up to position (0,0) */
};

/* The Border structure for the requester: */
struct Border requester_border=
{
  0, 0,        /* LeftEdge, TopEdge. */
  1,           /* FrontPen, colour register 1. */
  0,           /* BackPen, for the moment unused. */
  JAM1,        /* DrawMode, draw the lines with colour 1. */
  5,           /* Count, 5 pair of coordinates in the array. */
  requester_border_points, /* XY, pointer to the array with the coord. */
  NULL,        /* NextBorder, no other Border structures are connected. */
};



/**********************************/
/* THE TEXT INSIDE THE REQUESTER: */
/**********************************/

/* The IntuiText structure used to print some text inside the requester: */
struct IntuiText requester_text=
{
  1,         /* FrontPen, colour register 1. */
  0,         /* BackPen, unused since JAM1. */
  JAM1,      /* DrawMode, draw the characters with colour 1, do not */
             /* change the background. */
  14, 8,     /* LeftEdge, TopEdge. */
  NULL,      /* ITextFont, use default font. */
  "Please enter your name:", /* IText, the text that will be printed. */
  NULL,      /* NextText, no other IntuiText structures are connected. */
};



/****************************/
/* THE REQUESTER STRUCTURE: */
/****************************/

struct Requester my_requester=
{
  NULL,              /* OlderRequester, used by Intuition. */
  40, 20,            /* LeftEdge, TopEdge, 40 pixels out, 20 lines down. */
  283, 65,           /* Width, Height, 283 pixels wide, 65 lines high. */
  0, 0,              /* RelLeft, RelTop, Since POINTREL flag is not set, */
                     /* Intuition ignores these values. */
  &cancel_gadget,    /* ReqGadget, pointer to the first gadget. */
  &requester_border, /* ReqBorder, pointer to a Border structure. */
  &requester_text,   /* ReqText, pointer to a IntuiText structure. */
  NULL,              /* Flags, no flags set. */
  2,                 /* BackFill, draw everything on a black background. */
  NULL,              /* ReqLayer, used by Intuition. Set to NULL. */
  NULL,              /* ReqPad1, used by Intuition. Set to NULL. */
  NULL,              /* ImageBMap, no predrawn Bitmap. Set to NULL. */
                     /*           (The PREDRAWN flag was not set) */
  NULL,              /* RWindow, used by Intuition. Set to NULL. */
  NULL               /* ReqPad2, used by Intuition. Set to NULL. */
};



/* Declare a pointer to a Window structure: */
struct Window *my_window;

/* Declare and initialize your NewWindow structure: */
struct NewWindow my_new_window=
{
  0,             /* LeftEdge    x position of the window. */
  0,             /* TopEdge     y positio of the window. */
  640,           /* Width       640 pixels wide. */
  200,           /* Height      200 lines high. */
  0,             /* DetailPen   Text should be drawn with colour reg. 0 */
  1,             /* BlockPen    Blocks should be drawn with colour reg. 1 */
  CLOSEWINDOW|   /* IDCMPFlags  The window will give us a message if the */
                 /*             user has selected the Close window gad, */
  GADGETDOWN|    /*             or a gadget has been pressed on, or */
  GADGETUP|      /*             a gadge has been released. */
  REQSET|        /*             Send a message also if a requester has */
  REQCLEAR,      /*             been activated or deactivated. */
  SMART_REFRESH| /* Flags       Intuition should refresh the window. */
  WINDOWCLOSE|   /*             Close Gadget. */
  WINDOWDRAG|    /*             Drag gadget. */
  WINDOWDEPTH|   /*             Depth arrange Gadgets. */
  WINDOWSIZING|  /*             Sizing Gadget. */
  ACTIVATE,      /*             The window should be Active when opened. */
  NULL,          /* FirstGadget No gadget connected to this window. */
  NULL,          /* CheckMark   Use Intuition's default CheckMark. */
  "The Window",  /* Title       Title of the window. */
  NULL,          /* Screen      Connected to the Workbench Screen. */
  NULL,          /* BitMap      No Custom BitMap. */
  140,           /* MinWidth    We will not allow the window to become */
  50,            /* MinHeight   smaller than 140 x 50, and not bigger */
  300,           /* MaxWidth    than 300 x 200. */
  200,           /* MaxHeight */
  WBENCHSCREEN   /* Type        Connected to the Workbench Screen. */
};



main()
{
  /* Boolean variable used for the while loop: */
  BOOL close_me;

  /* Declare a variable in which we will store the IDCMP flag: */
  ULONG class;

  /* Declare a variable in which we will store the address of the */
  /* gadget which sent the message: */
  APTR address;

  /* Declare a pointer to an IntuiMessage structure: */
  struct IntuiMessage *my_message;

  /* We use this variable to check if the requester has ben activated */
  /* or not: */
  BOOL result;


  /* Before we can use Intuition we need to open the Intuition Library: */
  IntuitionBase = (struct IntuitionBase *)
    OpenLibrary( "intuition.library", 0 );

  if( IntuitionBase == NULL )
    exit(); /* Could NOT open the Intuition Library! */
```

```
/* We will now try to open the window: */
my_window = (struct Window *) OpenWindow( &my_new_window );

/* Have we opened the window succesfully? */
if(my_window == NULL)
{
  /* Could NOT open the Window! */

  /* Close the Intuition Library since we have opened it: */
  CloseLibrary( IntuitionBase );

  exit();
}



/* We have opened the window, and everything seems to be OK. */



/* We will now try to activate the requester: */
result=Request( &my_requester, my_window );

if( !result )  /* !result is the same thing as result==FALSE */
{
  /* Intuition could not activate the requester! */
  /* In this case we do not need to quit since it does not matter if */
  /* the requester was activated or not. I just wanted to show how */
  /* you can check if you have opened or not the requester. */

  printf("Could not activate the requester!\n");
}
else
{
  /* Intuition could open the requester! */
  printf("Try to close the window!\n");
}



close_me = FALSE;

/* Stay in the while loop until the user has selected the Close window */
/* gadget. However, in this example the user first need to deactivate */
/* the requester before he can select the Close window gadget: */
while( !close_me )
{
  /* Wait until we have recieved a message: */
  Wait( 1 << my_window->UserPort->mp_SigBit );

  /* As long as we collect messages sucessfully: */
  while(my_message=(struct IntuiMessage *) GetMsg(my_window->UserPort))
  {
    /* After we have collected the message we can read it, and save any */
    /* important values which we maybe want to check later: */

    /* Store the IDCMP flag: */
    class = my_message->Class;

    /* Store the address: */
    address = my_message->IAddress;
    /* After we have read it we reply as fast as possible: */
    /* REMEMBER! Do never try to read a message after you have replied! */
    /* Some other process has maybe changed it. */
    ReplyMsg( my_message );

    /* Check which IDCMP flag was sent: */
    switch( class )
    {
      case CLOSEWINDOW:  /* The user selected the Close window gadget! */
             close_me=TRUE;
             break;

      case GADGETDOWN:   /* The user has pressed on a gadget. */

             if( address == (APTR) &ok_gadget )
               printf("The user pressed on the OK gadget!\n");

             if( address == (APTR) &cancel_gadget )
               printf("The user pressed on the CANCEL gadget!\n");

             if( address == (APTR) &string_gadget )
               printf("The user selected the string gadget!\n");

             break;

      case GADGETUP:     /* The user has released a gadget. */

             if( address == (APTR) &ok_gadget )
               printf("The user released the OK gadget!\n");

             if( address == (APTR) &cancel_gadget )
               printf("The user released the CANCEL gadget!\n");

             if( address == (APTR) &string_gadget )
             {
               printf("The user released the string gadget!\n");

               /* Print out the string: */
               printf("Name: %s\n\n", my_buffer);
             }

             break;

      case REQSET:       /* Requester activated. */
            printf("Requester activated!\n");
            break;

      case REQCLEAR:     /* Requester deactivated. */
            printf("Requester deactivated!\n");
            printf("You can now close the window.\n");
            break;
    }
  }
}



/* Print out the string: */
printf("Name: %s\n\n", my_buffer);
```

```
  /* We should always close the windows we have opened before we leave: */
  CloseWindow( my_window );



  /* Close the Intuition Library since we have opened it: */
  CloseLibrary( IntuitionBase );

  /* THE END */
}
```

Example8

Same as Example7, except that it is an Integer gadget.

```
/* Example8                                                     */
/* This program will open a normal window which is connected to the     */
/* Workbench Screen. The window will use all System Gadgets, and will   */
/* close first when the user has selected the System gadget Close       */
/* window. Inside the window we have activated an Application requester */
/* with three connecting gadgets. Two are Boolean gadgets ("OK and      */
/* "CANCEL"), and one is an Integer gadget.                             */



#include <intuition/intuition.h>



struct IntuitionBase *IntuitionBase;



/***********************************/
/* THE INTEGER GADGET's STRUCTURES: */
/***********************************/

/* The coordinates for the box around the integer gadget: */
SHORT integer_border_points[]=
{
   -7, -4, /* Start at position (-7, -4) */
  200, -4, /* Draw a line to the right to position (200,-4) */
  200, 11, /* Draw a line down to position (200,11) */
   -7, 11, /* Draw a line to the left to position (-7,11) */
   -7, -4  /* Finish of by drawing a line up to position (-7,-4) */
};

/* The Border structure for the integer gadget: */
struct Border integer_border=
{
  0, 0,                  /* LeftEdge, TopEdge. */
  1,                     /* FrontPen, colour register 1. */
  0,                     /* BackPen, for the moment unused. */
  JAM1,                  /* DrawMode, draw the lines with colour 1. */
  5,                     /* Count, 5 pair of coordinates in the array. */
  integer_border_points,/* XY, pointer to the array with the coordinates. */
  NULL,                  /* NextBorder, no other Border structures. */
};



/* The IntuiText structure for the integer gadget: */
struct IntuiText integer_text=
{
  1,          /* FrontPen, colour register 1. (white) */
  0,          /* BackPen, not used since JAM1. */
  JAM1,       /* DrawMode, draw the characters with colour 1, and do not */
              /* bother about the background. */
  -53, 0,     /* LeftEdge, TopEdge. */
  NULL,       /* ITextFont, use default font. */
  "Age: ",    /* IText, the text that will be printed. */
  NULL,       /* NextText, no other IntuiText structures. */
};



UBYTE my_buffer[25]; /* 25 characters including the NULL-sign. */
```

```
UBYTE my_undo_buffer[25]; /* Must be at least as big as my_buffer. */



struct StringInfo integer_info=
{
  my_buffer,        /* Buffer, pointer to a null-terminated string. */
  my_undo_buffer,   /* UndoBuffer, pointer to a null-terminated string. */
                    /* (Remember my_buffer is equal to &my_buffer[0]) */
  0,                /* BufferPos, initial position of the cursor. */
  25,               /* MaxChars, 25 characters inc. null-sign ('\0'). */
  0,                /* DispPos, first character in the string should be */
                    /* first character in the display. */

  /* Intuition initializes and maintaines these variables: */

  0,                /* UndoPos */
  0,                /* NumChars */
  0,                /* DispCount */
  0, 0,             /* CLeft, CTop */
  NULL,             /* LayerPtr */
  NULL,             /* LongInt */
  NULL,             /* AltKeyMap */
};


struct Gadget integer_gadget=
{
  NULL,           /* NextGadget, no more gadgets in the list. */
  68,             /* LeftEdge, 68 pixels out. */
  26,             /* TopEdge, 26 lines down. */
  198,            /* Width, 198 pixels wide. */
  8,              /* Height, 8 pixels lines heigh. */
  GADGHCOMP,      /* Flags, draw the select box in the complement */
                  /* colours. Note: it actually only the cursor which */
                  /* will be drawn in the complement colours (yellow). */
                  /* If you set the flag GADGHNONE the cursor will not be */
                  /* highlighted, and the user will therefore not be able */
                  /* to see it. */
  GADGIMMEDIATE|  /* Activation, our program will recieve a message when */
  RELVERIFY|      /* the user has selected this gadget, and when the user */
                  /* has released it. */
  LONGINT,        /* An Integer gadget. */
  STRGADGET|      /* GadgetType, a String gadget which is connected to */
  REQGADGET,      /* a requester. IMPORTANT! Every gadget which is */
                  /* connectd to a requester must have the REQGADGET flsg */
                  /* set in the GadgetType field. */
  (APTR) &integer_border, /* GadgetRender, a pointer to our Border struc. */
  NULL,           /* SelectRender, NULL since we do not supply the gadget */
                  /* with an alternative image. */
  &integer_text,  /* GadgetText, a pointer to our IntuiText structure. */
  NULL,           /* MutualExclude, no mutual exclude. */
  (APTR) &integer_info, /* SpecialInfo, a pointer to a StringInfo str. */
  0,              /* GadgetID, no id. */
  NULL            /* UserData, no user data connected to the gadget. */
};



/******************************/
/* THE OK GADGET's STRUCTURES: */
/******************************/
```

```
/* The coordinates for the OK box: */
SHORT ok_border_points[]=
{
   0,  0, /* Start at position (0,0) */
  22,  0, /* Draw a line to the right to position (22,0) */
  22, 10, /* Draw a line down to position (22,10) */
   0, 10, /* Draw a line to the left to position (0,10) */
   0,  0  /* Finish of by drawing a line up to position (0,0) */
};

/* The Border structure: */
struct Border ok_border=
{
  0, 0,        /* LeftEdge, TopEdge. */
  1,           /* FrontPen, colour register 1. */
  0,           /* BackPen, for the moment unused. */
  JAM1,        /* DrawMode, draw the lines with colour 1. */
  5,           /* Count, 5 pair of coordinates in the array. */
  ok_border_points, /* XY, pointer to the array with the coord. */
  NULL,        /* NextBorder, no other Border structures are connected. */
};

/* The IntuiText structure: */
struct IntuiText ok_text=
{
  1,      /* FrontPen, colour register 1. */
  0,      /* BackPen, not used since JAM1. */
  JAM1,   /* DrawMode, draw the characters with colour 1, do not */
          /* change the background. */
  4, 2,   /* LeftEdge, TopEdge. */
  NULL,   /* ITextFont, use default font. */
  "OK",   /* IText, the text that will be printed. */
  NULL,   /* NextText, no other IntuiText structures are connected. */
};

struct Gadget ok_gadget=
{
  &integer_gadget, /* NextGadget, linked to the Integer gadget. */
  14,              /* LeftEdge, 14 pixels out. */
  47,              /* TopEdge, 47 lines down. */
  23,              /* Width, 23 pixels wide. */
  11,              /* Height, 11 pixels lines heigh. */
  GADGHCOMP,       /* Flags, when this gadget is highlighted, the gadget */
                   /* will be rendered in the complement colours. */
                   /* (Colour 0 (00) will be changed to colour 3 (11) */
                   /* (Colour 1 (01)           - " -           2 (10) */
                   /* (Colour 2 (10)           - " -           1 (01) */
                   /* (Colour 3 (11)           - " -           0 (00) */
  GADGIMMEDIATE|   /* Activation, our program will recieve a message when */
  RELVERIFY|       /* the user has selected this gadget, and when the user */
                   /* has released it. */
  ENDGADGET,       /* When the user has selected this gadget, the */
                   /* requester is satisfied, and is deactivated. */
                   /* IMPORTANT! At least one gadget per requester */
                   /* must have the flag ENDGADGET set. If not, the */
                   /* requester would never be deactivated! */

  BOOLGADGET|      /* GadgetType, a Boolean gadget which is connected to */
  REQGADGET,       /* a requester. IMPORTANT! Every gadget which is */
                   /* connectd to a requester must have the REQGADGET flsg */
                   /* set in the GadgetType field. */
```

```
  (APTR) &ok_border, /* GadgetRender, a pointer to our Border struc. */
  NULL,          /* SelectRender, NULL since we do not supply the gadget */
                 /* with an alternative image. (We complement the */
                 /* colours instead) */
  &ok_text,      /* GadgetText, a pointer to our IntuiText structure. */
                 /* (See chapter 3 GRAPHICS for more information) */
  NULL,          /* MutualExclude, no mutual exclude. */
  NULL,          /* SpecialInfo, NULL since this is a Boolean gadget. */
                 /* (It is not a Proportional/String or Integer gdget) */
  0,             /* GadgetID, no id. */
  NULL           /* UserData, no user data connected to the gadget. */
};



/**********************************/
/* THE CANCEL GADGET's STRUCTURES: */
/**********************************/

/* The coordinates for the CANCEL box: */
SHORT cancel_border_points[]=
{
   0,  0, /* Start at position (0,0) */
  54,  0, /* Draw a line to the right to position (54,0) */
  54, 10, /* Draw a line down to position (54,10) */
   0, 10, /* Draw a line to the left to position (0,10) */
   0,  0  /* Finish of by drawing a line up to position (0,0) */
};

/* The Border structure: */
struct Border cancel_border=
{
  0, 0,        /* LeftEdge, TopEdge. */
  1,           /* FrontPen, colour register 1. */
  0,           /* BackPen, for the moment unused. */
  JAM1,        /* DrawMode, draw the lines with colour 1. */
  5,           /* Count, 5 pair of coordinates in the array. */
  cancel_border_points, /* XY, pointer to the array with the coord. */
  NULL,        /* NextBorder, no other Border structures are connected. */
};

/* The IntuiText structure: */
struct IntuiText cancel_text=
{
  1,        /* FrontPen, colour register 1. */
  0,        /* BackPen, not used since JAM1. */
  JAM1,     /* DrawMode, draw the characters with colour 1, do not */
            /* change the background. */
  4, 2,     /* LeftEdge, TopEdge. */
  NULL,     /* ITextFont, use default font. */
  "CANCEL", /* IText, the text that will be printed. */
  NULL,     /* NextText, no other IntuiText structures are connected. */
};

struct Gadget cancel_gadget=
{
  &ok_gadget,    /* NextGadget, linked to the OK gadget. */
  214,           /* LeftEdge, 214 pixels out. */
  47,            /* TopEdge, 47 lines down. */
  55,            /* Width, 55 pixels wide. */
  11,            /* Height, 11 pixels lines heigh. */
  GADGHCOMP,     /* Flags, when this gadget is highlighted, the gadget */
                 /* will be rendered in the complement colours. */
                 /* (Colour 0 (00) will be changed to colour 3 (11) */
                 /* (Colour 1 (01)           - " -           2 (10) */
                 /* (Colour 2 (10)           - " -           1 (01) */
                 /* (Colour 3 (11)           - " -           0 (00) */
  GADGIMMEDIATE| /* Activation, our program will recieve a message when */
  RELVERIFY|     /* the user has selected this gadget, and when the user */
                 /* has released it. */
  ENDGADGET,     /* When the user has selected this gadget, the */
                 /* requester is satisfied, and is deactivated. */
                 /* IMPORTANT! At least one gadget per requester */
                 /* must have the flag ENDGADGET set. If not, the */
                 /* requester would never be deactivated! */

  BOOLGADGET|    /* GadgetType, a Boolean gadget which is connected to */
  REQGADGET,     /* a requester. IMPORTANT! Every gadget which is */
                 /* connectd to a requester must have the REQGADGET flsg */
                 /* set in the GadgetType field. */
  (APTR) &cancel_border, /* GadgetRender, a pointer to our Border struc. */
  NULL,          /* SelectRender, NULL since we do not supply the gadget */
                 /* with an alternative image. (We complement the */
                 /* colours instead) */
  &cancel_text,  /* GadgetText, a pointer to our IntuiText structure. */
                 /* (See chapter 3 GRAPHICS for more information) */
  NULL,          /* MutualExclude, no mutual exclude. */
  NULL,          /* SpecialInfo, NULL since this is a Boolean gadget. */
                 /* (It is not a Proportional/String or Integer gdget) */
  0,             /* GadgetID, no id. */
  NULL           /* UserData, no user data connected to the gadget. */
};



/***********************************************************************/
/* Note:                                                               */
/* Remember that every gadget which is connected to a requester must   */
/* have the flag REQGADGET set in the GadgetType field. Remember also  */
/* that at least one gadget per requester must have the ENDGADGET flag */
/* set in the Activation field.                                        */
/* In this example we have three gadgets connected to the requester.   */
/* All of them has the REQGADGET flag set, and the OK and CANCEL gadget */
/* has also the ENDGADGET flag set.                                    */
/***********************************************************************/



/***********************************/
/* THE BORDER AROUND THE REQUESTER: */
/***********************************/

/* The coordinates for the box around the requester: */
SHORT requester_border_points[]=
{
    0,  0, /* Start at position (0,0) */
  282,  0, /* Draw a line to the right. */
  282, 64, /* Draw a line down. */
    0, 64, /* Draw a line to the left. */
    0,  0  /* Finish of by drawing a line up to position (0,0) */
};

/* The Border structure for the requester: */
struct Border requester_border=
{
  0, 0,        /* LeftEdge, TopEdge. */
  1,           /* FrontPen, colour register 1. */
  0,           /* BackPen, for the moment unused. */
  JAM1,        /* DrawMode, draw the lines with colour 1. */
  5,           /* Count, 5 pair of coordinates in the array. */
  requester_border_points, /* XY, pointer to the array with the coord. */
  NULL,        /* NextBorder, no other Border structures are connected. */
};



/**********************************/
/* THE TEXT INSIDE THE REQUESTER: */
/**********************************/

/* The IntuiText structure used to print some text inside the requester: */
struct IntuiText requester_text=
{
  1,         /* FrontPen, colour register 1. */
  0,         /* BackPen, unused since JAM1. */
  JAM1,      /* DrawMode, draw the characters with colour 1, do not */
             /* change the background. */
  14, 8,     /* LeftEdge, TopEdge. */
  NULL,      /* ITextFont, use default font. */
  "Please enter your age:", /* IText, the text that will be printed. */
  NULL,      /* NextText, no other IntuiText structures are connected. */
};



/***************************/
/* THE REQUESTER STRUCTURE: */
/***************************/

struct Requester my_requester=
{
  NULL,              /* OlderRequester, used by Intuition. */
  40, 20,            /* LeftEdge, TopEdge, 40 pixels out, 20 lines down. */
  283, 65,           /* Width, Height, 283 pixels wide, 65 lines high. */
  0, 0,              /* RelLeft, RelTop, Since POINTREL flag is not set, */
                     /* Intuition ignores these values. */
  &cancel_gadget,    /* ReqGadget, pointer to the first gadget. */
  &requester_border, /* ReqBorder, pointer to a Border structure. */
  &requester_text,   /* ReqText, pointer to a IntuiText structure. */
  NULL,              /* Flags, no flags set. */
  2,                 /* BackFill, draw everything on a black background. */
  NULL,              /* ReqLayer, used by Intuition. Set to NULL. */
  NULL,              /* ReqPad1, used by Intuition. Set to NULL. */
  NULL,              /* ImageBMap, no predrawn Bitmap. Set to NULL. */
                     /*           (The PREDRAWN flag was not set) */
  NULL,              /* RWindow, used by Intuition. Set to NULL. */
  NULL               /* ReqPad2, used by Intuition. Set to NULL. */
};



/* Declare a pointer to a Window structure: */
struct Window *my_window;

/* Declare and initialize your NewWindow structure: */
```

```
struct NewWindow my_new_window=
{
  0,             /* LeftEdge    x position of the window. */
  0,             /* TopEdge     y positio of the window. */
  640,           /* Width       640 pixels wide. */
  200,           /* Height      200 lines high. */
  0,             /* DetailPen   Text should be drawn with colour reg. 0 */
  1,             /* BlockPen    Blocks should be drawn with colour reg. 1 */
  CLOSEWINDOW|   /* IDCMPFlags  The window will give us a message if the */
                 /*             user has selected the Close window gad, */
  GADGETDOWN|    /*             or a gadget has been pressed on, or */
  GADGETUP|      /*             a gadge has been released. */
  REQSET|        /*             Send a message also if a requester has */
  REQCLEAR,      /*             been activated or deactivated. */
  SMART_REFRESH| /* Flags       Intuition should refresh the window. */
  WINDOWCLOSE|   /*             Close Gadget. */
  WINDOWDRAG|    /*             Drag gadget. */
  WINDOWDEPTH|   /*             Depth arrange Gadgets. */
  WINDOWSIZING|  /*             Sizing Gadget. */
  ACTIVATE,      /*             The window should be Active when opened. */
  NULL,          /* FirstGadget No gadget connected to this window. */
  NULL,          /* CheckMark   Use Intuition's default CheckMark. */
  "The Window",  /* Title       Title of the window. */
  NULL,          /* Screen      Connected to the Workbench Screen. */
  NULL,          /* BitMap      No Custom BitMap. */
  140,           /* MinWidth    We will not allow the window to become */
  50,            /* MinHeight   smaller than 140 x 50, and not bigger */
  300,           /* MaxWidth    than 300 x 200. */
  200,           /* MaxHeight */
  WBENCHSCREEN   /* Type        Connected to the Workbench Screen. */
};



main()
{
  /* Boolean variable used for the while loop: */
  BOOL close_me;

  /* Declare a variable in which we will store the IDCMP flag: */
  ULONG class;

  /* Declare a variable in which we will store the address of the */
  /* gadget which sent the message: */
  APTR address;

  /* Declare a pointer to an IntuiMessage structure: */
  struct IntuiMessage *my_message;

  /* We use this variable to check if the requester has ben activated */
  /* or not: */
  BOOL result;



  /* Put an integer value in the string: */
  /* This is very important! */
  strcpy( my_buffer, "0" );



  /* Before we can use Intuition we need to open the Intuition Library: */
```

```
  IntuitionBase = (struct IntuitionBase *)
    OpenLibrary( "intuition.library", 0 );

  if( IntuitionBase == NULL )
    exit(); /* Could NOT open the Intuition Library! */



  /* We will now try to open the window: */
  my_window = (struct Window *) OpenWindow( &my_new_window );

  /* Have we opened the window succesfully? */
  if(my_window == NULL)
  {
    /* Could NOT open the Window! */

    /* Close the Intuition Library since we have opened it: */
    CloseLibrary( IntuitionBase );

    exit();
  }



  /* We have opened the window, and everything seems to be OK. */



  /* We will now try to activate the requester: */
  result=Request( &my_requester, my_window );

  if( !result )  /* !result is the same thing as result==FALSE */
  {
    /* Intuition could not activate the requester! */
    /* In this case we do not need to quit since it does not matter if */
    /* the requester was activated or not. I just wanted to show how */
    /* you can check if you have opened or not the requester. */

    printf("Could not activate the requester!\n");
  }
  else
  {
    /* Intuition could open the requester! */
    printf("Try to close the window!\n");
  }



  close_me = FALSE;

  /* Stay in the while loop until the user has selected the Close window */
  /* gadget. However, in this example the user first need to deactivate */
  /* the requester before he can select the Close window gadget: */
  while( !close_me )
  {
    /* Wait until we have recieved a message: */
    Wait( 1 << my_window->UserPort->mp_SigBit );

    /* As long as we collect messages sucessfully: */
    while(my_message=(struct IntuiMessage *) GetMsg(my_window->UserPort))
    {
      /* After we have collected the message we can read it, and save any */
    /* important values which we maybe want to check later: */

    /* Store the IDCMP flag: */
    class = my_message->Class;

    /* Store the address: */
    address = my_message->IAddress;

    /* After we have read it we reply as fast as possible: */
    /* REMEMBER! Do never try to read a message after you have replied! */
    /* Some other process has maybe changed it. */
    ReplyMsg( my_message );

    /* Check which IDCMP flag was sent: */
    switch( class )
    {
      case CLOSEWINDOW:  /* The user selected the Close window gadget! */
             close_me=TRUE;
             break;

      case GADGETDOWN:   /* The user has pressed on a gadget. */

             if( address == (APTR) &ok_gadget )
               printf("The user pressed on the OK gadget!\n");

             if( address == (APTR) &cancel_gadget )
               printf("The user pressed on the CANCEL gadget!\n");

             if( address == (APTR) &integer_gadget )
               printf("The user selected the Integer gadget!\n");

             break;

      case GADGETUP:     /* The user has released a gadget. */

             if( address == (APTR) &ok_gadget )
               printf("The user released the OK gadget!\n");

             if( address == (APTR) &cancel_gadget )
               printf("The user released the CANCEL gadget!\n");

             if( address == (APTR) &integer_gadget )
             {
               printf("The user released the Integer gadget!\n");

               /* Print out the integer value: */
               printf("Nr: %d\n\n", integer_info.LongInt);
             }
             break;

      case REQSET:       /* Requester activated. */
            printf("Requester activated!\n");
            break;

      case REQCLEAR:     /* Requester deactivated. */
            printf("Requester deactivated!\n");
            printf("You can now close the window.\n");
            break;
    }
  }
}
```

```
  /* Print out the integer value: */
  printf("Nr: %d\n\n", integer_info.LongInt);



  /* We should always close the windows we have opened before we leave: */
  CloseWindow( my_window );



  /* Close the Intuition Library since we have opened it: */
  CloseLibrary( IntuitionBase );

  /* THE END */
}
```

Example9
Same as Example8, except that it is a Proportional gadget.

```
/* Example9                                                         */
/* This program will open a normal window which is connected to the     */
/* Workbench Screen. The window will use all System Gadgets, and will   */
/* close first when the user has selected the System gadget Close       */
/* window. Inside the window we have activated an Application requester */
/* with three connecting gadgets. Two are Boolean gadgets ("OK and      */
/* "CANCEL"), and one is a Proportional gadget.                             */



#include <intuition/intuition.h>



struct IntuitionBase *IntuitionBase;



/*****************************************/
/* THE PROPORTIONAL GADGET's STRUCTURES: */
/*****************************************/

/* The IntuiText structure for the proportional gadget: */
struct IntuiText prop_text=
{
  1,          /* FrontPen, colour register 1. */
  0,          /* BackPen, colour register 0. */
  JAM1,       /* DrawMode, draw the characters with colour 1, do not */
              /* change the background. */
  -65, 2,     /* LeftEdge, TopEdge. */
  NULL,       /* ITextFont, use default font. */
  "Colour:",  /* IText, the text that will be printed. */
  NULL,       /* NextText, no other IntuiText structures. */
};


/* We need to declare an Image structure for the knob, but since */
/* Intuition will take care of the size etc of the knob, we do not need */
/* to initialize the Image structure: */
struct Image prop_image;


struct PropInfo prop_info=
{
  FREEHORIZ|      /* Flags, the knob should be moved horizontally, and */
  AUTOKNOB,       /* Intuition should take care of the knob image. */
  0,              /* HorizPot, start position of the knob. */
  0,              /* VertPot, 0 since we will not move the knob hor. */
  MAXBODY * 1/16, /* HorizBody, 16 steps. */
  0,              /* VertBody, 0 since we will not move the knob hor. */

  /* These variables are initialized and maintained by Intuition: */

  0,              /* CWidth */
  0,              /* CHeight */
  0, 0,           /* HPotRes, VPotRes */
  0,              /* LeftBorder */
  0               /* TopBorder */
};


struct Gadget prop_gadget=
{
  NULL,              /* NextGadget, no more gadgets in the list. */
  80,                /* LeftEdge, 80 pixels out. */
  30,                /* TopEdge, 30 lines down. */
  189,               /* Width, 189 pixels wide. */
  12,                /* Height, 12 pixels lines heigh. */
  GADGHCOMP,         /* Flags, complement the colours. */
  GADGIMMEDIATE|     /* Activation, our program will recieve a message */
  RELVERIFY,         /* when the user has selected this gadget, and when */
                     /* the user has released it. */
  PROPGADGET,        /* GadgetType, a Proportional gadget. */
  (APTR) &prop_image,/* GadgetRender, a pointer to our Image structure. */
                     /* (Intuition will take care of the knob image) */
                     /* (See chapter 3 GRAPHICS for more information) */
  NULL,              /* SelectRender, NULL since we do not supply the */
                     /* gadget with an alternative image. */
  &prop_text,        /* GadgetText, colour. */
  NULL,              /* MutualExclude, no mutual exclude. */
  (APTR) &prop_info, /* SpecialInfo, pointer to a PropInfo structure. */
  0,                 /* GadgetID, no id. */
  NULL               /* UserData, no user data connected to the gadget. */
};



/*******************************/
/* THE OK GADGET's STRUCTURES: */
/*******************************/

/* The coordinates for the OK box: */
SHORT ok_border_points[]=
{
   0,  0, /* Start at position (0,0) */
  22,  0, /* Draw a line to the right to position (22,0) */
  22, 10, /* Draw a line down to position (22,10) */
   0, 10, /* Draw a line to the left to position (0,10) */
   0,  0  /* Finish of by drawing a line up to position (0,0) */
};

/* The Border structure: */
struct Border ok_border=
{
  0, 0,        /* LeftEdge, TopEdge. */
  1,           /* FrontPen, colour register 1. */
  0,           /* BackPen, for the moment unused. */
  JAM1,        /* DrawMode, draw the lines with colour 1. */
  5,           /* Count, 5 pair of coordinates in the array. */
  ok_border_points, /* XY, pointer to the array with the coord. */
  NULL,        /* NextBorder, no other Border structures are connected. */
};

/* The IntuiText structure: */
struct IntuiText ok_text=
{
  1,      /* FrontPen, colour register 1. */
  0,      /* BackPen, not used since JAM1. */
  JAM1,   /* DrawMode, draw the characters with colour 1, do not */
          /* change the background. */
  4, 2,   /* LeftEdge, TopEdge. */
  NULL,   /* ITextFont, use default font. */
  "OK",   /* IText, the text that will be printed. */
  NULL,   /* NextText, no other IntuiText structures are connected. */
```

```
};

struct Gadget ok_gadget=
{
  &prop_gadget,  /* NextGadget, linked to the Proportional gadget. */
  14,            /* LeftEdge, 14 pixels out. */
  47,            /* TopEdge, 47 lines down. */
  23,            /* Width, 23 pixels wide. */
  11,            /* Height, 11 pixels lines heigh. */
  GADGHCOMP,     /* Flags, when this gadget is highlighted, the gadget */
                 /* will be rendered in the complement colours. */
                 /* (Colour 0 (00) will be changed to colour 3 (11) */
                 /* (Colour 1 (01)           - " -           2 (10) */
                 /* (Colour 2 (10)           - " -           1 (01) */
                 /* (Colour 3 (11)           - " -           0 (00) */
  GADGIMMEDIATE| /* Activation, our program will recieve a message when */
  RELVERIFY|     /* the user has selected this gadget, and when the user */
                 /* has released it. */
  ENDGADGET,     /* When the user has selected this gadget, the */
                 /* requester is satisfied, and is deactivated. */
                 /* IMPORTANT! At least one gadget per requester */
                 /* must have the flag ENDGADGET set. If not, the */
                 /* requester would never be deactivated! */

  BOOLGADGET|    /* GadgetType, a Boolean gadget which is connected to */
  REQGADGET,     /* a requester. IMPORTANT! Every gadget which is */
                 /* connectd to a requester must have the REQGADGET flsg */
                 /* set in the GadgetType field. */
 (APTR) &ok_border, /* GadgetRender, a pointer to our Border struc. */
  NULL,          /* SelectRender, NULL since we do not supply the gadget */
                 /* with an alternative image. (We complement the */
                 /* colours instead) */
  &ok_text,      /* GadgetText, a pointer to our IntuiText structure. */
                 /* (See chapter 3 GRAPHICS for more information) */
  NULL,          /* MutualExclude, no mutual exclude. */
  NULL,          /* SpecialInfo, NULL since this is a Boolean gadget. */
                 /* (No binary mask) */
  0,             /* GadgetID, no id. */
  NULL           /* UserData, no user data connected to the gadget. */
};



/***********************************/
/* THE CANCEL GADGET's STRUCTURES: */
/***********************************/

/* The coordinates for the CANCEL box: */
SHORT cancel_border_points[]=
{
   0,  0, /* Start at position (0,0) */
  54,  0, /* Draw a line to the right to position (54,0) */
  54, 10, /* Draw a line down to position (54,10) */
   0, 10, /* Draw a line to the left to position (0,10) */
   0,  0  /* Finish of by drawing a line up to position (0,0) */
};

/* The Border structure: */
struct Border cancel_border=
{
  0, 0,        /* LeftEdge, TopEdge. */
  1,           /* FrontPen, colour register 1. */
  0,           /* BackPen, for the moment unused. */
  JAM1,        /* DrawMode, draw the lines with colour 1. */
  5,           /* Count, 5 pair of coordinates in the array. */
  cancel_border_points, /* XY, pointer to the array with the coord. */
  NULL,        /* NextBorder, no other Border structures are connected. */
};

/* The IntuiText structure: */
struct IntuiText cancel_text=
{
  1,        /* FrontPen, colour register 1. */
  0,        /* BackPen, not used since JAM1. */
  JAM1,     /* DrawMode, draw the characters with colour 1, do not */
            /* change the background. */
  4, 2,     /* LeftEdge, TopEdge. */
  NULL,     /* ITextFont, use default font. */
  "CANCEL", /* IText, the text that will be printed. */
  NULL,     /* NextText, no other IntuiText structures are connected. */
};

struct Gadget cancel_gadget=
{
  &ok_gadget,    /* NextGadget, linked to the OK gadget. */
  214,           /* LeftEdge, 214 pixels out. */
  47,            /* TopEdge, 47 lines down. */
  55,            /* Width, 55 pixels wide. */
  11,            /* Height, 11 pixels lines heigh. */
  GADGHCOMP,     /* Flags, when this gadget is highlighted, the gadget */
                 /* will be rendered in the complement colours. */
                 /* (Colour 0 (00) will be changed to colour 3 (11) */
                 /* (Colour 1 (01)           - " -           2 (10) */
                 /* (Colour 2 (10)           - " -           1 (01) */
                 /* (Colour 3 (11)           - " -           0 (00) */
  GADGIMMEDIATE| /* Activation, our program will recieve a message when */
  RELVERIFY|     /* the user has selected this gadget, and when the user */
                 /* has released it. */
  ENDGADGET,     /* When the user has selected this gadget, the */
                 /* requester is satisfied, and is deactivated. */
                 /* IMPORTANT! At least one gadget per requester */
                 /* must have the flag ENDGADGET set. If not, the */
                 /* requester would never be deactivated! */

  BOOLGADGET|    /* GadgetType, a Boolean gadget which is connected to */
  REQGADGET,     /* a requester. IMPORTANT! Every gadget which is */
                 /* connectd to a requester must have the REQGADGET flsg */
                 /* set in the GadgetType field. */
 (APTR) &cancel_border, /* GadgetRender, a pointer to our Border struc. */
  NULL,          /* SelectRender, NULL since we do not supply the gadget */
                 /* with an alternative image. (We complement the */
                 /* colours instead) */
 &cancel_text,   /* GadgetText, a pointer to our IntuiText structure. */
                 /* (See chapter 3 GRAPHICS for more information) */
  NULL,          /* MutualExclude, no mutual exclude. */
  NULL,          /* SpecialInfo, NULL since this is a Boolean gadget. */
                 /* (No binary mask) */
  0,             /* GadgetID, no id. */
  NULL           /* UserData, no user data connected to the gadget. */
};



/***********************************************************************/
```

```
/* Note:                                                              */
/* Remember that every gadget which is connected to a requester must    */
/* have the flag REQGADGET set in the GadgetType field. Remember also   */
/* that at least one gadget per requester must have the ENDGADGET flag  */
/* set in the Activation field.                                         */
/* In this example we have three gadgets connected to the requester.    */
/* All of them has the REQGADGET flag set, and the OK and CANCEL gadget */
/* has also the ENDGADGET flag set.                                     */
/************************************************************************/



/*************************************/
/* THE BORDER AROUND THE REQUESTER: */
/*************************************/

/* The coordinates for the box around the requester: */
SHORT requester_border_points[]=
{
    0,  0, /* Start at position (0,0) */
  282,  0, /* Draw a line to the right. */
  282, 64, /* Draw a line down. */
    0, 64, /* Draw a line to the left. */
    0,  0  /* Finish of by drawing a line up to position (0,0) */
};

/* The Border structure for the requester: */
struct Border requester_border=
{
  0, 0,        /* LeftEdge, TopEdge. */
  1,           /* FrontPen, colour register 1. */
  0,           /* BackPen, for the moment unused. */
  JAM1,        /* DrawMode, draw the lines with colour 1. */
  5,           /* Count, 5 pair of coordinates in the array. */
  requester_border_points, /* XY, pointer to the array with the coord. */
  NULL,        /* NextBorder, no other Border structures are connected. */
};



/***********************************/
/* THE TEXT INSIDE THE REQUESTER: */
/***********************************/

/* The IntuiText structure used to print some text inside the requester: */
struct IntuiText requester_text=
{
  1,         /* FrontPen, colour register 1. */
  0,         /* BackPen, unused since JAM1. */
  JAM1,      /* DrawMode, draw the characters with colour 1, do not */
             /* change the background. */
  14, 8,     /* LeftEdge, TopEdge. */
  NULL,      /* ITextFont, use default font. */
  "Please set the colour value:", /* IText, the text. */
  NULL,      /* NextText, no other IntuiText structures are connected. */
};



/*****************************/
/* THE REQUESTER STRUCTURE: */
/*****************************/

struct Requester my_requester=
{
  NULL,              /* OlderRequester, used by Intuition. */
  40, 20,            /* LeftEdge, TopEdge, 40 pixels out, 20 lines down. */
  283, 65,           /* Width, Height, 283 pixels wide, 65 lines high. */
  0, 0,              /* RelLeft, RelTop, Since POINTREL flag is not set, */
                     /* Intuition ignores these values. */
  &cancel_gadget,    /* ReqGadget, pointer to the first gadget. */
  &requester_border, /* ReqBorder, pointer to a Border structure. */
  &requester_text,   /* ReqText, pointer to a IntuiText structure. */
  NULL,              /* Flags, no flags set. */
  2,                 /* BackFill, draw everything on a black background. */
  NULL,              /* ReqLayer, used by Intuition. Set to NULL. */
  NULL,              /* ReqPad1, used by Intuition. Set to NULL. */
  NULL,              /* ImageBMap, no predrawn Bitmap. Set to NULL. */
                     /*           (The PREDRAWN flag was not set) */
  NULL,              /* RWindow, used by Intuition. Set to NULL. */
  NULL               /* ReqPad2, used by Intuition. Set to NULL. */
};



/* Declare a pointer to a Window structure: */
struct Window *my_window;

/* Declare and initialize your NewWindow structure: */
struct NewWindow my_new_window=
{
  0,             /* LeftEdge    x position of the window. */
  0,             /* TopEdge     y positio of the window. */
  640,           /* Width       640 pixels wide. */
  200,           /* Height      200 lines high. */
  0,             /* DetailPen   Text should be drawn with colour reg. 0 */
  1,             /* BlockPen    Blocks should be drawn with colour reg. 1 */
  CLOSEWINDOW|   /* IDCMPFlags  The window will give us a message if the */
                 /*             user has selected the Close window gad, */
  GADGETDOWN|    /*             or a gadget has been pressed on, or */
  GADGETUP|      /*             a gadge has been released. */
  REQSET|        /*             Send a message also if a requester has */
  REQCLEAR,      /*             been activated or deactivated. */
  SMART_REFRESH| /* Flags       Intuition should refresh the window. */
  WINDOWCLOSE|   /*             Close Gadget. */
  WINDOWDRAG|    /*             Drag gadget. */
  WINDOWDEPTH|   /*             Depth arrange Gadgets. */
  WINDOWSIZING|  /*             Sizing Gadget. */
  ACTIVATE,      /*             The window should be Active when opened. */
  NULL,          /* FirstGadget No gadget connected to this window. */
  NULL,          /* CheckMark   Use Intuition's default CheckMark. */
  "The Window",  /* Title       Title of the window. */
  NULL,          /* Screen      Connected to the Workbench Screen. */
  NULL,          /* BitMap      No Custom BitMap. */
  140,           /* MinWidth    We will not allow the window to become */
  50,            /* MinHeight   smaller than 140 x 50, and not bigger */
  300,           /* MaxWidth    than 300 x 200. */
  200,           /* MaxHeight */
  WBENCHSCREEN   /* Type        Connected to the Workbench Screen. */
};



main()
```

```
{
  /* Boolean variable used for the while loop: */
  BOOL close_me;

  /* Declare a variable in which we will store the IDCMP flag: */
  ULONG class;

  /* Declare a variable in which we will store the address of the */
  /* gadget which sent the message: */
  APTR address;

  /* Declare a pointer to an IntuiMessage structure: */
  struct IntuiMessage *my_message;

  /* We use this variable to check if the requester has ben activated */
  /* or not: */
  BOOL result;



  /* Before we can use Intuition we need to open the Intuition Library: */
  IntuitionBase = (struct IntuitionBase *)
    OpenLibrary( "intuition.library", 0 );

  if( IntuitionBase == NULL )
    exit(); /* Could NOT open the Intuition Library! */



  /* We will now try to open the window: */
  my_window = (struct Window *) OpenWindow( &my_new_window );

  /* Have we opened the window succesfully? */
  if(my_window == NULL)
  {
    /* Could NOT open the Window! */

    /* Close the Intuition Library since we have opened it: */
    CloseLibrary( IntuitionBase );

    exit();
  }



  /* We have opened the window, and everything seems to be OK. */



  /* We will now try to activate the requester: */
  result=Request( &my_requester, my_window );

  if( !result )  /* !result is the same thing as result==FALSE */
  {
    /* Intuition could not activate the requester! */
    /* In this case we do not need to quit since it does not matter if */
    /* the requester was activated or not. I just wanted to show how */
    /* you can check if you have opened or not the requester. */

    printf("Could not activate the requester!\n");
  }
  else
```

```
  {
    /* Intuition could open the requester! */
    printf("Try to close the window!\n");
  }



  close_me = FALSE;

  /* Stay in the while loop until the user has selected the Close window */
  /* gadget. However, in this example the user first need to deactivate */
  /* the requester before he can select the Close window gadget: */
  while( !close_me )
  {
    /* Wait until we have recieved a message: */
    Wait( 1 << my_window->UserPort->mp_SigBit );

    /* As long as we collect messages sucessfully: */
    while(my_message=(struct IntuiMessage *) GetMsg(my_window->UserPort))
    {
      /* After we have collected the message we can read it, and save any */
      /* important values which we maybe want to check later: */

      /* Store the IDCMP flag: */
      class = my_message->Class;

      /* Store the address: */
      address = my_message->IAddress;

      /* After we have read it we reply as fast as possible: */
      /* REMEMBER! Do never try to read a message after you have replied! */
      /* Some other process has maybe changed it. */
      ReplyMsg( my_message );

      /* Check which IDCMP flag was sent: */
      switch( class )
      {
        case CLOSEWINDOW:  /* The user selected the Close window gadget! */
               close_me=TRUE;
               break;

        case GADGETDOWN:   /* The user has pressed on a gadget. */

               if( address == (APTR) &ok_gadget )
                 printf("The user pressed on the OK gadget!\n");

               if( address == (APTR) &cancel_gadget )
                 printf("The user pressed on the CANCEL gadget!\n");

               if( address == (APTR) &prop_gadget )
                 printf("The user selected the Proportional gadget!\n");

               break;

        case GADGETUP:     /* The user has released a gadget. */

               if( address == (APTR) &ok_gadget )
                 printf("The user released the OK gadget!\n");

               if( address == (APTR) &cancel_gadget )
                 printf("The user released the CANCEL gadget!\n");
```

```
              if( address == (APTR) &prop_gadget )
              {
                printf("The user released the Proportional gadget!\n");

                /* Print out the colour value: */
                printf("Colour= %1.0f\n\n", (float) prop_info.HorizPot
                                            / MAXPOT*16);
              }
              break;

        case REQSET:       /* Requester activated. */
             printf("Requester activated!\n");
             break;

        case REQCLEAR:     /* Requester deactivated. */
             printf("Requester deactivated!\n");
             printf("You can now close the window.\n");
             break;
      }
    }
  }



  /* Print out the colour value: */
  printf( "Colour= %1.0f\n\n", (float) prop_info.HorizPot / MAXPOT*16 );



  /* We should always close the windows we have opened before we leave: */
  CloseWindow( my_window );



  /* Close the Intuition Library since we have opened it: */
  CloseLibrary( IntuitionBase );

  /* THE END */
}
```

A.6 ALERTS

Example1

This example displays an Alert message at the top of the display.

```
/* Example1                                                          */
/* This example displays an Alert message at the top of the display:   */
/*                                                                      */
/*  ----------------------------------------------------------------  */
/*  |                                                               |  */
/*  |  DANGER! Stupid user behind the keyboard!                     |  */
/*  |                                                               |  */
/*  |  Press Left Button to Retry   Press Right Button to Abort     |  */
/*  |                                                               |  */
/*  ----------------------------------------------------------------  */



#include <intuition/intuition.h>



struct IntuitionBase *IntuitionBase;



main()
{
  /* The string which will be printed out: */
  char message[106];

  /* In this variable will we store what DisplayAlert() returned: */
  BOOL result;



  /* Before we can use Intuition we need to open the Intuition Library: */
  IntuitionBase = (struct IntuitionBase *)
    OpenLibrary( "intuition.library", 0 );

  if( IntuitionBase == NULL )
    exit(); /* Could NOT open the Intuition Library! */



  /* We will now fill the message array with our requirements: */

  /* Put the first string into the array. Remember to give space for 3 */
  /* characters in the beginning. We will there store the x (2 bytes)  */
  /* and y (1 byte) position of the text:                              */

  strcpy( message, "   DANGER! Stupid user behind the keyboard!");


  /* Put the second string into the array. Remember to give space for   */
  /* 5 (!) characters/bytes. We will there store the NULL sign which    */
  /* finish of the first string, the TRUE sign which tells Intuition    */
  /* that another string will come, and three bytes used to position    */
  /* the text:                                                          */

  strcat( message,
          "     Press Left Button to Retry   Press Right Button to Abort");


  message[0]=0;       /* X position of the first string */
  message[1]=32;      /*                 - " -           */
  message[2]=16;      /* Y               - " -           */
```

```
  message[43]='\0';   /* NULL sign which finish of the first string. */
  message[44]=TRUE;   /* Continuation byte set to TRUE (new string). */

  message[45]=0;      /* X position of the second string. */
  message[46]=32;     /*                 - " -              */
  message[47]=32;     /* Y               - " -              */

  message[104]='\0';  /* NULL sign which finish of the second string. */
  message[105]=FALSE; /* Continuation byte set to FALSE (last string). */



  /* We will now display the Alert message: */
  result = DisplayAlert( RECOVERY_ALERT, message, 48 );

  /*****************************************************************/
  /* RECOVERY_ALERT: The system will survive after this message have */
  /*                 been displayed.                                 */
  /* message:        Pointer to the string which contains the text   */
  /*                 we want to display + information about where we */
  /*                 want to display it (x/y position) etc.          */
  /* 48:             The height of the Alert box. (48 lines high)    */
  /*****************************************************************/



  if(result)
  {
    /* result is equal to TRUE, left button was pressed: */
    printf("RETRY: Left button was pressed\n");
  }
  else
  {
    /* result is equal to FALSE, right button was pressed: */
    printf("ABORT: Right button was pressed\n");
  }



  /* Close the Intuition Library since we have opened it: */
  CloseLibrary( IntuitionBase );

  /* THE END */
}
```

A.7 MENUS

Example1

This program opens a normal window to which we connect a menu strip. The menu consists of four items: Plain, Bold, Underlined and Italic. The user can select either Plain or a combination of the other styles. (If the user selects Plain all other modes will be mutual excluded, but if the user on the other hand selects Bold, Underlined or Italic, only the Plain option will be mutual excluded.

This example also shows how a program should handle the IDCMP flags, and how to collect several messages from one single menu event.

```
/* Example1                                                         */
/* This program opens a normal window to which we connect a menu strip. */
/* The menu will look like this:                                    */
/*                                                                  */
/* Mode                                                             */
/* -----------------                                                */
/* | v Plain       |                                                */
/* |   Bold        |                                                */
/* |   Underlined  |                                                */
/* |   Italic      |                                                */
/* -----------------                                                */
/*                                                                  */
/* The user can select either Plain or a combination of the other   */
/* styles. (If the user selects Plain all other modes will be mutual */
/* excluded, but if the user on the other hand selects Bold, Underlined */
/* or Italic, the Plain option will be mutul excluded.              */
/*                                                                  */
/* This example also shows how a program should handle the IDCMP flags, */
/* and how to collect several messages from one single menu event.  */



#include <intuition/intuition.h>



struct IntuitionBase *IntuitionBase;



/***********************************************************************/
/*                          F O U R T H   I T E M                      */
/***********************************************************************/

/* The text for the fourth item: */
struct IntuiText my_fourth_text=
{
  2,            /* FrontPen, black. */
  0,            /* BackPen, not used since JAM1. */
  JAM1,         /* DrawMode, do not change the background. */
  CHECKWIDTH,   /* LeftEdge, CHECKWIDTH amount of pixels out. */
                /* This will leave enough space for the check mark. */
  1,            /* TopEdge, 1 line down. */
  NULL,         /* TextAttr, default font. */
  "Italic",     /* IText, the string. */
  NULL          /* NextItem, no link to other IntuiText structures. */
};

/* The MenuItem structure for the fourth item: */
struct MenuItem my_fourth_item=
{
  NULL,            /* NextItem, this is the last item in the list. */
  0,               /* LeftEdge, 0 pixels out. */
  30,              /* TopEdge, 30 lines down. */
  150,             /* Width, 150 pixels wide. */
  10,              /* Height, 10 lines high. */
  ITEMTEXT|        /* Flags, render this item with text. */
  ITEMENABLED|     /*        this item will be enabled. */
  CHECKIT|         /*        it is an attribute item. */
  HIGHCOMP,        /*        complement the colours when highlihted. */
  0x00000001,      /* MutualExclude, mutualexclude the first item only. */
  (APTR) &my_fourth_text, /* ItemFill, pointer to the text. */
  NULL,            /* SelectFill, nothing since we complement the col. */
  0,               /* Command, no command-key sequence. */
  NULL,            /* SubItem, no subitem list. */
  MENUNULL,        /* NextSelect, no items selected. */
};



/***********************************************************************/
/*                          T H I R D   I T E M                        */
/***********************************************************************/

/* The text for the third item: */
struct IntuiText my_third_text=
{
  2,            /* FrontPen, black. */
  0,            /* BackPen, not used since JAM1. */
  JAM1,         /* DrawMode, do not change the background. */
  CHECKWIDTH,   /* LeftEdge, CHECKWIDTH amount of pixels out. */
                /* This will leave enough space for the check mark. */
  1,            /* TopEdge, 1 line down. */
  NULL,         /* TextAttr, default font. */
  "Underlined", /* IText, the string. */
  NULL          /* NextItem, no link to other IntuiText structures. */
};

/* The MenuItem structure for the third item: */
struct MenuItem my_third_item=
{
  &my_fourth_item, /* NextItem, linked to the fourth item. */
  0,               /* LeftEdge, 0 pixels out. */
  20,              /* TopEdge, 20 lines down. */
  150,             /* Width, 150 pixels wide. */
  10,              /* Height, 10 lines high. */
  ITEMTEXT|        /* Flags, render this item with text. */
  ITEMENABLED|     /*        this item will be enabled. */
  CHECKIT|         /*        it is an attribute item. */
  HIGHCOMP,        /*        complement the colours when highlihted. */
  0x00000001,      /* MutualExclude, mutualexclude the first item only. */
  (APTR) &my_third_text, /* ItemFill, pointer to the text. */
  NULL,            /* SelectFill, nothing since we complement the col. */
  0,               /* Command, no command-key sequence. */
  NULL,            /* SubItem, no subitem list. */
  MENUNULL,        /* NextSelect, no items selected. */
};



/***********************************************************************/
/*                         S E C O N D   I T E M                       */
/***********************************************************************/

/* The text for the second item: */
struct IntuiText my_second_text=
{
  2,          /* FrontPen, black. */
  0,          /* BackPen, not used since JAM1. */
  JAM1,       /* DrawMode, do not change the background. */
  CHECKWIDTH, /* LeftEdge, CHECKWIDTH amount of pixels out. */
              /* This will leave enough space for the check mark. */
  1,          /* TopEdge, 1 line down. */
  NULL,       /* TextAttr, default font. */
  "Bold",     /* IText, the string. */
  NULL        /* NextItem, no link to other IntuiText structures. */
};

/* The MenuItem structure for the second item: */
struct MenuItem my_second_item=
{
  &my_third_item,  /* NextItem, linked to the third item. */
  0,               /* LeftEdge, 0 pixels out. */
  10,              /* TopEdge, 10 lines down. */
  150,             /* Width, 150 pixels wide. */
  10,              /* Height, 10 lines high. */
  ITEMTEXT|        /* Flags, render this item with text. */
  ITEMENABLED|     /*        this item will be enabled. */
  CHECKIT|         /*        it is an attribute item. */
  HIGHCOMP,        /*        complement the colours when highlihted. */
  0x00000001,      /* MutualExclude, mutualexclude the first item only. */
  (APTR) &my_second_text, /* ItemFill, pointer to the text. */
  NULL,            /* SelectFill, nothing since we complement the col. */
  0,               /* Command, no command-key sequence. */
  NULL,            /* SubItem, no subitem list. */
  MENUNULL,        /* NextSelect, no items selected. */
};



/**********************************************************************/
/*                         F I R S T   I T E M                        */
/**********************************************************************/

/* The text for the first item: */
struct IntuiText my_first_text=
{
  2,          /* FrontPen, black. */
  0,          /* BackPen, not used since JAM1. */
  JAM1,       /* DrawMode, do not change the background. */
  CHECKWIDTH, /* LeftEdge, CHECKWIDTH amount of pixels out. */
              /* This will leave enough space for the check mark. */
  1,          /* TopEdge, 1 line down. */
  NULL,       /* TextAttr, default font. */
  "Plain",    /* IText, the string. */
  NULL        /* NextItem, no link to other IntuiText structures. */
};

/* The MenuItem structure for the first item: */
struct MenuItem my_first_item=
{
  &my_second_item, /* NextItem, linked to the second item. */
  0,               /* LeftEdge, 0 pixels out. */
  0,               /* TopEdge, 0 lines down. */
  150,             /* Width, 150 pixels wide. */
  10,              /* Height, 10 lines high. */
  ITEMTEXT|        /* Flags, render this item with text. */
  ITEMENABLED|     /*        this item will be enabled. */
  CHECKIT|         /*        it is an attribute item. */
  CHECKED|         /*        this item is initially selected. */
  HIGHCOMP,        /*        complement the colours when highlihted. */
  0xFFFFFFFE,      /* MutualExclude, mutualexclude all items except the */
                   /*                first one. */
  (APTR) &my_first_text, /* ItemFill, pointer to the text. */
  NULL,            /* SelectFill, nothing since we complement the col. */
  0,               /* Command, no command-key sequence. */
```

```
  NULL,            /* SubItem, no subitem list. */
  MENUNULL,        /* NextSelect, no items selected. */
};



/**********************************************************************/
/*                               M E N U                              */
/**********************************************************************/

/* The Menu structure for the first (and only) menu: */
struct Menu my_menu=
{
  NULL,          /* NextMenu, no more menu structures. */
  0,             /* LeftEdge, left corner. */
  0,             /* TopEdge, for the moment ignored by Intuition. */
  50,            /* Width, 50 pixels wide. */
  0,             /* Height, for the moment ignored by Intuition. */
  MENUENABLED,   /* Flags, this menu will be enabled. */
  "Mode",        /* MenuName, the string. */
  &my_first_item /* FirstItem, pointer to the first item in the list. */
};



/* Declare a pointer to a Window structure: */
struct Window *my_window;

/* Declare and initialize your NewWindow structure: */
struct NewWindow my_new_window=
{
  50,            /* LeftEdge    x position of the window. */
  25,            /* TopEdge     y positio of the window. */
  200,           /* Width       200 pixels wide. */
  100,           /* Height      100 lines high. */
  0,             /* DetailPen   Text should be drawn with colour reg. 0 */
  1,             /* BlockPen    Blocks should be drawn with colour reg. 1 */
  CLOSEWINDOW|   /* IDCMPFlags  The window will give us a message if the */
                 /*             user has selected the Close window gad. */
  MENUPICK,
  SMART_REFRESH| /* Flags       Intuition should refresh the window. */
  WINDOWCLOSE|   /*             Close Gadget. */
  WINDOWDRAG|    /*             Drag gadget. */
  WINDOWDEPTH|   /*             Depth arrange Gadgets. */
  WINDOWSIZING|  /*             Sizing Gadget. */
  ACTIVATE,      /*             The window should be Active when opened. */
  NULL,          /* FirstGadget No Custom gadgets. */
  NULL,          /* CheckMark   Use Intuition's default CheckMark. */
  "Style Editor",/* Title       Title of the window. */
  NULL,          /* Screen      Connected to the Workbench Screen. */
  NULL,          /* BitMap      No Custom BitMap. */
  80,            /* MinWidth    We will not allow the window to become */
  30,            /* MinHeight   smaller than 80 x 30, and not bigger */
  300,           /* MaxWidth    than 300 x 200. */
  200,           /* MaxHeight */
  WBENCHSCREEN   /* Type        Connected to the Workbench Screen. */
};



main()
{
```

```
/* Boolean variable used for the while loop: */
BOOL close_me;

/* Declare a variable in which we will store the IDCMP flag: */
ULONG class;

/* If we recieve a MENUPICK event, the Code field of the message */
/* structure will contain the menu number of the first selected item. */
/* Declare a variable to store the Code value in, and an extra menu */
/* number variable: */
USHORT code, menu_number;

/* Declare a MenuItem pointer: */
struct MenuItem *item;

/* Declare a pointer to an IntuiMessage structure: */
struct IntuiMessage *my_message;



/* Before we can use Intuition we need to open the Intuition Library: */
IntuitionBase = (struct IntuitionBase *)
  OpenLibrary( "intuition.library", 0 );

if( IntuitionBase == NULL )
  exit(); /* Could NOT open the Intuition Library! */



/* We will now try to open the window: */
my_window = (struct Window *) OpenWindow( &my_new_window );

/* Have we opened the window succesfully? */
if(my_window == NULL)
{
  /* Could NOT open the Window! */

  /* Close the Intuition Library since we have opened it: */
  CloseLibrary( IntuitionBase );

  exit();
}



/* We have opened the window, and everything seems to be OK. */



SetMenuStrip( my_window, &my_menu );
printf("Menustrip connected to window!\n");


close_me = FALSE;

/* Stay in the while loop until the user has selected the Close window */
/* gadget: */
while( close_me == FALSE )
{
  /* Wait until we have recieved a message: */
  Wait( 1 << my_window->UserPort->mp_SigBit );
```

```
    /* As long as we collect messages sucessfully we stay in the loop: */
    while(my_message=(struct IntuiMessage *) GetMsg( my_window->UserPort ))
    {
      /* After we have collected the message we can read it, and save any */
      /* important values which we maybe want to check later: */
      class = my_message->Class;
      code = my_message->Code;


      /* After we have read it we reply as fast as possible: */
      /* REMEMBER! Do never try to read a message after you have replied! */
      /* Some other process has maybe changed it. */
      ReplyMsg( my_message );

      /* Check which IDCMP flag was sent: */
      if( class == CLOSEWINDOW )
        close_me=TRUE; /* The user selected the Close window gadget! */

      if(class == MENUPICK)
      {
        printf("\nMenu pick!\n");
        menu_number = code;

        while( menu_number != MENUNULL )
        {
          /* Get the address of the item: */
          item = (struct MenuItem *) ItemAddress( &my_menu, menu_number );


          /* Print out the menu number plus etc: */
          printf("menu_number= %d\n", menu_number );
          printf("MENUNUM = %d\n", MENUNUM(menu_number) );
          printf("ITEMNUM = %d\n", ITEMNUM(menu_number) );
          printf("SUBNUM  = %d\n", SUBNUM(menu_number) );


          /* Get the following item's menu number: */
          menu_number = item->NextSelect;
        }
      }
    }
  }



  printf("Menustrip removed from window!\n");
  ClearMenuStrip( my_window );



  /* Close the window: */
  CloseWindow( my_window );



  /* Close the Intuition Library since we have opened it: */
  CloseLibrary( IntuitionBase );

  /* THE END */
}
```

Example2

This example is very similar to Example1, but we have this time put the edit styles in a subitem box which is connected to the one and only item box called "Style".

```
/* Example2                                                          */
/* This program opens a normal window to which we connect a menu strip. */
/* The menu will look like this:                                     */
/*                                                                   */
/* Edit                                                              */
/* ----------                                                        */
/* | Style -----------------                                         */
/* --------| v Plain       |                                         */
/*         |   Bold        |                                         */
/*         |   Underlined  |                                         */
/*         |   Italic      |                                         */
/*         -----------------                                         */
/*                                                                   */
/* This example is very similar to Example1, but we have this time put  */
/* the edit styles in a subitem box which is connected to the one and   */
/* only item box called "Style"                                      */



#include <intuition/intuition.h>



struct IntuitionBase *IntuitionBase;



/**********************************************************************/
/*                      F O U R T H   S U B I T E M                   */
/**********************************************************************/

/* The text for the fourth subitem: */
struct IntuiText my_fourth_text=
{
  2,            /* FrontPen, black. */
  0,            /* BackPen, not used since JAM1. */
  JAM1,         /* DrawMode, do not change the background. */
  CHECKWIDTH,   /* LeftEdge, CHECKWIDTH amount of pixels out. */
                /* This will leave enough space for the check mark. */
  1,            /* TopEdge, 1 line down. */
  NULL,         /* TextAttr, default font. */
  "Italic",     /* IText, the string. */
  NULL          /* NextItem, no link to other IntuiText structures. */
};

/* The MenuItem structure for the fourth subitem: */
struct MenuItem my_fourth_subitem=
{
  NULL,            /* NextItem, this is the last subitem in the list. */
  50,              /* LeftEdge, 50 pixels out. */
  35,              /* TopEdge, 35 lines down. */
  150,             /* Width, 150 pixels wide. */
  10,              /* Height, 10 lines high. */
  ITEMTEXT|        /* Flags, render this item with text. */
  ITEMENABLED|     /*        this item will be enabled. */
  CHECKIT|         /*        it is an attribute item. */
  HIGHCOMP,        /*        complement the colours when highlihted. */
  0x00000001,      /* MutualExclude, mutualexclude the first subitem. */
  (APTR) &my_fourth_text, /* ItemFill, pointer to the text. */
  NULL,            /* SelectFill, nothing since we complement the col. */
  0,               /* Command, no command-key sequence. */
  NULL,            /* SubItem, ignored by Intuition. */
  MENUNULL,        /* NextSelect, no items selected. */
};



/**********************************************************************/
/*                         T H I R D   S U B I T E M                  */
/**********************************************************************/

/* The text for the third subitem: */
struct IntuiText my_third_text=
{
  2,            /* FrontPen, black. */
  0,            /* BackPen, not used since JAM1. */
  JAM1,         /* DrawMode, do not change the background. */
  CHECKWIDTH,   /* LeftEdge, CHECKWIDTH amount of pixels out. */
                /* This will leave enough space for the check mark. */
  1,            /* TopEdge, 1 line down. */
  NULL,         /* TextAttr, default font. */
  "Underlined", /* IText, the string. */
  NULL          /* NextItem, no link to other IntuiText structures. */
};

/* The MenuItem structure for the third subitem: */
struct MenuItem my_third_subitem=
{
  &my_fourth_subitem, /* NextItem, linked to the fourth subitem. */
  50,              /* LeftEdge, 50 pixels out. */
  25,              /* TopEdge, 25 lines down. */
  150,             /* Width, 150 pixels wide. */
  10,              /* Height, 10 lines high. */
  ITEMTEXT|        /* Flags, render this item with text. */
  ITEMENABLED|     /*        this item will be enabled. */
  CHECKIT|         /*        it is an attribute item. */
  HIGHCOMP,        /*        complement the colours when highlihted. */
  0x00000001,      /* MutualExclude, mutualexclude the first subitem. */
  (APTR) &my_third_text, /* ItemFill, pointer to the text. */
  NULL,            /* SelectFill, nothing since we complement the col. */
  0,               /* Command, no command-key sequence. */
  NULL,            /* SubItem, ignored by Intuition. */
  MENUNULL,        /* NextSelect, no items selected. */
};



/**********************************************************************/
/*                        S E C O N D   S U B I T E M                 */
/**********************************************************************/

/* The text for the second subitem: */
struct IntuiText my_second_text=
{
  2,          /* FrontPen, black. */
  0,          /* BackPen, not used since JAM1. */
  JAM1,       /* DrawMode, do not change the background. */
  CHECKWIDTH, /* LeftEdge, CHECKWIDTH amount of pixels out. */
              /* This will leave enough space for the check mark. */
  1,          /* TopEdge, 1 line down. */
  NULL,       /* TextAttr, default font. */
  "Bold",     /* IText, the string. */
  NULL        /* NextItem, no link to other IntuiText structures. */
};
```

```
/* The MenuItem structure for the second subitem: */
struct MenuItem my_second_subitem=
{
  &my_third_subitem, /* NextItem, linked to the third subitem. */
  50,              /* LeftEdge, 50 pixels out. */
  15,              /* TopEdge, 15 lines down. */
  150,             /* Width, 150 pixels wide. */
  10,              /* Height, 10 lines high. */
  ITEMTEXT|        /* Flags, render this item with text. */
  ITEMENABLED|     /*        this item will be enabled. */
  CHECKIT|         /*        it is an attribute item. */
  HIGHCOMP,        /*        complement the colours when highlihted. */
  0x00000001,      /* MutualExclude, mutualexclude the first subitem. */
  (APTR) &my_second_text, /* ItemFill, pointer to the text. */
  NULL,            /* SelectFill, nothing since we complement the col. */
  0,               /* Command, no command-key sequence. */
  NULL,            /* SubItem, ignored by Intuition. */
  MENUNULL,        /* NextSelect, no items selected. */
};



/***********************************************************************/
/*                     F I R S T   S U B I T E M                       */
/***********************************************************************/

/* The text for the first subitem: */
struct IntuiText my_first_text=
{
  2,          /* FrontPen, black. */
  0,          /* BackPen, not used since JAM1. */
  JAM1,       /* DrawMode, do not change the background. */
  CHECKWIDTH, /* LeftEdge, CHECKWIDTH amount of pixels out. */
              /* This will leave enough space for the check mark. */
  1,          /* TopEdge, 1 line down. */
  NULL,       /* TextAttr, default font. */
  "Plain",    /* IText, the string. */
  NULL        /* NextItem, no link to other IntuiText structures. */
};

/* The MenuItem structure for the first subitem: */
struct MenuItem my_first_subitem=
{
  &my_second_subitem, /* NextItem, linked to the second subitem. */
  50,              /* LeftEdge, 50 pixels out. */
  5,               /* TopEdge, 5 lines down. */
  150,             /* Width, 150 pixels wide. */
  10,              /* Height, 10 lines high. */
  ITEMTEXT|        /* Flags, render this item with text. */
  ITEMENABLED|     /*        this item will be enabled. */
  CHECKIT|         /*        it is an attribute item. */
  CHECKED|         /*        this item is initially selected. */
  HIGHCOMP,        /*        complement the colours when highlihted. */
  0xFFFFFFFE,      /* MutualExclude, mutualexclude all items except the */
                   /*                first one. */
  (APTR) &my_first_text, /* ItemFill, pointer to the text. */
  NULL,            /* SelectFill, nothing since we complement the col. */
  0,               /* Command, no command-key sequence. */
  NULL,            /* SubItem, ignored by Intuition. */
  MENUNULL,        /* NextSelect, no items selected. */
};
```

```
/***********************************************************************/
/*                       T H E   O N L Y   I T E M                     */
/***********************************************************************/

/* The text for the item: */
struct IntuiText my_text=
{
  2,          /* FrontPen, black. */
  0,          /* BackPen, not used since JAM1. */
  JAM1,       /* DrawMode, do not change the background. */
  0,          /* LeftEdge, 0 pixels out. */
              /* No space is needed for a check mark. */
  1,          /* TopEdge, 1 line down. */
  NULL,       /* TextAttr, default font. */
  "Style",    /* IText, the string. */
  NULL        /* NextItem, no link to other IntuiText structures. */
};

/* The MenuItem structure for the item: */
struct MenuItem my_item=
{
  NULL,              /* NextItem, no more items after this one. */
  0,                 /* LeftEdge, 0 pixels out. */
  0,                 /* TopEdge, 0 lines down. */
  100,               /* Width, 100 pixels wide. */
  10,                /* Height, 10 lines high. */
  ITEMTEXT|          /* Flags, render this item with text. */
  ITEMENABLED|       /*        this item will be enabled. */
                     /*        it is an action item. (CHECKIT is not set) */
  HIGHCOMP,          /*        complement the colours when highlihted. */
  0,                 /* MutualExclude, no mutualexclude. */
  (APTR) &my_text,   /* ItemFill, pointer to the text. */
  NULL,              /* SelectFill, nothing since we complement the col. */
  0,                 /* Command, no command-key sequence. */
  &my_first_subitem, /* SubItem, pointer to the first subitem. */
  MENUNULL,          /* NextSelect, no items selected. */
};



/***********************************************************************/
/*                              M E N U                                */
/***********************************************************************/

/* The Menu structure for the first (and only) menu: */
struct Menu my_menu=
{
  NULL,        /* NextMenu, no more menu structures. */
  0,           /* LeftEdge, left corner. */
  0,           /* TopEdge, for the moment ignored by Intuition. */
  50,          /* Width, 50 pixels wide. */
  0,           /* Height, for the moment ignored by Intuition. */
  MENUENABLED, /* Flags, this menu will be enabled. */
  "Edit",      /* MenuName, the string. */
  &my_item     /* FirstItem, pointer to the first (and only) item in */
               /* the list. */
};
```

```
/* Declare a pointer to a Window structure: */
struct Window *my_window;

/* Declare and initialize your NewWindow structure: */
struct NewWindow my_new_window=
{
  50,            /* LeftEdge    x position of the window. */
  25,            /* TopEdge     y positio of the window. */
  200,           /* Width       200 pixels wide. */
  100,           /* Height      100 lines high. */
  0,             /* DetailPen   Text should be drawn with colour reg. 0 */
  1,             /* BlockPen    Blocks should be drawn with colour reg. 1 */
  CLOSEWINDOW|   /* IDCMPFlags  The window will give us a message if the */
                 /*             user has selected the Close window gad. */
  MENUPICK,
  SMART_REFRESH| /* Flags       Intuition should refresh the window. */
  WINDOWCLOSE|   /*             Close Gadget. */
  WINDOWDRAG|    /*             Drag gadget. */
  WINDOWDEPTH|   /*             Depth arrange Gadgets. */
  WINDOWSIZING|  /*             Sizing Gadget. */
  ACTIVATE,      /*             The window should be Active when opened. */
  NULL,          /* FirstGadget No Custom gadgets. */
  NULL,          /* CheckMark   Use Intuition's default CheckMark. */
  "Style Editor",/* Title       Title of the window. */
  NULL,          /* Screen      Connected to the Workbench Screen. */
  NULL,          /* BitMap      No Custom BitMap. */
  80,            /* MinWidth    We will not allow the window to become */
  30,            /* MinHeight   smaller than 80 x 30, and not bigger */
  300,           /* MaxWidth    than 300 x 200. */
  200,           /* MaxHeight */
  WBENCHSCREEN   /* Type        Connected to the Workbench Screen. */
};



main()
{
  /* Boolean variable used for the while loop: */
  BOOL close_me;

  /* Declare a variable in which we will store the IDCMP flag: */
  ULONG class;

  /* If we recieve a MENUPICK event, the Code field of the message */
  /* structure will contain the menu number of the first selected item. */
  /* Declare a variable to store the Code value in, and an extra menu */
  /* number variable: */
  USHORT code, menu_number;

  /* Declare a MenuItem pointer: */
  struct MenuItem *item;

  /* Declare a pointer to an IntuiMessage structure: */
  struct IntuiMessage *my_message;



  /* Before we can use Intuition we need to open the Intuition Library: */
  IntuitionBase = (struct IntuitionBase *)
    OpenLibrary( "intuition.library", 0 );

  if( IntuitionBase == NULL )
    exit(); /* Could NOT open the Intuition Library! */



  /* We will now try to open the window: */
  my_window = (struct Window *) OpenWindow( &my_new_window );

  /* Have we opened the window succesfully? */
  if(my_window == NULL)
  {
    /* Could NOT open the Window! */

    /* Close the Intuition Library since we have opened it: */
    CloseLibrary( IntuitionBase );

    exit();
  }



  /* We have opened the window, and everything seems to be OK. */



  SetMenuStrip( my_window, &my_menu );
  printf("Menustrip connected to window!\n");


  close_me = FALSE;

  /* Stay in the while loop until the user has selected the Close window */
  /* gadget: */
  while( close_me == FALSE )
  {
    /* Wait until we have recieved a message: */
    Wait( 1 << my_window->UserPort->mp_SigBit );

    /* As long as we collect messages sucessfully we stay in the loop: */
    while(my_message=(struct IntuiMessage *) GetMsg( my_window->UserPort ))
    {
      /* After we have collected the message we can read it, and save any */
      /* important values which we maybe want to check later: */
      class = my_message->Class;
      code = my_message->Code;


      /* After we have read it we reply as fast as possible: */
      /* REMEMBER! Do never try to read a message after you have replied! */
      /* Some other process has maybe changed it. */
      ReplyMsg( my_message );

      /* Check which IDCMP flag was sent: */
      if( class == CLOSEWINDOW )
        close_me=TRUE; /* The user selected the Close window gadget! */

      if(class == MENUPICK)
      {
        printf("\nMenu pick!\n");
        menu_number = code;

        while( menu_number != MENUNULL )
        {
          /* Get the address of the item: */
          item = (struct MenuItem *) ItemAddress( &my_menu, menu_number );


          /* Print out the menu number plus etc: */
          printf("menu_number= %d\n", menu_number );
          printf("MENUNUM = %d\n", MENUNUM(menu_number) );
          printf("ITEMNUM = %d\n", ITEMNUM(menu_number) );
          printf("SUBNUM  = %d\n", SUBNUM(menu_number) );


          /* Get the following item's menu number: */
          menu_number = item->NextSelect;
        }
      }
    }
  }



  printf("Menustrip removed from window!\n");
  ClearMenuStrip( my_window );



  /* Close the window: */
  CloseWindow( my_window );



  /* Close the Intuition Library since we have opened it: */
  CloseLibrary( IntuitionBase );

  /* THE END */
}
```

Example3

This example is very similar to Example2, but the user can this time also access the subitems from the keyboard. For example, to select Bold the user only needs to press the right Amiga key [A] together with the "B" key.

```
/* Example3                                                           */
/* This program opens a normal window to which we connect a menu strip. */
/* The menu will look like this:                                      */
/*                                                                    */
/* Edit                                                               */
/* ----------                                                         */
/* | Style -----------------------                                    */
/* --------| v Plain       [A] P |                                    */
/*         |   Bold        [A] B |                                    */
/*         |   Underlined  [A] U |                                    */
/*         |   Italic      [A] I |                                    */
/*         -----------------------                                    */
/*                                                                    */
/* This example is very similar to Example2, but the user can this time */
/* also access the subitems from the keyboard. For example, to select   */
/* Bold the user only needs to press the right Amiga key [A] together   */
/* with the "B" key. */



#include <intuition/intuition.h>



struct IntuitionBase *IntuitionBase;



/***********************************************************************/
/*                     F O U R T H   S U B I T E M                     */
/***********************************************************************/

/* The text for the fourth subitem: */
struct IntuiText my_fourth_text=
{
  2,            /* FrontPen, black. */
  0,            /* BackPen, not used since JAM1. */
  JAM1,         /* DrawMode, do not change the background. */
  CHECKWIDTH,   /* LeftEdge, CHECKWIDTH amount of pixels out. */
                /* This will leave enough space for the check mark. */
  1,            /* TopEdge, 1 line down. */
  NULL,         /* TextAttr, default font. */
  "Italic",     /* IText, the string. */
  NULL          /* NextItem, no link to other IntuiText structures. */
};

/* The MenuItem structure for the fourth subitem: */
struct MenuItem my_fourth_subitem=
{
  NULL,             /* NextItem, this is the last subitem in the list. */
  50,               /* LeftEdge, 50 pixels out. */
  35,               /* TopEdge, 35 lines down. */
  150,              /* Width, 150 pixels wide. */
                    /*        150 pixels is enough in this example (the */
                    /*        Amiga key + character fits perfectly), but */
                    /*        if you are not sure you can always add the */
                    /*        constant COMMWIDTH. Eg, 150 + COMMWIDTH. */
  10,               /* Height, 10 lines high. */
  ITEMTEXT|         /* Flags, render this item with text. */
  ITEMENABLED|      /*        this item will be enabled. */
  CHECKIT|          /*        it is an attribute item. */
  COMMSEQ|          /*        also accessable from the keyboard. */
  HIGHCOMP,         /*        complement the colours when highlihted. */
  0x00000001,       /* MutualExclude, mutualexclude the first subitem. */
  (APTR) &my_fourth_text, /* ItemFill, pointer to the text. */
  NULL,             /* SelectFill, nothing since we complement the col. */
  'I',              /* Command, the user can select this item by      */
                    /*          pressing the right Amiga key together */
                    /*          with the I key. Remember to:          */
                    /*          1. Set the flag COMMSEQ.              */
                    /*          2. Make the itembox wide enough.      */
                    /*          (Intuition does not care if you write */
                    /*          a capital letter or not. Pressing the */
                    /*          Amiga key together with an 'I' or an  */
                    /*          'i' makes no difference.)             */
  NULL,             /* SubItem, ignored by Intuition. */
  MENUNULL,         /* NextSelect, no items selected. */
};



/***********************************************************************/
/*                      T H I R D   S U B I T E M                      */
/***********************************************************************/

/* The text for the third subitem: */
struct IntuiText my_third_text=
{
  2,            /* FrontPen, black. */
  0,            /* BackPen, not used since JAM1. */
  JAM1,         /* DrawMode, do not change the background. */
  CHECKWIDTH,   /* LeftEdge, CHECKWIDTH amount of pixels out. */
                /* This will leave enough space for the check mark. */
  1,            /* TopEdge, 1 line down. */
  NULL,         /* TextAttr, default font. */
  "Underlined", /* IText, the string. */
  NULL          /* NextItem, no link to other IntuiText structures. */
};

/* The MenuItem structure for the third subitem: */
struct MenuItem my_third_subitem=
{
  &my_fourth_subitem, /* NextItem, linked to the fourth subitem. */
  50,               /* LeftEdge, 50 pixels out. */
  25,               /* TopEdge, 25 lines down. */
  150,              /* Width, 150 pixels wide. */
  10,               /* Height, 10 lines high. */
  ITEMTEXT|         /* Flags, render this item with text. */
  ITEMENABLED|      /*        this item will be enabled. */
  CHECKIT|          /*        it is an attribute item. */
  COMMSEQ|          /*        also accessable from the keyboard. */
  HIGHCOMP,         /*        complement the colours when highlihted. */
  0x00000001,       /* MutualExclude, mutualexclude the first subitem. */
  (APTR) &my_third_text, /* ItemFill, pointer to the text. */
  NULL,             /* SelectFill, nothing since we complement the col. */
  'U',              /* Command, the user can select this item by      */
                    /*          pressing the right Amiga key together */
                    /*          with the U key. */
  NULL,             /* SubItem, ignored by Intuition. */
  MENUNULL,         /* NextSelect, no items selected. */
};
```

```
/***********************************************************************/
/*                      S E C O N D   S U B I T E M                    */
/***********************************************************************/

/* The text for the second subitem: */
struct IntuiText my_second_text=
{
  2,          /* FrontPen, black. */
  0,          /* BackPen, not used since JAM1. */
  JAM1,       /* DrawMode, do not change the background. */
  CHECKWIDTH, /* LeftEdge, CHECKWIDTH amount of pixels out. */
              /* This will leave enough space for the check mark. */
  1,          /* TopEdge, 1 line down. */
  NULL,       /* TextAttr, default font. */
  "Bold",     /* IText, the string. */
  NULL        /* NextItem, no link to other IntuiText structures. */
};

/* The MenuItem structure for the second subitem: */
struct MenuItem my_second_subitem=
{
  &my_third_subitem, /* NextItem, linked to the third subitem. */
  50,              /* LeftEdge, 50 pixels out. */
  15,              /* TopEdge, 15 lines down. */
  150,             /* Width, 150 pixels wide. */
  10,              /* Height, 10 lines high. */
  ITEMTEXT|        /* Flags, render this item with text. */
  ITEMENABLED|     /*        this item will be enabled. */
  CHECKIT|         /*        it is an attribute item. */
  COMMSEQ|         /*        also accessable from the keyboard. */
  HIGHCOMP,        /*        complement the colours when highlihted. */
  0x00000001,      /* MutualExclude, mutualexclude the first subitem. */
  (APTR) &my_second_text, /* ItemFill, pointer to the text. */
  NULL,            /* SelectFill, nothing since we complement the col. */
  'B',             /* Command, the user can select this item by      */
                   /*          pressing the right Amiga key together */
                   /*          with the B key. */
  NULL,            /* SubItem, ignored by Intuition. */
  MENUNULL,        /* NextSelect, no items selected. */
};



/***********************************************************************/
/*                       F I R S T   S U B I T E M                     */
/***********************************************************************/

/* The text for the first subitem: */
struct IntuiText my_first_text=
{
  2,          /* FrontPen, black. */
  0,          /* BackPen, not used since JAM1. */
  JAM1,       /* DrawMode, do not change the background. */
  CHECKWIDTH, /* LeftEdge, CHECKWIDTH amount of pixels out. */
              /* This will leave enough space for the check mark. */
  1,          /* TopEdge, 1 line down. */
  NULL,       /* TextAttr, default font. */
  "Plain",    /* IText, the string. */
  NULL        /* NextItem, no link to other IntuiText structures. */
};

/* The MenuItem structure for the first subitem: */
struct MenuItem my_first_subitem=
{
  &my_second_subitem, /* NextItem, linked to the second subitem. */
  50,              /* LeftEdge, 50 pixels out. */
  5,               /* TopEdge, 5 lines down. */
  150,             /* Width, 150 pixels wide. */
  10,              /* Height, 10 lines high. */
  ITEMTEXT|        /* Flags, render this item with text. */
  ITEMENABLED|     /*        this item will be enabled. */
  CHECKIT|         /*        it is an attribute item. */
  CHECKED|         /*        this item is initially selected. */
  COMMSEQ|         /*        also accessable from the keyboard. */
  HIGHCOMP,        /*        complement the colours when highlihted. */
  0xFFFFFFFE,      /* MutualExclude, mutualexclude all items except the */
                   /*                first one. */
  (APTR) &my_first_text, /* ItemFill, pointer to the text. */
  NULL,            /* SelectFill, nothing since we complement the col. */
  'P',             /* Command, the user can select this item by      */
                   /*          pressing the right Amiga key together */
                   /*          with the P key. */
  NULL,            /* SubItem, ignored by Intuition. */
  MENUNULL,        /* NextSelect, no items selected. */
};



/***********************************************************************/
/*                         T H E   O N L Y   I T E M                   */
/***********************************************************************/

/* The text for the item: */
struct IntuiText my_text=
{
  2,          /* FrontPen, black. */
  0,          /* BackPen, not used since JAM1. */
  JAM1,       /* DrawMode, do not change the background. */
  0,          /* LeftEdge, 0 pixels out. */
              /* No space is needed for a check mark. */
  1,          /* TopEdge, 1 line down. */
  NULL,       /* TextAttr, default font. */
  "Style",    /* IText, the string. */
  NULL        /* NextItem, no link to other IntuiText structures. */
};

/* The MenuItem structure for the item: */
struct MenuItem my_item=
{
  NULL,              /* NextItem, no more items after this one. */
  0,                 /* LeftEdge, 0 pixels out. */
  0,                 /* TopEdge, 0 lines down. */
  100,               /* Width, 100 pixels wide. */
  10,                /* Height, 10 lines high. */
  ITEMTEXT|          /* Flags, render this item with text. */
  ITEMENABLED|       /*        this item will be enabled. */
                     /*        it is an action item. (CHECKIT is not set) */
  HIGHCOMP,          /*        complement the colours when highlihted. */
  0,                 /* MutualExclude, no mutualexclude. */
  (APTR) &my_text,   /* ItemFill, pointer to the text. */
  NULL,              /* SelectFill, nothing since we complement the col. */
  0,                 /* Command, no command-key sequence. */
  &my_first_subitem, /* SubItem, pointer to the first subitem. */
  MENUNULL,          /* NextSelect, no items selected. */
```

```c
};



/************************************************************************/
/*                              M E N U                                 */
/************************************************************************/

/* The Menu structure for the first (and only) menu: */
struct Menu my_menu=
{
  NULL,        /* NextMenu, no more menu structures. */
  0,           /* LeftEdge, left corner. */
  0,           /* TopEdge, for the moment ignored by Intuition. */
  50,          /* Width, 50 pixels wide. */
  0,           /* Height, for the moment ignored by Intuition. */
  MENUENABLED, /* Flags, this menu will be enabled. */
  "Edit",      /* MenuName, the string. */
  &my_item     /* FirstItem, pointer to the first (and only) item in */
               /* the list. */
};



/* Declare a pointer to a Window structure: */
struct Window *my_window;

/* Declare and initialize your NewWindow structure: */
struct NewWindow my_new_window=
{
  50,             /* LeftEdge    x position of the window. */
  25,             /* TopEdge     y positio of the window. */
  200,            /* Width       200 pixels wide. */
  100,            /* Height      100 lines high. */
  0,              /* DetailPen   Text should be drawn with colour reg. 0 */
  1,              /* BlockPen    Blocks should be drawn with colour reg. 1 */
  CLOSEWINDOW|    /* IDCMPFlags  The window will give us a message if the */
                  /*             user has selected the Close window gad. */
  MENUPICK,
  SMART_REFRESH|  /* Flags       Intuition should refresh the window. */
  WINDOWCLOSE|    /*             Close Gadget. */
  WINDOWDRAG|     /*             Drag gadget. */
  WINDOWDEPTH|    /*             Depth arrange Gadgets. */
  WINDOWSIZING|   /*             Sizing Gadget. */
  ACTIVATE,       /*             The window should be Active when opened. */
  NULL,           /* FirstGadget No Custom gadgets. */
  NULL,           /* CheckMark   Use Intuition's default CheckMark. */
  "Style Editor",/* Title       Title of the window. */
  NULL,           /* Screen      Connected to the Workbench Screen. */
  NULL,           /* BitMap      No Custom BitMap. */
  80,             /* MinWidth    We will not allow the window to become */
  30,             /* MinHeight   smaller than 80 x 30, and not bigger */
  300,            /* MaxWidth    than 300 x 200. */
  200,            /* MaxHeight */
  WBENCHSCREEN    /* Type        Connected to the Workbench Screen. */
};



main()
{
  /* Boolean variable used for the while loop: */
  BOOL close_me;

  /* Declare a variable in which we will store the IDCMP flag: */
  ULONG class;

  /* If we recieve a MENUPICK event, the Code field of the message */
  /* structure will contain the menu number of the first selected item. */
  /* Declare a variable to store the Code value in, and an extra menu */
  /* number variable: */
  USHORT code, menu_number;

  /* Declare a MenuItem pointer: */
  struct MenuItem *item;

  /* Declare a pointer to an IntuiMessage structure: */
  struct IntuiMessage *my_message;



  /* Before we can use Intuition we need to open the Intuition Library: */
  IntuitionBase = (struct IntuitionBase *)
    OpenLibrary( "intuition.library", 0 );

  if( IntuitionBase == NULL )
    exit(); /* Could NOT open the Intuition Library! */



  /* We will now try to open the window: */
  my_window = (struct Window *) OpenWindow( &my_new_window );

  /* Have we opened the window succesfully? */
  if(my_window == NULL)
  {
    /* Could NOT open the Window! */

    /* Close the Intuition Library since we have opened it: */
    CloseLibrary( IntuitionBase );

    exit();
  }



  /* We have opened the window, and everything seems to be OK. */



  SetMenuStrip( my_window, &my_menu );
  printf("Menustrip connected to window!\n");


  close_me = FALSE;

  /* Stay in the while loop until the user has selected the Close window */
  /* gadget: */
  while( close_me == FALSE )
  {
    /* Wait until we have recieved a message: */
    Wait( 1 << my_window->UserPort->mp_SigBit );

    /* As long as we collect messages sucessfully we stay in the loop: */
    while(my_message=(struct IntuiMessage *) GetMsg( my_window->UserPort ))
    {
      /* After we have collected the message we can read it, and save any */
      /* important values which we maybe want to check later: */
      class = my_message->Class;
      code = my_message->Code;


      /* After we have read it we reply as fast as possible: */
      /* REMEMBER! Do never try to read a message after you have replied! */
      /* Some other process has maybe changed it. */
      ReplyMsg( my_message );

      /* Check which IDCMP flag was sent: */
      if( class == CLOSEWINDOW )
        close_me=TRUE; /* The user selected the Close window gadget! */

      if(class == MENUPICK)
      {
        printf("\nMenu pick!\n");
        menu_number = code;

        while( menu_number != MENUNULL )
        {
          /* Get the address of the item: */
          item = (struct MenuItem *) ItemAddress( &my_menu, menu_number );


          /* Print out the menu number plus etc: */
          printf("menu_number= %d\n", menu_number );
          printf("MENUNUM = %d\n", MENUNUM(menu_number) );
          printf("ITEMNUM = %d\n", ITEMNUM(menu_number) );
          printf("SUBNUM  = %d\n", SUBNUM(menu_number) );


          /* Get the following item's menu number: */
          menu_number = item->NextSelect;
        }
      }
    }
  }



  printf("Menustrip removed from window!\n");
  ClearMenuStrip( my_window );



  /* Close the window: */
  CloseWindow( my_window );



  /* Close the Intuition Library since we have opened it: */
  CloseLibrary( IntuitionBase );

  /* THE END */
}
```

Example4

This program opens a normal window to which we connect a menu strip. The menu consists of two items: Readmode and Editmode. The readmode item is selected and ghosted, and when the user selects the editmode item, it will become disabled (ghosted) while the readmode item will be enabled (not ghosted). This means that if the program is in "readmode", the user should only be able to chose the "editmode", and v.v. The purpose with this program is to show how you can use the OnMenu and OffMenu functions in order to make an "user-friendly interface".

```
/* Example4                                                              */
/* This program opens a normal window to which we connect a menu strip. */
/* The menu will look like this:                                         */
/*                                                                       */
/* Status                                                                */
/* --------------                                                        */
/* | v Readmode | (ghosted)                                              */
/* |   Editmode |                                                        */
/* --------------                                                        */
/*                                                                       */
/* The Readmode item is selected and ghosted, and when the user selects */
/* the Editmode item, it will become disabled (ghosted) while the read- */
/* mode item will be enabled (not ghosted). This means that if the      */
/* program is in "readmode", the user should only be able to chose the  */
/* "editmode", and v.v.                                                  */
/*                                                                       */
/* The purpose with this program is to show how you can use the OnMenu  */
/* and OffMenu functions inorder to make an "user-friendly interface".  */



#include <intuition/intuition.h>



struct IntuitionBase *IntuitionBase;



/***********************************************************************/
/*                        E D I T M O D E   I T E M                    */
/***********************************************************************/

/* The text for the editmode item: */
struct IntuiText my_editmode_text=
{
  2,          /* FrontPen, black. */
  0,          /* BackPen, not used since JAM1. */
  JAM1,       /* DrawMode, do not change the background. */
  CHECKWIDTH, /* LeftEdge, CHECKWIDTH amount of pixels out. */
              /* This will leave enough space for the check mark. */
  1,          /* TopEdge, 1 line down. */
  NULL,       /* TextAttr, default font. */
  "Editmode", /* IText, the string. */
  NULL        /* NextItem, no link to other IntuiText structures. */
};

/* The MenuItem structure for the editmode item: */
struct MenuItem my_editmode_item=
{
  NULL,             /* NextItem, last item in the list. */
  0,                /* LeftEdge, 0 pixels out. */
  10,               /* TopEdge, 10 lines down. */
  150,              /* Width, 150 pixels wide. */
  10,               /* Height, 10 lines high. */
  ITEMTEXT|         /* Flags, render this item with text. */
  ITEMENABLED|      /*        this item will be enabled. */
  CHECKIT|          /*        it is an attribute item. */
  HIGHCOMP,         /*        complement the colours when highlihted. */
  0xFFFFFFFD,       /* MutualExclude, mutualexclude all items except the */
                    /*                second (this) one. */
  (APTR) &my_editmode_text, /* ItemFill, pointer to the text. */
  NULL,             /* SelectFill, nothing since we complement the col. */
  0,                /* Command, not accessable from the keyboard. */
  NULL,             /* SubItem, ignored by Intuition. */
  MENUNULL,         /* NextSelect, no items selected. */
};



/***********************************************************************/
/*                        R E A D M O D E   I T E M                    */
/***********************************************************************/

/* The text for the readmode item: */
struct IntuiText my_readmode_text=
{
  2,          /* FrontPen, black. */
  0,          /* BackPen, not used since JAM1. */
  JAM1,       /* DrawMode, do not change the background. */
  CHECKWIDTH, /* LeftEdge, CHECKWIDTH amount of pixels out. */
              /* This will leave enough space for the check mark. */
  1,          /* TopEdge, 1 line down. */
  NULL,       /* TextAttr, default font. */
  "Readmode", /* IText, the string. */
  NULL        /* NextItem, no link to other IntuiText structures. */
};

/* The MenuItem structure for the readmode item: */
struct MenuItem my_readmode_item=
{
  &my_editmode_item, /* NextItem, pointer to the second (edit) item. */
  0,                 /* LeftEdge, 0 pixels out. */
  0,                 /* TopEdge, 0 lines down. */
  150,               /* Width, 150 pixels wide. */
  10,                /* Height, 10 lines high. */
  ITEMTEXT|          /* Flags, render this item with text. */
                     /*        this item will be disabled. */
  CHECKIT|           /*        it is an attribute item. */
  CHECKED|           /*        this item is initially selected. */
  HIGHCOMP,          /*        complement the colours when highlihted. */
  0xFFFFFFFE,        /* MutualExclude, mutualexclude all items except the */
                     /*                first (this) one. */
  (APTR) &my_readmode_text, /* ItemFill, pointer to the text. */
  NULL,              /* SelectFill, nothing since we complement the col. */
  0,                 /* Command, not accessable from the keyboard. */
  NULL,              /* SubItem, ignored by Intuition. */
  MENUNULL,          /* NextSelect, no items selected. */
};



/***********************************************************************/
/*                              M E N U                                */
/***********************************************************************/

/* The Menu structure for the first (and only) menu: */
struct Menu my_menu=
{
  NULL,             /* NextMenu, no more menu structures. */
  0,                /* LeftEdge, left corner. */
  0,                /* TopEdge, for the moment ignored by Intuition. */
  50,               /* Width, 50 pixels wide. */
  0,                /* Height, for the moment ignored by Intuition. */
  MENUENABLED,      /* Flags, this menu will be enabled. */
  "Status",         /* MenuName, the string. */
  &my_readmode_item /* FirstItem, pointer to the first item in the list. */
};



/* Declare a pointer to a Window structure: */
struct Window *my_window;

/* Declare and initialize your NewWindow structure: */
struct NewWindow my_new_window=
{
  50,            /* LeftEdge    x position of the window. */
  25,            /* TopEdge     y positio of the window. */
  250,           /* Width       250 pixels wide. */
  100,           /* Height      100 lines high. */
  0,             /* DetailPen   Text should be drawn with colour reg. 0 */
  1,             /* BlockPen    Blocks should be drawn with colour reg. 1 */
  CLOSEWINDOW|   /* IDCMPFlags  The window will give us a message if the */
                 /*             user has selected the Close window gad. */
  MENUPICK,
  SMART_REFRESH| /* Flags       Intuition should refresh the window. */
  WINDOWCLOSE|   /*             Close Gadget. */
  WINDOWDRAG|    /*             Drag gadget. */
  WINDOWDEPTH|   /*             Depth arrange Gadgets. */
  WINDOWSIZING|  /*             Sizing Gadget. */
  ACTIVATE,      /*             The window should be Active when opened. */
  NULL,          /* FirstGadget No Custom gadgets. */
  NULL,          /* CheckMark   Use Intuition's default CheckMark. */
  "Read or Edit", /* Title      Title of the window. */
  NULL,          /* Screen      Connected to the Workbench Screen. */
  NULL,          /* BitMap      No Custom BitMap. */
  80,            /* MinWidth    We will not allow the window to become */
  30,            /* MinHeight   smaller than 80 x 30, and not bigger */
  300,           /* MaxWidth    than 300 x 200. */
  200,           /* MaxHeight */
  WBENCHSCREEN   /* Type        Connected to the Workbench Screen. */
};



main()
{
  /* Boolean variable used for the while loop: */
  BOOL close_me;

  /* Declare a variable in which we will store the IDCMP flag: */
  ULONG class;

  /* If we recieve a MENUPICK event, the Code field of the message */
  /* structure will contain the menu number of the first selected item. */
  /* Declare a variable to store the Code value in, and two extra menu */
  /* number variables: */
  USHORT code, menu_number, number;

  /* Declare a MenuItem pointer: */
  struct MenuItem *item;

  /* Declare a pointer to an IntuiMessage structure: */
  struct IntuiMessage *my_message;
```

```
/* Before we can use Intuition we need to open the Intuition Library: */
IntuitionBase = (struct IntuitionBase *)
  OpenLibrary( "intuition.library", 0 );

if( IntuitionBase == NULL )
  exit(); /* Could NOT open the Intuition Library! */



/* We will now try to open the window: */
my_window = (struct Window *) OpenWindow( &my_new_window );

/* Have we opened the window succesfully? */
if(my_window == NULL)
{
  /* Could NOT open the Window! */

  /* Close the Intuition Library since we have opened it: */
  CloseLibrary( IntuitionBase );

  exit();
}



/* We have opened the window, and everything seems to be OK. */



SetMenuStrip( my_window, &my_menu );
printf("Menustrip connected to window!\n");


close_me = FALSE;

/* Stay in the while loop until the user has selected the Close window */
/* gadget: */
while( close_me == FALSE )
{
  /* Wait until we have recieved a message: */
  Wait( 1 << my_window->UserPort->mp_SigBit );

  /* As long as we collect messages sucessfully we stay in the loop: */
  while(my_message=(struct IntuiMessage *) GetMsg( my_window->UserPort ))
  {
    /* After we have collected the message we can read it, and save any */
    /* important values which we maybe want to check later: */
    class = my_message->Class;
    code = my_message->Code;


    /* After we have read it we reply as fast as possible: */
    /* REMEMBER! Do never try to read a message after you have replied! */
    /* Some other process has maybe changed it. */
    ReplyMsg( my_message );

    /* Check which IDCMP flag was sent: */
    if( class == CLOSEWINDOW )
      close_me=TRUE; /* The user selected the Close window gadget! */
    if(class == MENUPICK)
    {
      printf("\nMenu pick!\n");
      menu_number = code;

      while( menu_number != MENUNULL )
      {
        /* Get the address of the item: */
        item = (struct MenuItem *) ItemAddress( &my_menu, menu_number );



        /* Check which item was selected: */
        if( item == &my_readmode_item )
        {
          /* The Readmode (first) item was selected! */
          printf("We are now in READMODE!\n");

          /* Disable the Readmode item: */
          number = SHIFTMENU( 0 ) + SHIFTITEM( 0 ) + SHIFTSUB( NOSUB );
          /*       first menu       first item       no subitem. */
          OffMenu( my_window, number );

          /* Enable the Editmode item: */
          number = SHIFTMENU( 0 ) + SHIFTITEM( 1 ) + SHIFTSUB( NOSUB );
          /*       first menu       second item      no subitem. */
          OnMenu( my_window, number );
        }

        if( item == &my_editmode_item )
        {
          /* The Editmode (second) item was selected! */
          printf("We are now in EDITMODE!\n");

          /* Disable the Editmode item: */
          number = SHIFTMENU( 0 ) + SHIFTITEM( 1 ) + SHIFTSUB( NOSUB );
          /*       first menu       second item      no subitem. */
          OffMenu( my_window, number );

          /* Enable the Readmode item: */
          number = SHIFTMENU( 0 ) + SHIFTITEM( 0 ) + SHIFTSUB( NOSUB );
          /*       first menu       first item       no subitem. */
          OnMenu( my_window, number );
        }



        /* Get the following item's menu number: */
        menu_number = item->NextSelect;
      }
    }
  }
}



printf("Menustrip removed from window!\n");
ClearMenuStrip( my_window );



/* Close the window: */
```

```
  CloseWindow( my_window );



  /* Close the Intuition Library since we have opened it: */
  CloseLibrary( IntuitionBase );

  /* THE END */
}
```

Example5

Exactly as Example1 except that we have changed Intuition's
checkmark to our own customized "arrow".

```
/* Example5                                                               */
/* This program opens a normal window to which we connect a menu strip. */
/* The menu will look like this:                                          */
/*                                                                        */
/* Mode                                                                   */
/* ------------------                                                     */
/* | -> Plain       |                                                     */
/* |    Bold        |                                                     */
/* |    Underlined  |                                                     */
/* |    Italic      |                                                     */
/* ------------------                                                     */
/*                                                                        */
/* Exactly as Example1 except that we have changed Intuition's check-   */
/* mark to our own customized "arrow". If you want to use your own      */
/* image instead of Intuition's default one you need to:                */
/* 1. Declare and initialize an Image structure with your requirements. */
/* 2. Set the CheckMark field in the NewWindow structure to point at    */
/*    your Image.                                                         */



#include <intuition/intuition.h>



struct IntuitionBase *IntuitionBase;



/************************************************************************/
/*                            T H E   S P O T                           */
/************************************************************************/

/* Data for the spot: */
USHORT chip spot_data[]=
{
  0x0300,  /* 0000 0011 0000   0 : White   */
  0x0180,  /* 0000 0001 1000   1 : Orange  */
  0x00C0,  /* 0000 0000 1100               */
  0x0060,  /* 0000 0000 0110               */
  0xFFF0,  /* 1111 1111 1111               */
  0xFFF0,  /* 1111 1111 1111               */
  0x0060,  /* 0000 0000 0110               */
  0x00C0,  /* 0000 0000 1100               */
  0x0180,  /* 0000 0001 1000               */
  0x0300,  /* 0000 0011 0000               */
};

/* The spot's Image structure: */
struct Image spot=
{
  0,         /* LeftEdge, 0 pixels out. */
  -1,        /* TopEdge, 1 line up. */
  12,        /* Width, 12 pixels wide. */
  10,        /* Height, 10 lines heigh. */
  1,         /* Depth, one Bitplane. */
  spot_data, /* ImageData, pointer to the image data. */
  0x2,       /* PlanePick, affect the Bitplane one. */
  0x1,       /* PlaneOnOff, fill Bitplane zero with 1's. */
  NULL,      /* NextImage, no Image structure connected to this one. */
};
```

```
/************************************************************************/
/*                          F O U R T H   I T E M                       */
/************************************************************************/

/* The text for the fourth item: */
struct IntuiText my_fourth_text=
{
  2,            /* FrontPen, black. */
  0,            /* BackPen, not used since JAM1. */
  JAM1,         /* DrawMode, do not change the background. */
  CHECKWIDTH,   /* LeftEdge, CHECKWIDTH amount of pixels out. */
                /* This will leave enough space for the check mark. */
  1,            /* TopEdge, 1 line down. */
  NULL,         /* TextAttr, default font. */
  "Italic",     /* IText, the string. */
  NULL          /* NextItem, no link to other IntuiText structures. */
};

/* The MenuItem structure for the fourth item: */
struct MenuItem my_fourth_item=
{
  NULL,            /* NextItem, this is the last item in the list. */
  0,               /* LeftEdge, 0 pixels out. */
  30,              /* TopEdge, 30 lines down. */
  150,             /* Width, 150 pixels wide. */
  10,              /* Height, 10 lines high. */
  ITEMTEXT|        /* Flags, render this item with text. */
  ITEMENABLED|     /*        this item will be enabled. */
  CHECKIT|         /*        it is an attribute item. */
  HIGHCOMP,        /*        complement the colours when highlihted. */
  0x00000001,      /* MutualExclude, mutualexclude the first item only. */
  (APTR) &my_fourth_text, /* ItemFill, pointer to the text. */
  NULL,            /* SelectFill, nothing since we complement the col. */
  0,               /* Command, no command-key sequence. */
  NULL,            /* SubItem, no subitem list. */
  MENUNULL,        /* NextSelect, no items selected. */
};



/************************************************************************/
/*                           T H I R D   I T E M                        */
/************************************************************************/

/* The text for the third item: */
struct IntuiText my_third_text=
{
  2,            /* FrontPen, black. */
  0,            /* BackPen, not used since JAM1. */
  JAM1,         /* DrawMode, do not change the background. */
  CHECKWIDTH,   /* LeftEdge, CHECKWIDTH amount of pixels out. */
                /* This will leave enough space for the check mark. */
  1,            /* TopEdge, 1 line down. */
  NULL,         /* TextAttr, default font. */
  "Underlined", /* IText, the string. */
  NULL          /* NextItem, no link to other IntuiText structures. */
};

/* The MenuItem structure for the third item: */
struct MenuItem my_third_item=
{
  &my_fourth_item, /* NextItem, linked to the fourth item. */
  0,               /* LeftEdge, 0 pixels out. */
  20,              /* TopEdge, 20 lines down. */
  150,             /* Width, 150 pixels wide. */
  10,              /* Height, 10 lines high. */
  ITEMTEXT|        /* Flags, render this item with text. */
  ITEMENABLED|     /*        this item will be enabled. */
  CHECKIT|         /*        it is an attribute item. */
  HIGHCOMP,        /*        complement the colours when highlihted. */
  0x00000001,      /* MutualExclude, mutualexclude the first item only. */
  (APTR) &my_third_text, /* ItemFill, pointer to the text. */
  NULL,            /* SelectFill, nothing since we complement the col. */
  0,               /* Command, no command-key sequence. */
  NULL,            /* SubItem, no subitem list. */
  MENUNULL,        /* NextSelect, no items selected. */
};



/**************************************************************************/
/*                         S E C O N D   I T E M                          */
/**************************************************************************/

/* The text for the second item: */
struct IntuiText my_second_text=
{
  2,          /* FrontPen, black. */
  0,          /* BackPen, not used since JAM1. */
  JAM1,       /* DrawMode, do not change the background. */
  CHECKWIDTH, /* LeftEdge, CHECKWIDTH amount of pixels out. */
              /* This will leave enough space for the check mark. */
  1,          /* TopEdge, 1 line down. */
  NULL,       /* TextAttr, default font. */
  "Bold",     /* IText, the string. */
  NULL        /* NextItem, no link to other IntuiText structures. */
};

/* The MenuItem structure for the second item: */
struct MenuItem my_second_item=
{
  &my_third_item,  /* NextItem, linked to the third item. */
  0,               /* LeftEdge, 0 pixels out. */
  10,              /* TopEdge, 10 lines down. */
  150,             /* Width, 150 pixels wide. */
  10,              /* Height, 10 lines high. */
  ITEMTEXT|        /* Flags, render this item with text. */
  ITEMENABLED|     /*        this item will be enabled. */
  CHECKIT|         /*        it is an attribute item. */
  HIGHCOMP,        /*        complement the colours when highlihted. */
  0x00000001,      /* MutualExclude, mutualexclude the first item only. */
  (APTR) &my_second_text, /* ItemFill, pointer to the text. */
  NULL,            /* SelectFill, nothing since we complement the col. */
  0,               /* Command, no command-key sequence. */
  NULL,            /* SubItem, no subitem list. */
  MENUNULL,        /* NextSelect, no items selected. */
};



/**************************************************************************/
/*                          F I R S T   I T E M                           */
```

```
/**************************************************************************/

/* The text for the first item: */
struct IntuiText my_first_text=
{
  2,          /* FrontPen, black. */
  0,          /* BackPen, not used since JAM1. */
  JAM1,       /* DrawMode, do not change the background. */
  CHECKWIDTH, /* LeftEdge, CHECKWIDTH amount of pixels out. */
              /* This will leave enough space for the check mark. */
  1,          /* TopEdge, 1 line down. */
  NULL,       /* TextAttr, default font. */
  "Plain",    /* IText, the string. */
  NULL        /* NextItem, no link to other IntuiText structures. */
};

/* The MenuItem structure for the first item: */
struct MenuItem my_first_item=
{
  &my_second_item, /* NextItem, linked to the second item. */
  0,               /* LeftEdge, 0 pixels out. */
  0,               /* TopEdge, 0 lines down. */
  150,             /* Width, 150 pixels wide. */
  10,              /* Height, 10 lines high. */
  ITEMTEXT|        /* Flags, render this item with text. */
  ITEMENABLED|     /*        this item will be enabled. */
  CHECKIT|         /*        it is an attribute item. */
  CHECKED|         /*        this item is initially selected. */
  HIGHCOMP,        /*        complement the colours when highlihted. */
  0xFFFFFFFE,      /* MutualExclude, mutualexclude all items except the */
                   /*                first one. */
  (APTR) &my_first_text, /* ItemFill, pointer to the text. */
  NULL,            /* SelectFill, nothing since we complement the col. */
  0,               /* Command, no command-key sequence. */
  NULL,            /* SubItem, no subitem list. */
  MENUNULL,        /* NextSelect, no items selected. */
};



/**************************************************************************/
/*                               M E N U                                  */
/**************************************************************************/

/* The Menu structure for the first (and only) menu: */
struct Menu my_menu=
{
  NULL,           /* NextMenu, no more menu structures. */
  0,              /* LeftEdge, left corner. */
  0,              /* TopEdge, for the moment ignored by Intuition. */
  50,             /* Width, 50 pixels wide. */
  0,              /* Height, for the moment ignored by Intuition. */
  MENUENABLED,    /* Flags, this menu will be enabled. */
  "Mode",         /* MenuName, the string. */
  &my_first_item  /* FirstItem, pointer to the first item in the list. */
};



/* Declare a pointer to a Window structure: */
struct Window *my_window;
```

```
/* Declare and initialize your NewWindow structure: */
struct NewWindow my_new_window=
{
  50,            /* LeftEdge    x position of the window. */
  25,            /* TopEdge     y positio of the window. */
  200,           /* Width       200 pixels wide. */
  100,           /* Height      100 lines high. */
  0,             /* DetailPen   Text should be drawn with colour reg. 0 */
  1,             /* BlockPen    Blocks should be drawn with colour reg. 1 */
  CLOSEWINDOW|   /* IDCMPFlags  The window will give us a message if the */
                 /*             user has selected the Close window gad. */
  MENUPICK,
  SMART_REFRESH| /* Flags       Intuition should refresh the window. */
  WINDOWCLOSE|   /*             Close Gadget. */
  WINDOWDRAG|    /*             Drag gadget. */
  WINDOWDEPTH|   /*             Depth arrange Gadgets. */
  WINDOWSIZING|  /*             Sizing Gadget. */
  ACTIVATE,      /*             The window should be Active when opened. */
  NULL,          /* FirstGadget No Custom gadgets. */

  &spot,         /* CheckMark   Use our own customized checkmark. */

  "Style Editor",/* Title       Title of the window. */
  NULL,          /* Screen      Connected to the Workbench Screen. */
  NULL,          /* BitMap      No Custom BitMap. */
  80,            /* MinWidth    We will not allow the window to become */
  30,            /* MinHeight   smaller than 80 x 30, and not bigger */
  300,           /* MaxWidth    than 300 x 200. */
  200,           /* MaxHeight */
  WBENCHSCREEN   /* Type        Connected to the Workbench Screen. */
};



main()
{
  /* Boolean variable used for the while loop: */
  BOOL close_me;

  /* Declare a variable in which we will store the IDCMP flag: */
  ULONG class;

  /* If we recieve a MENUPICK event, the Code field of the message */
  /* structure will contain the menu number of the first selected item. */
  /* Declare a variable to store the Code value in, and an extra menu */
  /* number variable: */
  USHORT code, menu_number;

  /* Declare a MenuItem pointer: */
  struct MenuItem *item;

  /* Declare a pointer to an IntuiMessage structure: */
  struct IntuiMessage *my_message;



  /* Before we can use Intuition we need to open the Intuition Library: */
  IntuitionBase = (struct IntuitionBase *)
    OpenLibrary( "intuition.library", 0 );

  if( IntuitionBase == NULL )
    exit(); /* Could NOT open the Intuition Library! */
```

```
/* We will now try to open the window: */
my_window = (struct Window *) OpenWindow( &my_new_window );

/* Have we opened the window succesfully? */
if(my_window == NULL)
{
  /* Could NOT open the Window! */

  /* Close the Intuition Library since we have opened it: */
  CloseLibrary( IntuitionBase );

  exit();
}



/* We have opened the window, and everything seems to be OK. */



SetMenuStrip( my_window, &my_menu );
printf("Menustrip connected to window!\n");


close_me = FALSE;

/* Stay in the while loop until the user has selected the Close window */
/* gadget: */
while( close_me == FALSE )
{
  /* Wait until we have recieved a message: */
  Wait( 1 << my_window->UserPort->mp_SigBit );

  /* As long as we collect messages sucessfully we stay in the loop: */
  while(my_message=(struct IntuiMessage *) GetMsg( my_window->UserPort ))
  {
    /* After we have collected the message we can read it, and save any */
    /* important values which we maybe want to check later: */
    class = my_message->Class;
    code = my_message->Code;


    /* After we have read it we reply as fast as possible: */
    /* REMEMBER! Do never try to read a message after you have replied! */
    /* Some other process has maybe changed it. */
    ReplyMsg( my_message );

    /* Check which IDCMP flag was sent: */
    if( class == CLOSEWINDOW )
      close_me=TRUE; /* The user selected the Close window gadget! */

    if(class == MENUPICK)
    {
      printf("\nMenu pick!\n");
      menu_number = code;

      while( menu_number != MENUNULL )
      {
        /* Get the address of the item: */
        item = (struct MenuItem *) ItemAddress( &my_menu, menu_number );


        /* Print out the menu number plus etc: */
        printf("menu_number= %d\n", menu_number );
        printf("MENUNUM = %d\n", MENUNUM(menu_number) );
        printf("ITEMNUM = %d\n", ITEMNUM(menu_number) );
        printf("SUBNUM  = %d\n", SUBNUM(menu_number) );


        /* Get the following item's menu number: */
        menu_number = item->NextSelect;
      }
    }
  }
}



printf("Menustrip removed from window!\n");
ClearMenuStrip( my_window );



/* Close the window: */
CloseWindow( my_window );



/* Close the Intuition Library since we have opened it: */
CloseLibrary( IntuitionBase );

/* THE END */
}
```

Example6

This program opens a normal window to which we connect a menu strip. The menu consists of six small dices which are all action items. This example shows how you can use Images inside a menu.

```
/* Example6                                                               */
/* This program opens a normal window to which we connect a menu strip.  */
/* The menu consists of six small dices which are all action items. This */
/* example shows how you can use Images inside a menu.                   */



#include <intuition/intuition.h>



struct IntuitionBase *IntuitionBase;



/*************************************************************************/
/*                              D I C E   1                              */
/*************************************************************************/

/* Data for dice 1: */
USHORT chip dice1_data[]=
{
  0x0000,  /* 0000 0000 0000 0000   0 : Black  */
  0x0000,  /* 0000 0000 0000 0000   1 : Orange */
  0x0000,  /* 0000 0000 0000 0000              */
  0x03C0,  /* 0000 0011 1100 0000              */
  0x03C0,  /* 0000 0011 1100 0000              */
  0x0000,  /* 0000 0000 0000 0000              */
  0x0000,  /* 0000 0000 0000 0000              */
  0x0000   /* 0000 0000 0000 0000              */
};

/* Image structure for dice 1: */
struct Image dice1=
{
  0,          /* LeftEdge, 0 pixels out. */
  0,          /* TopEdge, 0 pixels down. */
  16,         /* Width, 16 pixels wide. */
  8,          /* Height, 8 lines heigh. */
  1,          /* Depth, one Bitplane. */
  dice1_data, /* ImageData, pointer to the image data. */
  0x1,        /* PlanePick, affect Bitplane zero. */
  0x2,        /* PlaneOnOff, fill Bitplane one with 1's. */
  NULL        /* NextImage, no Image structure connected to this one. */
};

/* The MenuItem structure for dice 1: */
struct MenuItem my_dice1_item=
{
  NULL,             /* NextItem, this is the last item in the list. */
  0,                /* LeftEdge, 0 pixels out. */
  50,               /* TopEdge, 50 lines down. */
  50,               /* Width, 50 pixels wide. */
  8,                /* Height, 8 lines high. */
  ITEMENABLED|      /* Flags, this item will be enabled. */
                    /*        render this item with an Image. */
                    /*        (ITEMTEXT is not set.) */
                    /*        it is an action item. */
                    /*        (CHECKIT is not set.) */
  HIGHCOMP,         /*        complement the colours when highlihted. */
  0x00000000,       /* MutualExclude, no mutualexclude. */
  (APTR) &dice1,    /* ItemFill, pointer to the image. */
  NULL,             /* SelectFill, nothing since we complement the col. */
  0,                /* Command, no command-key sequence. */
  NULL,             /* SubItem, no subitem list. */
  MENUNULL,         /* NextSelect, no items selected. */
};



/*************************************************************************/
/*                              D I C E   2                              */
/*************************************************************************/

/* Data for dice 2: */
USHORT chip dice2_data[]=
{
  0x0000,  /* 0000 0000 0000 0000   0 : Black  */
  0x7800,  /* 0111 1000 0000 0000   1 : Orange */
  0x7800,  /* 0111 1000 0000 0000              */
  0x0000,  /* 0000 0000 0000 0000              */
  0x0000,  /* 0000 0000 0000 0000              */
  0x001E,  /* 0000 0000 0001 1110              */
  0x001E,  /* 0000 0000 0001 1110              */
  0x0000   /* 0000 0000 0000 0000              */
};

/* Image structure for dice 2: */
struct Image dice2=
{
  0,          /* LeftEdge, 0 pixels out. */
  0,          /* TopEdge, 0 pixels down. */
  16,         /* Width, 16 pixels wide. */
  8,          /* Height, 8 lines heigh. */
  1,          /* Depth, one Bitplane. */
  dice2_data, /* ImageData, pointer to the image data. */
  0x1,        /* PlanePick, affect Bitplane zero. */
  0x2,        /* PlaneOnOff, fill Bitplane one with 1's. */
  NULL        /* NextImage, no Image structure connected to this one. */
};

/* The MenuItem structure for dice 2: */
struct MenuItem my_dice2_item=
{
  &my_dice1_item,  /* NextItem, pointer to the next item in the list. */
  0,               /* LeftEdge, 0 pixels out. */
  40,              /* TopEdge, 40 lines down. */
  50,              /* Width, 50 pixels wide. */
  8,               /* Height, 8 lines high. */
  ITEMENABLED|     /* Flags, this item will be enabled. */
                   /*        render this item with an Image. */
                   /*        (ITEMTEXT is not set.) */
                   /*        it is an action item. */
                   /*        (CHECKIT is not set.) */
  HIGHCOMP,        /*        complement the colours when highlihted. */
  0x00000000,      /* MutualExclude, no mutualexclude. */
  (APTR) &dice2,   /* ItemFill, pointer to the image. */
  NULL,            /* SelectFill, nothing since we complement the col. */
  0,               /* Command, no command-key sequence. */
  NULL,            /* SubItem, no subitem list. */
  MENUNULL,        /* NextSelect, no items selected. */
};
```

```
/*************************************************************************/
/*                               D I C E   3                              */
/*************************************************************************/

/* Data for dice 3: */
USHORT chip dice3_data[]=
{
  0x0000,  /* 0000 0000 0000 0000   0 : Black  */
  0x7800,  /* 0111 1000 0000 0000   1 : Orange */
  0x7800,  /* 0111 1000 0000 0000              */
  0x03C0,  /* 0000 0011 1100 0000              */
  0x03C0,  /* 0000 0011 1100 0000              */
  0x001E,  /* 0000 0000 0001 1110              */
  0x001E,  /* 0000 0000 0001 1110              */
  0x0000   /* 0000 0000 0000 0000              */
};

/* Image structure for dice 3: */
struct Image dice3=
{
  0,          /* LeftEdge, 0 pixels out. */
  0,          /* TopEdge, 0 pixels down. */
  16,         /* Width, 16 pixels wide. */
  8,          /* Height, 8 lines heigh. */
  1,          /* Depth, one Bitplane. */
  dice3_data, /* ImageData, pointer to the image data. */
  0x1,        /* PlanePick, affect Bitplane zero. */
  0x2,        /* PlaneOnOff, fill Bitplane one with 1's. */
  NULL        /* NextImage, no Image structure connected to this one. */
};

/* The MenuItem structure for dice 3: */
struct MenuItem my_dice3_item=
{
  &my_dice2_item,  /* NextItem, pointer to the next item in the list. */
  0,               /* LeftEdge, 0 pixels out. */
  30,              /* TopEdge, 30 lines down. */
  50,              /* Width, 50 pixels wide. */
  8,               /* Height, 8 lines high. */
  ITEMENABLED|     /* Flags, this item will be enabled. */
                   /*        render this item with an Image. */
                   /*        (ITEMTEXT is not set.) */
                   /*        it is an action item. */
                   /*        (CHECKIT is not set.) */
  HIGHCOMP,        /*        complement the colours when highlihted. */
  0x00000000,      /* MutualExclude, no mutualexclude. */
  (APTR) &dice3,   /* ItemFill, pointer to the image. */
  NULL,            /* SelectFill, nothing since we complement the col. */
  0,               /* Command, no command-key sequence. */
  NULL,            /* SubItem, no subitem list. */
  MENUNULL,        /* NextSelect, no items selected. */
};



/*************************************************************************/
/*                               D I C E   4                              */
/*************************************************************************/

/* Data for dice 4: */
USHORT chip dice4_data[]=
{
  0x0000,  /* 0000 0000 0000 0000   0 : Black  */
  0x781E,  /* 0111 1000 0001 1110   1 : Orange */
  0x781E,  /* 0111 1000 0001 1110              */
  0x0000,  /* 0000 0000 0000 0000              */
  0x0000,  /* 0000 0000 0000 0000              */
  0x781E,  /* 0111 1000 0001 1110              */
  0x781E,  /* 0111 1000 0001 1110              */
  0x0000   /* 0000 0000 0000 0000              */
};

/* Image structure for dice 4: */
struct Image dice4=
{
  0,          /* LeftEdge, 0 pixels out. */
  0,          /* TopEdge, 0 pixels down. */
  16,         /* Width, 16 pixels wide. */
  8,          /* Height, 8 lines heigh. */
  1,          /* Depth, one Bitplane. */
  dice4_data, /* ImageData, pointer to the image data. */
  0x1,        /* PlanePick, affect Bitplane zero. */
  0x2,        /* PlaneOnOff, fill Bitplane one with 1's. */
  NULL        /* NextImage, no Image structure connected to this one. */
};

/* The MenuItem structure for dice 4: */
struct MenuItem my_dice4_item=
{
  &my_dice3_item,  /* NextItem, pointer to the next item in the list. */
  0,               /* LeftEdge, 0 pixels out. */
  20,              /* TopEdge, 20 lines down. */
  50,              /* Width, 50 pixels wide. */
  8,               /* Height, 8 lines high. */
  ITEMENABLED|     /* Flags, this item will be enabled. */
                   /*        render this item with an Image. */
                   /*        (ITEMTEXT is not set.) */
                   /*        it is an action item. */
                   /*        (CHECKIT is not set.) */
  HIGHCOMP,        /*        complement the colours when highlihted. */
  0x00000000,      /* MutualExclude, no mutualexclude. */
  (APTR) &dice4,   /* ItemFill, pointer to the image. */
  NULL,            /* SelectFill, nothing since we complement the col. */
  0,               /* Command, no command-key sequence. */
  NULL,            /* SubItem, no subitem list. */
  MENUNULL,        /* NextSelect, no items selected. */
};



/*************************************************************************/
/*                               D I C E   5                              */
/*************************************************************************/

/* Data for dice 5: */
USHORT chip dice5_data[]=
{
  0x0000,  /* 0000 0000 0000 0000   0 : Black  */
  0x781E,  /* 0111 1000 0001 1110   1 : Orange */
  0x781E,  /* 0111 1000 0001 1110              */
  0x03C0,  /* 0000 0011 1100 0000              */
  0x03C0,  /* 0000 0011 1100 0000              */
  0x781E,  /* 0111 1000 0001 1110              */
  0x781E,  /* 0111 1000 0001 1110              */
  0x0000   /* 0000 0000 0000 0000              */
};

/* Image structure for dice 5: */
struct Image dice5=
{
  0,          /* LeftEdge, 0 pixels out. */
  0,          /* TopEdge, 0 pixels down. */
  16,         /* Width, 16 pixels wide. */
  8,          /* Height, 8 lines heigh. */
  1,          /* Depth, one Bitplane. */
  dice5_data, /* ImageData, pointer to the image data. */
  0x1,        /* PlanePick, affect Bitplane zero. */
  0x2,        /* PlaneOnOff, fill Bitplane one with 1's. */
  NULL        /* NextImage, no Image structure connected to this one. */
};

/* The MenuItem structure for dice 5: */
struct MenuItem my_dice5_item=
{
  &my_dice4_item,  /* NextItem, pointer to the next item in the list. */
  0,               /* LeftEdge, 0 pixels out. */
  10,              /* TopEdge, 10 lines down. */
  50,              /* Width, 50 pixels wide. */
  8,               /* Height, 8 lines high. */
  ITEMENABLED|     /* Flags, this item will be enabled. */
                   /*        render this item with an Image. */
                   /*        (ITEMTEXT is not set.) */
                   /*        it is an action item. */
                   /*        (CHECKIT is not set.) */
  HIGHCOMP,        /*        complement the colours when highlihted. */
  0x00000000,      /* MutualExclude, no mutualexclude. */
  (APTR) &dice5,   /* ItemFill, pointer to the image. */
  NULL,            /* SelectFill, nothing since we complement the col. */
  0,               /* Command, no command-key sequence. */
  NULL,            /* SubItem, no subitem list. */
  MENUNULL,        /* NextSelect, no items selected. */
};



/***********************************************************************/
/*                              D I C E   6                            */
/***********************************************************************/

/* Data for dice 6: */
USHORT chip dice6_data[]=
{
  0x0000,  /* 0000 0000 0000 0000   0 : Black  */
  0x7BDE,  /* 0111 1011 1101 1110   1 : Orange */
  0x7BDE,  /* 0111 1011 1101 1110              */
  0x0000,  /* 0000 0000 0000 0000              */
  0x0000,  /* 0000 0000 0000 0000              */
  0x7BDE,  /* 0111 1011 1101 1110              */
  0x7BDE,  /* 0111 1011 1101 1110              */
  0x0000   /* 0000 0000 0000 0000              */
};

/* Image structure for dice 6: */
struct Image dice6=
{
  0,          /* LeftEdge, 0 pixels out. */
  0,          /* TopEdge, 0 pixels down. */
  16,         /* Width, 16 pixels wide. */
  8,          /* Height, 8 lines heigh. */
  1,          /* Depth, one Bitplane. */
  dice6_data, /* ImageData, pointer to the image data. */
  0x1,        /* PlanePick, affect Bitplane zero. */
  0x2,        /* PlaneOnOff, fill Bitplane one with 1's. */
  NULL        /* NextImage, no Image structure connected to this one. */
};

/* The MenuItem structure for dice 6: */
struct MenuItem my_dice6_item=
{
  &my_dice5_item,  /* NextItem, pointer to the next item in the list. */
  0,               /* LeftEdge, 0 pixels out. */
  0,               /* TopEdge, 0 lines down. */
  50,              /* Width, 50 pixels wide. */
  8,               /* Height, 8 lines high. */
  ITEMENABLED|     /* Flags, this item will be enabled. */
                   /*        render this item with an Image. */
                   /*        (ITEMTEXT is not set.) */
                   /*        it is an action item. */
                   /*        (CHECKIT is not set.) */
  HIGHCOMP,        /*        complement the colours when highlihted. */
  0x00000000,      /* MutualExclude, no mutualexclude. */
  (APTR) &dice6,   /* ItemFill, pointer to the image. */
  NULL,            /* SelectFill, nothing since we complement the col. */
  0,               /* Command, no command-key sequence. */
  NULL,            /* SubItem, no subitem list. */
  MENUNULL,        /* NextSelect, no items selected. */
};



/***********************************************************************/
/*                              M E N U                                */
/***********************************************************************/

/* The Menu structure for the first (and only) menu: */
struct Menu my_menu=
{
  NULL,           /* NextMenu, no more menu structures. */
  0,              /* LeftEdge, left corner. */
  0,              /* TopEdge, for the moment ignored by Intuition. */
  50,             /* Width, 50 pixels wide. */
  0,              /* Height, for the moment ignored by Intuition. */
  MENUENABLED,    /* Flags, this menu will be enabled. */
  "Dice",         /* MenuName, the string. */
  &my_dice6_item /* FirstItem, pointer to the first item in the list. */
};



/* Declare a pointer to a Window structure: */
struct Window *my_window;

/* Declare and initialize your NewWindow structure: */
struct NewWindow my_new_window=
{
  50,             /* LeftEdge    x position of the window. */
  25,             /* TopEdge     y positio of the window. */
  200,           /* Width       200 pixels wide. */
  100,           /* Height      100 lines high. */
  0,             /* DetailPen   Text should be drawn with colour reg. 0 */
  1,             /* BlockPen    Blocks should be drawn with colour reg. 1 */
  CLOSEWINDOW|   /* IDCMPFlags  The window will give us a message if the */
                 /*             user has selected the Close window gad. */
  MENUPICK,
  SMART_REFRESH| /* Flags       Intuition should refresh the window. */
  WINDOWCLOSE|   /*             Close Gadget. */
  WINDOWDRAG|    /*             Drag gadget. */
  WINDOWDEPTH|   /*             Depth arrange Gadgets. */
  WINDOWSIZING|  /*             Sizing Gadget. */
  ACTIVATE,      /*             The window should be Active when opened. */
  NULL,          /* FirstGadget No Custom gadgets. */
  NULL,          /* CheckMark   Use Intuition's default checkmark. */
  "GAME",        /* Title       Title of the window. */
  NULL,          /* Screen      Connected to the Workbench Screen. */
  NULL,          /* BitMap      No Custom BitMap. */
  80,            /* MinWidth    We will not allow the window to become */
  30,            /* MinHeight   smaller than 80 x 30, and not bigger */
  300,           /* MaxWidth    than 300 x 200. */
  200,           /* MaxHeight */
  WBENCHSCREEN   /* Type        Connected to the Workbench Screen. */
};



main()
{
  /* Boolean variable used for the while loop: */
  BOOL close_me;

  /* Declare a variable in which we will store the IDCMP flag: */
  ULONG class;

  /* If we recieve a MENUPICK event, the Code field of the message */
  /* structure will contain the menu number of the first selected item. */
  /* Declare a variable to store the Code value in, and an extra menu */
  /* number variable: */
  USHORT code, menu_number;

  /* Declare a MenuItem pointer: */
  struct MenuItem *item;

  /* Declare a pointer to an IntuiMessage structure: */
  struct IntuiMessage *my_message;



  /* Before we can use Intuition we need to open the Intuition Library: */
  IntuitionBase = (struct IntuitionBase *)
    OpenLibrary( "intuition.library", 0 );

  if( IntuitionBase == NULL )
    exit(); /* Could NOT open the Intuition Library! */



  /* We will now try to open the window: */
  my_window = (struct Window *) OpenWindow( &my_new_window );

  /* Have we opened the window succesfully? */
```

```
if(my_window == NULL)
{
  /* Could NOT open the Window! */

  /* Close the Intuition Library since we have opened it: */
  CloseLibrary( IntuitionBase );

  exit();
}



/* We have opened the window, and everything seems to be OK. */



SetMenuStrip( my_window, &my_menu );
printf("Menustrip connected to window!\n");


close_me = FALSE;

/* Stay in the while loop until the user has selected the Close window */
/* gadget: */
while( close_me == FALSE )
{
  /* Wait until we have recieved a message: */
  Wait( 1 << my_window->UserPort->mp_SigBit );

  /* As long as we collect messages sucessfully we stay in the loop: */
  while(my_message=(struct IntuiMessage *) GetMsg( my_window->UserPort ))
  {
    /* After we have collected the message we can read it, and save any */
    /* important values which we maybe want to check later: */
    class = my_message->Class;
    code = my_message->Code;


    /* After we have read it we reply as fast as possible: */
    /* REMEMBER! Do never try to read a message after you have replied! */
    /* Some other process has maybe changed it. */
    ReplyMsg( my_message );

    /* Check which IDCMP flag was sent: */
    if( class == CLOSEWINDOW )
      close_me=TRUE; /* The user selected the Close window gadget! */

    if(class == MENUPICK)
    {
      printf("\nMenu pick!\n");
      menu_number = code;

      while( menu_number != MENUNULL )
      {
        /* Get the address of the item: */
        item = (struct MenuItem *) ItemAddress( &my_menu, menu_number );


        /* Print out the menu number plus etc: */
        printf("menu_number= %d\n", menu_number );
        printf("MENUNUM = %d\n", MENUNUM(menu_number) );
        printf("ITEMNUM = %d\n", ITEMNUM(menu_number) );
        printf("SUBNUM  = %d\n", SUBNUM(menu_number) );


        /* Get the following item's menu number: */
        menu_number = item->NextSelect;
      }
    }
  }
}



printf("Menustrip removed from window!\n");
ClearMenuStrip( my_window );



/* Close the window: */
CloseWindow( my_window );



/* Close the Intuition Library since we have opened it: */
CloseLibrary( IntuitionBase );

/* THE END */
}
```

Example7

This program opens a normal window to which we connect a menu strip. The menu consists of one small action item with two images.

```
/* Example7                                                              */
/* This program opens a normal window to which we connect a menu strip.  */
/* The menu consists of one small action item with two images.           */



#include <intuition/intuition.h>



struct IntuitionBase *IntuitionBase;



/*************************************************************************/
/*                         F A C E   S L E E P I N G                     */
/*************************************************************************/

USHORT chip face_sleeping_data[]=
{
  0x7FFF,0xFC00, /* Bitplane ZERO */
  0xFE10,0xFE00,
  0xF9D7,0x3E00,
  0xE7D7,0xCE00,
  0xDFD7,0xF600,
  0xDF93,0xF600,
  0xC054,0x0600,
  0xDFD7,0xF600,
  0xC010,0x0600,
  0xFFFF,0xFE00,
  0x7FFF,0xFC00,

  0x7FFF,0xFC00, /* Bitplane ONE */
  0xFFFF,0xFE00,
  0xFFFF,0xFE00,
  0xFFFF,0xFE00,
  0xFFFF,0xFE00,
  0xFFFF,0xFE00,
  0xFFBB,0xFE00,
  0xE038,0x0E00,
  0xFFFF,0xFE00,
  0xFFFF,0xFE00,
  0x7FFF,0xFC00
};

/* Image structure for the sleeping face: */
struct Image face_sleeping=
{
  0,           /* LeftEdge, 0 pixels out. */
  0,           /* TopEdge, 0 pixels down. */
  23,          /* Width, 23 pixels wide. */
  11,          /* Height, 11 lines heigh. */
  2,           /* Depth, two Bitplanes. */
  face_sleeping_data, /* ImageData, pointer to the image data. */
  0x3,         /* PlanePick, affect Bitplane zero and one. */
  0x0,         /* PlaneOnOff, do not bother about any Bitplanes. */
  NULL         /* NextImage, no Image structure connected to this one. */
};



/*************************************************************************/
```

```
/*                          F A C E   A W A K E                          */
/*************************************************************************/

USHORT chip face_awake_data[]=
{
  0x7FFF,0xFC00, /* Bitplane ZERO */
  0xFE10,0xFE00,
  0xF9D7,0x3E00,
  0xE7D7,0xCE00,
  0xDFD7,0xF600,
  0xDE10,0xF600,
  0xDC10,0x7600,
  0xDC10,0x7600,
  0xC010,0x0600,
  0xFFFF,0xFE00,
  0x7FFF,0xFC00,

  0x7FFF,0xFC00, /* Bitplane ONE */
  0xFFFF,0xFE00,
  0xFE38,0xFE00,
  0xF838,0x3E00,
  0xE038,0x0E00,
  0xE1FF,0x0E00,
  0xE3FF,0x8E00,
  0xE3FF,0x8E00,
  0xFFFF,0xFE00,
  0xFFFF,0xFE00,
  0x7FFF,0xFC00
};

/* Image structure for the awake face: */
struct Image face_awake=
{
  0,           /* LeftEdge, 0 pixels out. */
  0,           /* TopEdge, 0 pixels down. */
  23,          /* Width, 23 pixels wide. */
  11,          /* Height, 11 lines heigh. */
  2,           /* Depth, two Bitplanes. */
  face_awake_data, /* ImageData, pointer to the image data. */
  0x3,         /* PlanePick, affect Bitplane zero and one. */
  0x0,         /* PlaneOnOff, do not bother about any Bitplanes. */
  NULL         /* NextImage, no Image structure connected to this one. */
};



/*************************************************************************/
/*                            M E N U   I T E M                          */
/*************************************************************************/

/* The one and only MenuItem structure: */
struct MenuItem my_item=
{
  NULL,             /* NextItem, this is the one and only item. */
  0,                /* LeftEdge, 0 pixels out. */
  0,                /* TopEdge, 0 lines down. */
  50,               /* Width, 50 pixels wide. */
  11,               /* Height, 11 lines high. */
  ITEMENABLED|      /* Flags, this item will be enabled. */
                    /*        render this item with an Image. */
                    /*        (ITEMTEXT is not set.) */
                    /*        it is an action item. */
```

```
                  /*        (CHECKIT is not set.) */
  HIGHIMAGE,      /*        display an alternative Image when highl. */
  0x00000000,     /* MutualExclude, no mutualexclude. */
  (APTR) &face_sleeping, /* ItemFill, pointer to the image. */
  (APTR) &face_awake,    /* SelectFill, pointer to the alternative image. */
  0,              /* Command, no command-key sequence. */
  NULL,           /* SubItem, no subitem list. */
  MENUNULL,       /* NextSelect, no items selected. */
};



/***********************************************************************/
/*                              M E N U                                */
/***********************************************************************/

/* The Menu structure for the first (and only) menu: */
struct Menu my_menu=
{
  NULL,        /* NextMenu, no more menu structures. */
  0,           /* LeftEdge, left corner. */
  0,           /* TopEdge, for the moment ignored by Intuition. */
  50,          /* Width, 50 pixels wide. */
  0,           /* Height, for the moment ignored by Intuition. */
  MENUENABLED, /* Flags, this menu will be enabled. */
  "Face",      /* MenuName, the string. */
  &my_item     /* FirstItem, pointer to the first item in the list. */
};



/* Declare a pointer to a Window structure: */
struct Window *my_window;

/* Declare and initialize your NewWindow structure: */
struct NewWindow my_new_window=
{
  50,            /* LeftEdge    x position of the window. */
  25,            /* TopEdge     y positio of the window. */
  200,           /* Width       200 pixels wide. */
  100,           /* Height      100 lines high. */
  0,             /* DetailPen   Text should be drawn with colour reg. 0 */
  1,             /* BlockPen    Blocks should be drawn with colour reg. 1 */
  CLOSEWINDOW|   /* IDCMPFlags  The window will give us a message if the */
                 /*             user has selected the Close window gad. */
  MENUPICK,
  SMART_REFRESH| /* Flags       Intuition should refresh the window. */
  WINDOWCLOSE|   /*             Close Gadget. */
  WINDOWDRAG|    /*             Drag gadget. */
  WINDOWDEPTH|   /*             Depth arrange Gadgets. */
  WINDOWSIZING|  /*             Sizing Gadget. */
  ACTIVATE,      /*             The window should be Active when opened. */
  NULL,          /* FirstGadget No Custom gadgets. */
  NULL,          /* CheckMark   Use Intuition's default checkmark. */
  "Person",      /* Title       Title of the window. */
  NULL,          /* Screen      Connected to the Workbench Screen. */
  NULL,          /* BitMap      No Custom BitMap. */
  80,            /* MinWidth    We will not allow the window to become */
  30,            /* MinHeight   smaller than 80 x 30, and not bigger */
  300,           /* MaxWidth    than 300 x 200. */
  200,           /* MaxHeight */
  WBENCHSCREEN   /* Type        Connected to the Workbench Screen. */
};



main()
{
  /* Boolean variable used for the while loop: */
  BOOL close_me;

  /* Declare a variable in which we will store the IDCMP flag: */
  ULONG class;

  /* If we recieve a MENUPICK event, the Code field of the message */
  /* structure will contain the menu number of the first selected item. */
  /* Declare a variable to store the Code value in, and an extra menu */
  /* number variable: */
  USHORT code, menu_number;

  /* Declare a MenuItem pointer: */
  struct MenuItem *item;

  /* Declare a pointer to an IntuiMessage structure: */
  struct IntuiMessage *my_message;



  /* Before we can use Intuition we need to open the Intuition Library: */
  IntuitionBase = (struct IntuitionBase *)
    OpenLibrary( "intuition.library", 0 );

  if( IntuitionBase == NULL )
    exit(); /* Could NOT open the Intuition Library! */



  /* We will now try to open the window: */
  my_window = (struct Window *) OpenWindow( &my_new_window );

  /* Have we opened the window succesfully? */
  if(my_window == NULL)
  {
    /* Could NOT open the Window! */

    /* Close the Intuition Library since we have opened it: */
    CloseLibrary( IntuitionBase );

    exit();
  }



  /* We have opened the window, and everything seems to be OK. */



  SetMenuStrip( my_window, &my_menu );
  printf("Menustrip connected to window!\n");


  close_me = FALSE;

  /* Stay in the while loop until the user has selected the Close window */
  /* gadget: */
  while( close_me == FALSE )
  {
    /* Wait until we have recieved a message: */
    Wait( 1 << my_window->UserPort->mp_SigBit );

    /* As long as we collect messages sucessfully we stay in the loop: */
    while(my_message=(struct IntuiMessage *) GetMsg( my_window->UserPort ))
    {
      /* After we have collected the message we can read it, and save any */
      /* important values which we maybe want to check later: */
      class = my_message->Class;
      code = my_message->Code;


      /* After we have read it we reply as fast as possible: */
      /* REMEMBER! Do never try to read a message after you have replied! */
      /* Some other process has maybe changed it. */
      ReplyMsg( my_message );

      /* Check which IDCMP flag was sent: */
      if( class == CLOSEWINDOW )
        close_me=TRUE; /* The user selected the Close window gadget! */

      if(class == MENUPICK)
      {
        printf("\nMenu pick!\n");
        menu_number = code;

        while( menu_number != MENUNULL )
        {
          /* Get the address of the item: */
          item = (struct MenuItem *) ItemAddress( &my_menu, menu_number );


          /* Print out the menu number plus etc: */
          printf("menu_number= %d\n", menu_number );
          printf("MENUNUM = %d\n", MENUNUM(menu_number) );
          printf("ITEMNUM = %d\n", ITEMNUM(menu_number) );
          printf("SUBNUM  = %d\n", SUBNUM(menu_number) );


          /* Get the following item's menu number: */
          menu_number = item->NextSelect;
        }
      }
    }
  }



  printf("Menustrip removed from window!\n");
  ClearMenuStrip( my_window );



  /* Close the window: */
  CloseWindow( my_window );



  /* Close the Intuition Library since we have opened it: */
  CloseLibrary( IntuitionBase );

  /* THE END */
}
```

Example8

Same as Example1 except that we this time will verify any menu operations. If the user tries to activate this program's menu we check if the position of the pointer is somewhere at the top of the window (less than 10 lines down). In that case the menu operation will continue as normal, otherwise we cancel the menu operation.

```
/* Example8                                                            */
/* Same as Example1 except that we this time will verify any menu      */
/* operations. If the user tries to activate this program's menu we    */
/* check if the position of the pointer, and if it is somewhere at the */
/* top of the window (less than 10 lines down) the menu operation will */
/* continue as normal, else we cancel the menu operation.              */



#include <intuition/intuition.h>



struct IntuitionBase *IntuitionBase;



/************************************************************************/
/*                         F O U R T H   I T E M                        */
/************************************************************************/

/* The text for the fourth item: */
struct IntuiText my_fourth_text=
{
  2,            /* FrontPen, black. */
  0,            /* BackPen, not used since JAM1. */
  JAM1,         /* DrawMode, do not change the background. */
  CHECKWIDTH,   /* LeftEdge, CHECKWIDTH amount of pixels out. */
                /* This will leave enough space for the check mark. */
  1,            /* TopEdge, 1 line down. */
  NULL,         /* TextAttr, default font. */
  "Italic",     /* IText, the string. */
  NULL          /* NextItem, no link to other IntuiText structures. */
};

/* The MenuItem structure for the fourth item: */
struct MenuItem my_fourth_item=
{
  NULL,            /* NextItem, this is the last item in the list. */
  0,               /* LeftEdge, 0 pixels out. */
  30,              /* TopEdge, 30 lines down. */
  150,             /* Width, 150 pixels wide. */
  10,              /* Height, 10 lines high. */
  ITEMTEXT|        /* Flags, render this item with text. */
  ITEMENABLED|     /*        this item will be enabled. */
  CHECKIT|         /*        it is an attribute item. */
  HIGHCOMP,        /*        complement the colours when highlihted. */
  0x00000001,      /* MutualExclude, mutualexclude the first item only. */
  (APTR) &my_fourth_text, /* ItemFill, pointer to the text. */
  NULL,            /* SelectFill, nothing since we complement the col. */
  0,               /* Command, no command-key sequence. */
  NULL,            /* SubItem, no subitem list. */
  MENUNULL,        /* NextSelect, no items selected. */
};



/************************************************************************/
/*                          T H I R D   I T E M                         */
/************************************************************************/

/* The text for the third item: */
```

```
struct IntuiText my_third_text=
{
  2,            /* FrontPen, black. */
  0,            /* BackPen, not used since JAM1. */
  JAM1,         /* DrawMode, do not change the background. */
  CHECKWIDTH,   /* LeftEdge, CHECKWIDTH amount of pixels out. */
                /* This will leave enough space for the check mark. */
  1,            /* TopEdge, 1 line down. */
  NULL,         /* TextAttr, default font. */
  "Underlined", /* IText, the string. */
  NULL          /* NextItem, no link to other IntuiText structures. */
};

/* The MenuItem structure for the third item: */
struct MenuItem my_third_item=
{
  &my_fourth_item, /* NextItem, linked to the fourth item. */
  0,               /* LeftEdge, 0 pixels out. */
  20,              /* TopEdge, 20 lines down. */
  150,             /* Width, 150 pixels wide. */
  10,              /* Height, 10 lines high. */
  ITEMTEXT|        /* Flags, render this item with text. */
  ITEMENABLED|     /*        this item will be enabled. */
  CHECKIT|         /*        it is an attribute item. */
  HIGHCOMP,        /*        complement the colours when highlihted. */
  0x00000001,      /* MutualExclude, mutualexclude the first item only. */
  (APTR) &my_third_text, /* ItemFill, pointer to the text. */
  NULL,            /* SelectFill, nothing since we complement the col. */
  0,               /* Command, no command-key sequence. */
  NULL,            /* SubItem, no subitem list. */
  MENUNULL,        /* NextSelect, no items selected. */
};



/************************************************************************/
/*                         S E C O N D   I T E M                        */
/************************************************************************/

/* The text for the second item: */
struct IntuiText my_second_text=
{
  2,          /* FrontPen, black. */
  0,          /* BackPen, not used since JAM1. */
  JAM1,       /* DrawMode, do not change the background. */
  CHECKWIDTH, /* LeftEdge, CHECKWIDTH amount of pixels out. */
              /* This will leave enough space for the check mark. */
  1,          /* TopEdge, 1 line down. */
  NULL,       /* TextAttr, default font. */
  "Bold",     /* IText, the string. */
  NULL        /* NextItem, no link to other IntuiText structures. */
};

/* The MenuItem structure for the second item: */
struct MenuItem my_second_item=
{
  &my_third_item,  /* NextItem, linked to the third item. */
  0,               /* LeftEdge, 0 pixels out. */
  10,              /* TopEdge, 10 lines down. */
  150,             /* Width, 150 pixels wide. */
  10,              /* Height, 10 lines high. */
  ITEMTEXT|        /* Flags, render this item with text. */
  ITEMENABLED|     /*        this item will be enabled. */
  CHECKIT|         /*        it is an attribute item. */
  HIGHCOMP,        /*        complement the colours when highlihted. */
  0x00000001,      /* MutualExclude, mutualexclude the first item only. */
  (APTR) &my_second_text, /* ItemFill, pointer to the text. */
  NULL,            /* SelectFill, nothing since we complement the col. */
  0,               /* Command, no command-key sequence. */
  NULL,            /* SubItem, no subitem list. */
  MENUNULL,        /* NextSelect, no items selected. */
};



/************************************************************************/
/*                          F I R S T   I T E M                         */
/************************************************************************/

/* The text for the first item: */
struct IntuiText my_first_text=
{
  2,          /* FrontPen, black. */
  0,          /* BackPen, not used since JAM1. */
  JAM1,       /* DrawMode, do not change the background. */
  CHECKWIDTH, /* LeftEdge, CHECKWIDTH amount of pixels out. */
              /* This will leave enough space for the check mark. */
  1,          /* TopEdge, 1 line down. */
  NULL,       /* TextAttr, default font. */
  "Plain",    /* IText, the string. */
  NULL        /* NextItem, no link to other IntuiText structures. */
};

/* The MenuItem structure for the first item: */
struct MenuItem my_first_item=
{
  &my_second_item, /* NextItem, linked to the second item. */
  0,               /* LeftEdge, 0 pixels out. */
  0,               /* TopEdge, 0 lines down. */
  150,             /* Width, 150 pixels wide. */
  10,              /* Height, 10 lines high. */
  ITEMTEXT|        /* Flags, render this item with text. */
  ITEMENABLED|     /*        this item will be enabled. */
  CHECKIT|         /*        it is an attribute item. */
  CHECKED|         /*        this item is initially selected. */
  HIGHCOMP,        /*        complement the colours when highlihted. */
  0xFFFFFFFE,      /* MutualExclude, mutualexclude all items except the */
                   /*                first one. */
  (APTR) &my_first_text, /* ItemFill, pointer to the text. */
  NULL,            /* SelectFill, nothing since we complement the col. */
  0,               /* Command, no command-key sequence. */
  NULL,            /* SubItem, no subitem list. */
  MENUNULL,        /* NextSelect, no items selected. */
};



/************************************************************************/
/*                              M E N U                                 */
/************************************************************************/

/* The Menu structure for the first (and only) menu: */
struct Menu my_menu=
{
  NULL,            /* NextMenu, no more menu structures. */
  0,               /* LeftEdge, left corner. */
  0,               /* TopEdge, for the moment ignored by Intuition. */
  50,              /* Width, 50 pixels wide. */
  0,               /* Height, for the moment ignored by Intuition. */
  MENUENABLED,     /* Flags, this menu will be enabled. */
  "Mode",          /* MenuName, the string. */
  &my_first_item /* FirstItem, pointer to the first item in the list. */
};



/* Declare a pointer to a Window structure: */
struct Window *my_window;

/* Declare and initialize your NewWindow structure: */
struct NewWindow my_new_window=
{
  0,               /* LeftEdge    x position of the window. */
  0,               /* TopEdge     y positio of the window. */
  320,             /* Width       320 pixels wide. */
  100,             /* Height      100 lines high. */
  0,               /* DetailPen   Text should be drawn with colour reg. 0 */
  1,               /* BlockPen    Blocks should be drawn with colour reg. 1 */
  CLOSEWINDOW|     /* IDCMPFlags  The window will give us a message if the */
                   /*             user has selected the Close window gad. */
  MENUPICK|        /*             Or if the user has done a menu operation, */
  MENUVERIFY,      /*             or if the user tries to activate a menu. */
  SMART_REFRESH|   /* Flags       Intuition should refresh the window. */
  WINDOWCLOSE|     /*             Close Gadget. */
  WINDOWDRAG|      /*             Drag gadget. */
  WINDOWDEPTH|     /*             Depth arrange Gadgets. */
  WINDOWSIZING|    /*             Sizing Gadget. */
  ACTIVATE,        /*             The window should be Active when opened. */
  NULL,            /* FirstGadget No Custom gadgets. */
  NULL,            /* CheckMark   Use Intuition's default CheckMark. */
  "Style Editor",/* Title       Title of the window. */
  NULL,            /* Screen      Connected to the Workbench Screen. */
  NULL,            /* BitMap      No Custom BitMap. */
  80,              /* MinWidth    We will not allow the window to become */
  30,              /* MinHeight   smaller than 80 x 30, and not bigger */
  300,             /* MaxWidth    than 300 x 200. */
  200,             /* MaxHeight */
  WBENCHSCREEN     /* Type        Connected to the Workbench Screen. */
};



main()
{
  /* Boolean variable used for the while loop: */
  BOOL close_me;

  /* Declare a variable in which we will store the IDCMP flag: */
  ULONG class;

  /* If we recieve a MENUPICK event, the Code field of the message */
  /* structure will contain the menu number of the first selected item. */
  /* Declare a variable to store the Code value in, and an extra menu */
  /* number variable: */
  USHORT code, menu_number;
```

```
/* Declare a MenuItem pointer: */
struct MenuItem *item;

/* Declare a pointer to an IntuiMessage structure: */
struct IntuiMessage *my_message;



/* Before we can use Intuition we need to open the Intuition Library: */
IntuitionBase = (struct IntuitionBase *)
  OpenLibrary( "intuition.library", 0 );

if( IntuitionBase == NULL )
  exit(); /* Could NOT open the Intuition Library! */



/* We will now try to open the window: */
my_window = (struct Window *) OpenWindow( &my_new_window );

/* Have we opened the window succesfully? */
if(my_window == NULL)
{
  /* Could NOT open the Window! */

  /* Close the Intuition Library since we have opened it: */
  CloseLibrary( IntuitionBase );

  exit();
}



/* We have opened the window, and everything seems to be OK. */



SetMenuStrip( my_window, &my_menu );
printf("Menustrip connected to window!\n");


close_me = FALSE;

/* Stay in the while loop until the user has selected the Close window */
/* gadget: */
while( close_me == FALSE )
{
  /* Wait until we have recieved a message: */
  Wait( 1 << my_window->UserPort->mp_SigBit );

  /* As long as we collect messages sucessfully we stay in the loop: */
  while(my_message=(struct IntuiMessage *) GetMsg( my_window->UserPort ))
  {
    /* After we have collected the message we can read it, and save any */
    /* important values which we maybe want to check later: */
    class = my_message->Class;
    code = my_message->Code;



    /* Before we reply we need to see if we have recieved a MENUVERIFY */
    /* message: */
```

```
    if( class == MENUVERIFY )
    {
      /* Yes, we have recieved a MENUVERIFY message!                   */
      /* The user wants to activate a menu, but the problem is that we  */
      /* do not know if it is our window's menu that will be activated, */
      /* or some other window's menu. We can however check it by        */
      /* examining the Code field of the message. If it is equal to     */
      /* MENUWAITING, it means that it is not your window's menu that   */
      /* will be activated, but if Code is equal to MENUHOT it means it */
      /* is is your window's menu that will be activated!               */

      if( code == MENUWAITING )
      {
        /* It is not your window's menu that will be activated! */

        /* Your program can take a pause if necessary. You maybe want */
        /* to finish of with some drawings, so your program does not  */
        /* trash any menus. This is especially important if you are   */
        /* using the low-level graphics rutines since they do not     */
        /* bother about, windows etc, and will draw over and destroy  */
        /* anything in their way.                                     */

        /* Once the program is ready it should reply the message, and */
        /* the menu will be activated.                                */

        printf("Another program's menu will be displayed!\n");
      }
      else
        if( code == MENUHOT )
        {
          /* It is your window's menu that will be activated! */

          /* You can now take a pause and finish of with something     */
          /* before you let Intuition activate the menu, or you can    */
          /* even stop the whole menu operation if necessary. If you   */
          /* are writing a paint program you maybe only want the user  */
          /* to be able to activate the menu if the pointer is at the  */
          /* top of the display. That would mean that the user can     */
          /* draw with the right mouse button, and when the user wants */
          /* to make a menu choice, he/she simply moves the pointer to */
          /* the top of the display, and then presses the right mouse  */
          /* button.                                                   */

          /* We will now check if the pointer is somewhere at the top  */
          /* of the display: */
          if( my_window->MouseY < 10)
          {
            /* The Y coordinate of the pointer is at least less than   */
            /* 10 lines below the TopEdge of the window.               */

            /* The menu operation should continue as soon as possible! */
            printf("OK!\n");
          }
          else
          {
            /* The pointer is below the Title bar of the window! */
            /* Cancel the whole menu operation! */

            /* To cancel a menu operation you need to change the Code   */
            /* field to MENUCANCEL. IMPORTANT! Do not change the code   */
            /* variable since it is just a copy of the real Code value. */
            /* What we need to do is to change the real value, and that */
            /* is still OK since we have not replied yet.               */

            my_message->Code=MENUCANCEL;

            printf("Menu operation canceled!\n");
          }
        }
    }

    /* After we have read it we reply as fast as possible: */
    /* REMEMBER! Do never try to read a message after you have replied! */
    /* Some other process has maybe changed it. */
    ReplyMsg( my_message );

    /* Check which IDCMP flag was sent: */
    if( class == CLOSEWINDOW )
      close_me=TRUE; /* The user selected the Close window gadget! */

    if(class == MENUPICK)
    {
      printf("\nMenu pick!\n");
      menu_number = code;

      while( menu_number != MENUNULL )
      {
        /* Get the address of the item: */
        item = (struct MenuItem *) ItemAddress( &my_menu, menu_number );


        /* Print out the menu number plus etc: */
        printf("menu_number= %d\n", menu_number );
        printf("MENUNUM = %d\n", MENUNUM(menu_number) );
        printf("ITEMNUM = %d\n", ITEMNUM(menu_number) );
        printf("SUBNUM  = %d\n", SUBNUM(menu_number) );


        /* Get the following item's menu number: */
        menu_number = item->NextSelect;
      }
    }
  }
}



printf("Menustrip removed from window!\n");
ClearMenuStrip( my_window );



/* Close the window: */
CloseWindow( my_window );



/* Close the Intuition Library since we have opened it: */
CloseLibrary( IntuitionBase );

/* THE END */
}
```

A.8 IDCMP

Example1

This program explains how to use the IDCMP flag MOUSEBUTTONS.

```c
/* Example1                                         */
/* This example shows how to handle MOUSEBUTTONS event. */



#include <intuition/intuition.h>



struct IntuitionBase *IntuitionBase;



/* Declare a pointer to a Window structure: */
struct Window *my_window;

/* Declare and initialize your NewWindow structure: */
struct NewWindow my_new_window=
{
  50,            /* LeftEdge    x position of the window. */
  25,            /* TopEdge     y positio of the window. */
  320,           /* Width       320 pixels wide. */
  100,           /* Height      100 lines high. */
  0,             /* DetailPen   Text should be drawn with colour reg. 0 */
  1,             /* BlockPen    Blocks should be drawn with colour reg. 1 */
  CLOSEWINDOW|   /* IDCMPFlags  We will recieve a message when the user:  */
                 /*             selects the Close window gad, or when the */
  MOUSEBUTTONS,  /*             user presses/releases the mouse buttons.  */
  SMART_REFRESH| /* Flags       Intuition should refresh the window. */
  WINDOWCLOSE|   /*             Close Gadget. */
  WINDOWDRAG|    /*             Drag gadget. */
  WINDOWDEPTH|   /*             Depth arrange Gadgets. */
  WINDOWSIZING|  /*             Sizing Gadget. */
  ACTIVATE|      /*             The window should be Active when opened. */

  RMBTRAP,       /*             We do not want any menu operations for   */
                 /*             this window. We can then recieve         */
                 /*             MOUSEBUTTONS events even if the right    */
                 /*             mouse button is pressed. (Such event are */
                 /*             normally swollowed by Intuition.)        */

  NULL,          /* FirstGadget No gadgets connected to this window. */
  NULL,          /* CheckMark   Use Intuition's default CheckMark. */
  "PRESS THE BUTTONS", /* Title Title of the window. */
  NULL,          /* Screen      Connected to the Workbench Screen. */
  NULL,          /* BitMap      No Custom BitMap. */
  100,           /* MinWidth    We will not allow the window to become */
  50,            /* MinHeight   smaller than 100 x 50, and not bigger */
  400,           /* MaxWidth    than 400 x 200. */
  200,           /* MaxHeight */
  WBENCHSCREEN   /* Type        Connected to the Workbench Screen. */
};



main()
{
  /* Boolean variable used for the while loop: */
  BOOL close_me;

  ULONG class; /* IDCMP flag. */
```

```c
  USHORT code; /* Code. */

  /* Pointer to an IntuiMessage structure: */
  struct IntuiMessage *my_message;



  /* Before we can use Intuition we need to open the Intuition Library: */
  IntuitionBase = (struct IntuitionBase *)
    OpenLibrary( "intuition.library", 0 );

  if( IntuitionBase == NULL )
    exit(); /* Could NOT open the Intuition Library! */



  /* We will now try to open the window: */
  my_window = (struct Window *) OpenWindow( &my_new_window );

  /* Have we opened the window succesfully? */
  if(my_window == NULL)
  {
    /* Could NOT open the Window! */

    /* Close the Intuition Library since we have opened it: */
    CloseLibrary( IntuitionBase );

    exit();
  }



  /* We have opened the window, and everything seems to be OK. */

  printf("Press the mouse buttons!\n");



  close_me = FALSE;

  /* Stay in the while loop until the user has selected the Close window */
  /* gadget: */
  while( close_me == FALSE )
  {
    /* Wait until we have recieved a message: */
    Wait( 1 << my_window->UserPort->mp_SigBit );

    /* As long as we can collect messages successfully we stay in the */
    /* while-loop: */
    while(my_message = (struct IntuiMessage *) GetMsg(my_window->UserPort))
    {
      /* After we have successfully collected the message we can read */
      /* it, and save any important values which we maybe want to check */
      /* later: */
      class = my_message->Class;
      code  = my_message->Code;

      /* After we have read it we reply as fast as possible: */
      /* REMEMBER! Do never try to read a message after you have replied! */
      /* (Some other process has maybe changed it.) */
      ReplyMsg( my_message );
      /* Check which IDCMP flag was sent: */
      switch( class )
      {
        case CLOSEWINDOW:  /* The user selected the Close window gadget! */
               close_me=TRUE;
               break;

        case MOUSEBUTTONS: /* The user pressed/released a mouse button. */
               /* We shall now check wich button, and if it was pressed */
               /* or released: */
               switch( code )
               {
                 case SELECTDOWN: /* Left button pressed. */
                        printf("Left mouse button pressed.\n");
                        break;
                 case SELECTUP:   /* Left button releassed. */
                        printf("Left mouse button released.\n");
                        break;
                 case MENUDOWN:   /* Right button pressed. */
                        printf("Right mouse button pressed.\n");
                        break;
                 case MENUUP:     /* Right button released. */
                        printf("Right mouse button released.\n");
                        break;
               }
               break;
      }
    }
  }



  /* Close the window: */
  CloseWindow( my_window );



  /* Close the Intuition Library: */
  CloseLibrary( IntuitionBase );

  /* THE END */
}
```

Example2

This program explains how to use the IDCMP flag MOUSEMOVE.

```c
/* Example2                                                 */
/* This program explains how to use the IDCMP flag MOUSEMOVE. */



#include <intuition/intuition.h>



struct IntuitionBase *IntuitionBase;



/* Declare a pointer to a Window structure: */
struct Window *my_window;

/* Declare and initialize your NewWindow structure: */
struct NewWindow my_new_window=
{
  50,            /* LeftEdge    x position of the window. */
  25,            /* TopEdge     y positio of the window. */
  320,           /* Width       320 pixels wide. */
  100,           /* Height      100 lines high. */
  0,             /* DetailPen   Text should be drawn with colour reg. 0 */
  1,             /* BlockPen    Blocks should be drawn with colour reg. 1 */
  CLOSEWINDOW|   /* IDCMPFlags  We will recieve a message when the user:  */
                 /*             selects the Close window gad, or when the */
  MOUSEMOVE,     /*             user moves the mouse.                     */
  SMART_REFRESH| /* Flags       Intuition should refresh the window. */
  WINDOWCLOSE|   /*             Close Gadget. */
  WINDOWDRAG|    /*             Drag gadget. */
  WINDOWDEPTH|   /*             Depth arrange Gadgets. */
  WINDOWSIZING|  /*             Sizing Gadget. */
  ACTIVATE|      /*             The window should be Active when opened. */

  REPORTMOUSE,   /*             Create MOUSEMOVE messages whenever this */
                 /*             window is active and the mouse is moved. */

  NULL,          /* FirstGadget No gadgets connected to this window. */
  NULL,          /* CheckMark   Use Intuition's default CheckMark. */
  "MOVE THE MOUSE", /* Title    Title of the window. */
  NULL,          /* Screen      Connected to the Workbench Screen. */
  NULL,          /* BitMap      No Custom BitMap. */
  100,           /* MinWidth    We will not allow the window to become */
  50,            /* MinHeight   smaller than 100 x 50, and not bigger */
  400,           /* MaxWidth    than 400 x 200. */
  200,           /* MaxHeight */
  WBENCHSCREEN   /* Type        Connected to the Workbench Screen. */
};



/*********************************************************************/
/* Extra information:                                                */
/* If we set the IDCMP flag MOUSEMOVE then we tell Intuition that we  */
/* are interested in MOUSEMOVE events. However, we must tell Intuition */
/* how and when these messages should be created. There exist two      */
/* ways to do it:                                                     */
/* 1. Set the flag FOLLOWMOUSE in the Activation field in the Gadget   */
/*    structure. We will then recieve messages whenever the gadget is  */
/*    selected and the mouse is moved.                                */
/* 2. Set the flag REPORTMOUSE in the Flag field in the NewWindow      */
/*    structure. We will then recieve messages whenever the window is  */
/*    active and the mouse is moved. (Showed in this example.)         */
/*********************************************************************/



main()
{
  /* Boolean variable used for the while loop: */
  BOOL close_me;

  ULONG class; /* IDCMP flag. */

  SHORT x, y;  /* Position of the mouse (x,y). */

  BOOL mouse_moved;

  /* Pointer to an IntuiMessage structure: */
  struct IntuiMessage *my_message;



  /* Before we can use Intuition we need to open the Intuition Library: */
  IntuitionBase = (struct IntuitionBase *)
    OpenLibrary( "intuition.library", 0 );

  if( IntuitionBase == NULL )
    exit(); /* Could NOT open the Intuition Library! */



  /* We will now try to open the window: */
  my_window = (struct Window *) OpenWindow( &my_new_window );

  /* Have we opened the window succesfully? */
  if(my_window == NULL)
  {
    /* Could NOT open the Window! */

    /* Close the Intuition Library since we have opened it: */
    CloseLibrary( IntuitionBase );

    exit();
  }



  /* We have opened the window, and everything seems to be OK. */

  printf("Move the mouse!\n");



  close_me = FALSE;

  /* Stay in the while loop until the user has selected the Close window */
  /* gadget: */
  while( close_me == FALSE )
  {
    mouse_moved = FALSE;
    /* Wait until we have recieved a message: */
    Wait( 1 << my_window->UserPort->mp_SigBit );

    /* As long as we can collect messages successfully we stay in the */
    /* while-loop: */
    while(my_message = (struct IntuiMessage *) GetMsg(my_window->UserPort))
    {
      /* After we have successfully collected the message we can read */
      /* it, and save any important values which we maybe want to check */
      /* later: */
      class = my_message->Class;  /* IDCMP flag. */
      x     = my_message->MouseX; /* X position of the mouse. */
      y     = my_message->MouseY; /* Y position of the mouse. */


      /* After we have read it we reply as fast as possible: */
      /* REMEMBER! Do never try to read a message after you have replied! */
      /* (Some other process has maybe changed it.) */
      ReplyMsg( my_message );


      /* Check which IDCMP flag was sent: */
      switch( class )
      {
        case CLOSEWINDOW:  /* The user selected the Close window gadget! */
               close_me=TRUE;
               break;

        case MOUSEMOVE:    /* The user moved the mouse. */
               mouse_moved = TRUE;
               break;
      }
    }

    if( mouse_moved )
    {
      /* Since we recieve so many messages when the mouse is moved, we */
      /* respond first when the mouse has halted. Print out the mouse  */
      /* position relative to the top left corner of the window.       */
      printf("New position: (%d, %d)\n", x, y);
    }
  }



  /* Close the window: */
  CloseWindow( my_window );



  /* Close the Intuition Library: */
  CloseLibrary( IntuitionBase );

  /* THE END */
}
```

Example3

This program explains how to use the IDCMP flags: NEWSIZE, ACTIVEWINDOW and INACTIVEWINDOW.

```
/* Example3                                                   */
/* This program explains how to use the IDCMP flags: NEWSIZE, */
/* ACTIVEWINDOW and INACTIVEWINDOW.                           */



#include <intuition/intuition.h>



struct IntuitionBase *IntuitionBase;



/* Declare a pointer to a Window structure: */
struct Window *my_window;

/* Declare and initialize your NewWindow structure: */
struct NewWindow my_new_window=
{
  50,             /* LeftEdge    x position of the window. */
  25,             /* TopEdge     y positio of the window. */
  320,            /* Width       320 pixels wide. */
  100,            /* Height      100 lines high. */
  0,              /* DetailPen   Text should be drawn with colour reg. 0 */
  1,              /* BlockPen    Blocks should be drawn with colour r. 1 */
  CLOSEWINDOW|    /* IDCMPFlags  We will recieve a message when the user: */
                  /*             selects the Close window gad, or when    */

  NEWSIZE|        /*             the user resizes the window.             */
  ACTIVEWINDOW|   /*             We will also recieve a message whenever  */
  INACTIVEWINDOW, /*             the window is activated/deactivated.     */

  SMART_REFRESH|  /* Flags       Intuition should refresh the window. */
  WINDOWCLOSE|    /*             Close Gadget. */
  WINDOWDRAG|     /*             Drag gadget. */
  WINDOWDEPTH|    /*             Depth arrange Gadgets. */
  WINDOWSIZING|   /*             Sizing Gadget. */
  ACTIVATE,       /*             The window should be Active when opened. */
  NULL,           /* FirstGadget No gadgets connected to this window. */
  NULL,           /* CheckMark   Use Intuition's default CheckMark. */
  "PLAY WITH WINDOWS", /* Title  Title of the window. */
  NULL,           /* Screen      Connected to the Workbench Screen. */
  NULL,           /* BitMap      No Custom BitMap. */
  100,            /* MinWidth    We will not allow the window to become */
  50,             /* MinHeight   smaller than 100 x 50, and not bigger */
  400,            /* MaxWidth    than 400 x 200. */
  200,            /* MaxHeight */
  WBENCHSCREEN    /* Type        Connected to the Workbench Screen. */
};



main()
{
  /* Boolean variable used for the while loop: */
  BOOL close_me;

  ULONG class; /* IDCMP flag. */

  /* Pointer to an IntuiMessage structure: */
  struct IntuiMessage *my_message;
```

```


/* Before we can use Intuition we need to open the Intuition Library: */
IntuitionBase = (struct IntuitionBase *)
  OpenLibrary( "intuition.library", 0 );

if( IntuitionBase == NULL )
  exit(); /* Could NOT open the Intuition Library! */



/* We will now try to open the window: */
my_window = (struct Window *) OpenWindow( &my_new_window );

/* Have we opened the window succesfully? */
if(my_window == NULL)
{
  /* Could NOT open the Window! */

  /* Close the Intuition Library since we have opened it: */
  CloseLibrary( IntuitionBase );

  exit();
}



/* We have opened the window, and everything seems to be OK. */

printf("Play with the window!\n\n");



close_me = FALSE;

/* Stay in the while loop until the user has selected the Close window */
/* gadget: */
while( close_me == FALSE )
{
  /* Wait until we have recieved a message: */
  Wait( 1 << my_window->UserPort->mp_SigBit );


  /* As long as we can collect messages successfully we stay in the */
  /* while-loop: */
  while(my_message = (struct IntuiMessage *) GetMsg(my_window->UserPort))
  {
    /* After we have successfully collected the message we can read */
    /* it, and save any important values which we maybe want to check */
    /* later: */
    class = my_message->Class;  /* IDCMP flag. */


    /* After we have read it we reply as fast as possible: */
    /* REMEMBER! Do never try to read a message after you have replied! */
    /* (Some other process has maybe changed it.) */
    ReplyMsg( my_message );


    /* Check which IDCMP flag was sent: */
    switch( class )
```

```
      {
        case CLOSEWINDOW:    /* The user selected the Close window gad. */
               close_me=TRUE;
               break;

        case NEWSIZE:        /* The user resized the window. */
               printf("The window was resized!\n");
               printf("Width: %d\n", my_window->Width);
               printf("Height: %d\n\n", my_window->Height);
               break;

        case ACTIVEWINDOW:   /* Window actiavted. */
               printf("Window activated!\n\n");
               break;

        case INACTIVEWINDOW: /* Window inactiavted. */
               printf("Window inactivated!\n\n");
               break;
      }
    }
  }



  /* Close the window: */
  CloseWindow( my_window );



  /* Close the Intuition Library: */
  CloseLibrary( IntuitionBase );

  /* THE END */
}
```

Example4

This program explains how to use the IDCMP flag SIZEVERIFY.

```
/* Example4                                                  */
/* This program explains how to use the IDCMP flag SIZEVERIFY. */



#include <intuition/intuition.h>



struct IntuitionBase *IntuitionBase;



/* Declare a pointer to a Window structure: */
struct Window *my_window;

/* Declare and initialize your NewWindow structure: */
struct NewWindow my_new_window=
{
  50,             /* LeftEdge    x position of the window. */
  25,             /* TopEdge     y positio of the window. */
  320,            /* Width       320 pixels wide. */
  100,            /* Height      100 lines high. */
  0,              /* DetailPen   Text should be drawn with colour reg. 0 */
  1,              /* BlockPen    Blocks should be drawn with colour r. 1 */
  CLOSEWINDOW|    /* IDCMPFlags  We will recieve a message when the user: */
                  /*             selects the Close window gad.            */

  SIZEVERIFY,     /*             We will also recieve a verifying message */
                  /*             when the user resizes the window.        */
                  /*             However, the window will change size     */
                  /*             first when we have replied.              */

  SMART_REFRESH|  /* Flags       Intuition should refresh the window. */
  WINDOWCLOSE|    /*             Close Gadget. */
  WINDOWDRAG|     /*             Drag gadget. */
  WINDOWDEPTH|    /*             Depth arrange Gadgets. */
  WINDOWSIZING|   /*             Sizing Gadget. */
  ACTIVATE,       /*             The window should be Active when opened. */
  NULL,           /* FirstGadget No gadgets connected to this window. */
  NULL,           /* CheckMark   Use Intuition's default CheckMark. */
  "RESIZABLE WINDOW", /* Title   Title of the window. */
  NULL,           /* Screen      Connected to the Workbench Screen. */
  NULL,           /* BitMap      No Custom BitMap. */
  100,            /* MinWidth    We will not allow the window to become */
  50,             /* MinHeight   smaller than 100 x 50, and not bigger */
  400,            /* MaxWidth    than 400 x 200. */
  200,            /* MaxHeight */
  WBENCHSCREEN    /* Type        Connected to the Workbench Screen. */
};



/***********************************************************************/
/* Extra information:                                                  */
/* When we recieve a SIZEVERIFY message we may finish off with something */
/* before we allow Intuition to resize the window. The window will first */
/* be resized when we have replied.                                    */
/***********************************************************************/
```

```
main()
{
  /* Boolean variable used for the while loop: */
  BOOL close_me;

  ULONG class; /* IDCMP flag. */

  /* Pointer to an IntuiMessage structure: */
  struct IntuiMessage *my_message;



  /* Before we can use Intuition we need to open the Intuition Library: */
  IntuitionBase = (struct IntuitionBase *)
    OpenLibrary( "intuition.library", 0 );

  if( IntuitionBase == NULL )
    exit(); /* Could NOT open the Intuition Library! */



  /* We will now try to open the window: */
  my_window = (struct Window *) OpenWindow( &my_new_window );

  /* Have we opened the window succesfully? */
  if(my_window == NULL)
  {
    /* Could NOT open the Window! */

    /* Close the Intuition Library since we have opened it: */
    CloseLibrary( IntuitionBase );

    exit();
  }



  /* We have opened the window, and everything seems to be OK. */

  printf("Try to resize the window!\n\n");



  close_me = FALSE;

  /* Stay in the while loop until the user has selected the Close window */
  /* gadget: */
  while( close_me == FALSE )
  {
    /* Wait until we have recieved a message: */
    Wait( 1 << my_window->UserPort->mp_SigBit );


    /* As long as we can collect messages successfully we stay in the */
    /* while-loop: */
    while(my_message = (struct IntuiMessage *) GetMsg(my_window->UserPort))
    {
      /* After we have successfully collected the message we can read */
      /* it, and save any important values which we maybe want to check */
      /* later: */
      class = my_message->Class;  /* IDCMP flag. */
      /* Check if we have recieved a SIZEVERIFY message: */
      if( class == SIZEVERIFY )
      {
        /* The user tries to resize the window. However, it will first */
        /* change size when we have replied, so we may finish of with  */
        /* something before. We will now take a little pause, just to  */
        /* show that we can control the resizing of the window:        */
        printf("So you tried to resize the window! Tough luck!\n");
        printf("Here I decide when you may resize it, and I want\n");
        printf("to take a pause...\n");

        /* Wait 2 seconds: */
        Delay( 2 * 50 );

        printf("OK! You may now resize the window.\n\n");
        /* Once we reply the window will be resized. */
      }


      /* After we have read it we reply as fast as possible: */
      /* REMEMBER! Do never try to read a message after you have replied! */
      /* (Some other process has maybe changed it.) */
      ReplyMsg( my_message );


      /* Check which IDCMP flag was sent: */
      switch( class )
      {
        case CLOSEWINDOW:    /* The user selected the Close window gad. */
               close_me=TRUE;
               break;
      }
    }
  }



  /* Close the window: */
  CloseWindow( my_window );



  /* Close the Intuition Library: */
  CloseLibrary( IntuitionBase );

  /* THE END */
}
```

Example5

This program explains how to use the IDCMP flag RAWKEY.

```
/* Example5                                             */
/* This program explains how to use the IDCMP flag RAWKEY. */



#include <intuition/intuition.h>



struct IntuitionBase *IntuitionBase;



/* Declare a pointer to a Window structure: */
struct Window *my_window;

/* Declare and initialize your NewWindow structure: */
struct NewWindow my_new_window=
{
  50,             /* LeftEdge    x position of the window. */
  25,             /* TopEdge     y positio of the window. */
  320,            /* Width       320 pixels wide. */
  100,            /* Height      100 lines high. */
  0,              /* DetailPen   Text should be drawn with colour reg. 0 */
  1,              /* BlockPen    Blocks should be drawn with colour r. 1 */
  CLOSEWINDOW|    /* IDCMPFlags  We will recieve a message when the user */
                  /*             selects the Close window gad.           */

  RAWKEY,         /*             We will also recieve a message whenever */
                  /*             the user presses/releases a key.        */

  SMART_REFRESH|  /* Flags       Intuition should refresh the window. */
  WINDOWCLOSE|    /*             Close Gadget. */
  WINDOWDRAG|     /*             Drag gadget. */
  WINDOWDEPTH|    /*             Depth arrange Gadgets. */
  WINDOWSIZING|   /*             Sizing Gadget. */
  ACTIVATE,       /*             The window should be Active when opened. */
  NULL,           /* FirstGadget No gadgets connected to this window. */
  NULL,           /* CheckMark   Use Intuition's default CheckMark. */
  "PRESS MY KEYS",/* Title       Title of the window. */
  NULL,           /* Screen      Connected to the Workbench Screen. */
  NULL,           /* BitMap      No Custom BitMap. */
  100,            /* MinWidth    We will not allow the window to become */
  50,             /* MinHeight   smaller than 100 x 50, and not bigger */
  400,            /* MaxWidth    than 400 x 200. */
  200,            /* MaxHeight */
  WBENCHSCREEN    /* Type        Connected to the Workbench Screen. */
};



/***********************************************************************/
/* Extra information:                                                  */
/* Whenever the user presses/releases a key will we recieve a message.    */
/* The Code part of the message contains the raw (untranslated) keykodes. */
/* (See Appendix * for more information about raw keykodes.) The          */
/* Qualifier field of the message tells us if any qualifier (SHIFT/CTRL   */
/* etc) was also pressed. (See Appendix * for more information about      */
/* qualifiers.                                                            */
/***********************************************************************/
```

```
main()
{
  /* Boolean variable used for the while loop: */
  BOOL close_me;

  ULONG class;      /* IDCMP flag. */
  USHORT code;      /* Code. */
  USHORT qualifier; /* Qualifier. */

  /* Pointer to an IntuiMessage structure: */
  struct IntuiMessage *my_message;



  /* Before we can use Intuition we need to open the Intuition Library: */
  IntuitionBase = (struct IntuitionBase *)
    OpenLibrary( "intuition.library", 0 );

  if( IntuitionBase == NULL )
    exit(); /* Could NOT open the Intuition Library! */



  /* We will now try to open the window: */
  my_window = (struct Window *) OpenWindow( &my_new_window );

  /* Have we opened the window succesfully? */
  if(my_window == NULL)
  {
    /* Could NOT open the Window! */

    /* Close the Intuition Library since we have opened it: */
    CloseLibrary( IntuitionBase );

    exit();
  }



  /* We have opened the window, and everything seems to be OK. */

  printf("Press some keys!\n\n");



  close_me = FALSE;

  /* Stay in the while loop until the user has selected the Close window */
  /* gadget: */
  while( close_me == FALSE )
  {
    /* Wait until we have recieved a message: */
    Wait( 1 << my_window->UserPort->mp_SigBit );


    /* As long as we can collect messages successfully we stay in the */
    /* while-loop: */
    while(my_message = (struct IntuiMessage *) GetMsg(my_window->UserPort))
    {
      /* After we have successfully collected the message we can read */
      /* it, and save any important values which we maybe want to check */
      /* later: */
      class = my_message->Class;         /* IDCMP flag. */
      code = my_message->Code;           /* Code. */
      qualifier = my_message->Qualifier; /* Qualifier. */


      /* After we have read it we reply as fast as possible: */
      /* REMEMBER! Do never try to read a message after you have replied! */
      /* (Some other process has maybe changed it.) */
      ReplyMsg( my_message );


      /* Check which IDCMP flag was sent: */
      switch( class )
      {
        case CLOSEWINDOW:    /* The user selected the Close window gad. */
               close_me=TRUE;
               break;

        case RAWKEY:         /* The user pressed/released a key! */
               /* Print out the raw keycode (both as decimal and hex.): */
               printf("Raw keycode: %6d(d) %6x(h)\n", code, code );

               /* Print out the qualifier (both as decimal and hex.): */
               printf("Qualifier:   %6d(d) %6x(h)\n", qualifier, qualifier);

               /* This shows how you can check if a SHIFT or CTRL */
               /* qualifier key was also pressed:                 */
               if( qualifier &= IEQUALIFIER_LSHIFT )
                 printf("Left SHIFT button pressed\n");

               if( qualifier &= IEQUALIFIER_RSHIFT )
                 printf("Right SHIFT button pressed\n");

               if( qualifier &= IEQUALIFIER_CONTROL )
                 printf("CTRL button pressed\n");

               printf("\n");
               break;
      }
    }
  }



  /* Close the window: */
  CloseWindow( my_window );



  /* Close the Intuition Library: */
  CloseLibrary( IntuitionBase );

  /* THE END */
}
```

Example6

This program explains how to use the IDCMP flag VANILLAKEY.

```
/* Example6                                                */
/* This program explains how to use the IDCMP flag VANILLAKEY. */



#include <intuition/intuition.h>



struct IntuitionBase *IntuitionBase;



/* Declare a pointer to a Window structure: */
struct Window *my_window;

/* Declare and initialize your NewWindow structure: */
struct NewWindow my_new_window=
{
  50,             /* LeftEdge    x position of the window. */
  25,             /* TopEdge     y positio of the window. */
  320,            /* Width       320 pixels wide. */
  100,            /* Height      100 lines high. */
  0,              /* DetailPen   Text should be drawn with colour reg. 0 */
  1,              /* BlockPen    Blocks should be drawn with colour r. 1 */
  CLOSEWINDOW|    /* IDCMPFlags  We will recieve a message when the user */
                  /*             selects the Close window gad.           */

  VANILLAKEY,     /*             We will also recieve a message whenever */
                  /*             the user presses/releases a key.        */

  SMART_REFRESH|  /* Flags       Intuition should refresh the window. */
  WINDOWCLOSE|    /*             Close Gadget. */
  WINDOWDRAG|     /*             Drag gadget. */
  WINDOWDEPTH|    /*             Depth arrange Gadgets. */
  WINDOWSIZING|   /*             Sizing Gadget. */
  ACTIVATE,       /*             The window should be Active when opened. */
  NULL,           /* FirstGadget No gadgets connected to this window. */
  NULL,           /* CheckMark   Use Intuition's default CheckMark. */
  "PRESS MY KEYS",/* Title       Title of the window. */
  NULL,           /* Screen      Connected to the Workbench Screen. */
  NULL,           /* BitMap      No Custom BitMap. */
  100,            /* MinWidth    We will not allow the window to become */
  50,             /* MinHeight   smaller than 100 x 50, and not bigger */
  400,            /* MaxWidth    than 400 x 200. */
  200,            /* MaxHeight */
  WBENCHSCREEN    /* Type        Connected to the Workbench Screen. */
};



/***************************************************************************/
/* Extra information:                                                      */
/* Whenever the user presses/releases a key will we recieve a message.    */
/* The Code part of the message contains the translated keykode (Default  */
/* keymap used). (See Appendix * for more information about ASCII codes.) */
/***************************************************************************/



main()
{
```

```
/* Boolean variable used for the while loop: */
BOOL close_me;

ULONG class;      /* IDCMP flag. */
USHORT code;      /* Code. */

/* Pointer to an IntuiMessage structure: */
struct IntuiMessage *my_message;



/* Before we can use Intuition we need to open the Intuition Library: */
IntuitionBase = (struct IntuitionBase *)
  OpenLibrary( "intuition.library", 0 );

if( IntuitionBase == NULL )
  exit(); /* Could NOT open the Intuition Library! */



/* We will now try to open the window: */
my_window = (struct Window *) OpenWindow( &my_new_window );

/* Have we opened the window succesfully? */
if(my_window == NULL)
{
  /* Could NOT open the Window! */

  /* Close the Intuition Library since we have opened it: */
  CloseLibrary( IntuitionBase );

  exit();
}



/* We have opened the window, and everything seems to be OK. */

printf("Press some keys!\n\n");



close_me = FALSE;

/* Stay in the while loop until the user has selected the Close window */
/* gadget: */
while( close_me == FALSE )
{
  /* Wait until we have recieved a message: */
  Wait( 1 << my_window->UserPort->mp_SigBit );


  /* As long as we can collect messages successfully we stay in the */
  /* while-loop: */
  while(my_message = (struct IntuiMessage *) GetMsg(my_window->UserPort))
  {
    /* After we have successfully collected the message we can read */
    /* it, and save any important values which we maybe want to check */
    /* later: */
    class = my_message->Class;         /* IDCMP flag. */
    code = my_message->Code;           /* Code. */
```

```
      /* After we have read it we reply as fast as possible: */
      /* REMEMBER! Do never try to read a message after you have replied! */
      /* (Some other process has maybe changed it.) */
      ReplyMsg( my_message );


      /* Check which IDCMP flag was sent: */
      switch( class )
      {
        case CLOSEWINDOW:    /* The user selected the Close window gad. */
               close_me=TRUE;
               break;

        case VANILLAKEY:     /* The user pressed/released a key! */
               /* Print out the translated keycode (as dec. and hex.): */
               printf("Translated keycode: %6d(d) %6x(h)\n\n", code, code );

               break;
      }
    }
  }



  /* Close the window: */
  CloseWindow( my_window );



  /* Close the Intuition Library: */
  CloseLibrary( IntuitionBase );

  /* THE END */
}
```

Example7

This program explains how to use the IDCMP flags:
DISKINSERTED and DISKREMOVED.

```
/* Example7                                                           */
/* This program explains how to use the IDCMP flags: DISKINSERTED and */
/* DISKREMOVED.                                                       */



#include <intuition/intuition.h>



struct IntuitionBase *IntuitionBase;



/* Declare a pointer to a Window structure: */
struct Window *my_window;

/* Declare and initialize your NewWindow structure: */
struct NewWindow my_new_window=
{
  50,             /* LeftEdge    x position of the window. */
  25,             /* TopEdge     y positio of the window. */
  320,            /* Width       320 pixels wide. */
  100,            /* Height      100 lines high. */
  0,              /* DetailPen   Text should be drawn with colour reg. 0 */
  1,              /* BlockPen    Blocks should be drawn with colour r. 1 */
  CLOSEWINDOW|    /* IDCMPFlags  We will recieve a message when the user */
                  /*             selects the Close window gad.           */

  DISKINSERTED|   /*             We will also recieve a message whenever */
  DISKREMOVED,    /*             a disk is inserted or removed.          */

  SMART_REFRESH|  /* Flags       Intuition should refresh the window. */
  WINDOWCLOSE|    /*             Close Gadget. */
  WINDOWDRAG|     /*             Drag gadget. */
  WINDOWDEPTH|    /*             Depth arrange Gadgets. */
  WINDOWSIZING|   /*             Sizing Gadget. */
  ACTIVATE,       /*             The window should be Active when opened. */
  NULL,           /* FirstGadget No gadgets connected to this window. */
  NULL,           /* CheckMark   Use Intuition's default CheckMark. */
  "TOUCH MY DISKS",/* Title      Title of the window. */
  NULL,           /* Screen      Connected to the Workbench Screen. */
  NULL,           /* BitMap      No Custom BitMap. */
  100,            /* MinWidth    We will not allow the window to become */
  50,             /* MinHeight   smaller than 100 x 50, and not bigger */
  400,            /* MaxWidth    than 400 x 200. */
  200,            /* MaxHeight */
  WBENCHSCREEN    /* Type        Connected to the Workbench Screen. */
};



/***********************************************************************/
/* Extra information:                                                  */
/* The IDCMP messages DISKINSERTED and DISKREMOVED can not be "swollowed" */
/* by a program. All applications will hear about it if they want. This  */
/* is the opposite of other IDCMP flags which will be "swollowed" by the */
/* active window's program.                                            */
/***********************************************************************/
```

```
main()
{
  /* Boolean variable used for the while loop: */
  BOOL close_me;

  ULONG class; /* IDCMP flag. */

  /* Pointer to an IntuiMessage structure: */
  struct IntuiMessage *my_message;



  /* Before we can use Intuition we need to open the Intuition Library: */
  IntuitionBase = (struct IntuitionBase *)
    OpenLibrary( "intuition.library", 0 );

  if( IntuitionBase == NULL )
    exit(); /* Could NOT open the Intuition Library! */



  /* We will now try to open the window: */
  my_window = (struct Window *) OpenWindow( &my_new_window );

  /* Have we opened the window succesfully? */
  if(my_window == NULL)
  {
    /* Could NOT open the Window! */

    /* Close the Intuition Library since we have opened it: */
    CloseLibrary( IntuitionBase );

    exit();
  }



  /* We have opened the window, and everything seems to be OK. */

  printf("Insert or remove a disk!\n\n");



  close_me = FALSE;

  /* Stay in the while loop until the user has selected the Close window */
  /* gadget: */
  while( close_me == FALSE )
  {
    /* Wait until we have recieved a message: */
    Wait( 1 << my_window->UserPort->mp_SigBit );


    /* As long as we can collect messages successfully we stay in the */
    /* while-loop: */
    while(my_message = (struct IntuiMessage *) GetMsg(my_window->UserPort))
    {
      /* After we have successfully collected the message we can read */
      /* it, and save any important values which we maybe want to check */
      /* later: */
      class = my_message->Class;         /* IDCMP flag. */
      /* After we have read it we reply as fast as possible: */
      /* REMEMBER! Do never try to read a message after you have replied! */
      /* (Some other process has maybe changed it.) */
      ReplyMsg( my_message );


      /* Check which IDCMP flag was sent: */
      switch( class )
      {
        case CLOSEWINDOW:    /* The user selected the Close window gad. */
               close_me=TRUE;
               break;

        case DISKINSERTED:   /* The user inserted a disk in a drive. */
               printf("Disk inserted!\n");
               break;

        case DISKREMOVED:    /* The user removed a disk from a drive. */
               printf("Disk removed!\n");
               break;
      }
    }
  }



  /* Close the window: */
  CloseWindow( my_window );



  /* Close the Intuition Library: */
  CloseLibrary( IntuitionBase );

  /* THE END */
}
```

Example8

This program explains how to use the IDCMP flag INTUITICKS.

```
/* Example8                                                   */
/* This program explains how to use the IDCMP flag INTUITICKS. */



#include <intuition/intuition.h>



struct IntuitionBase *IntuitionBase;



/* Declare a pointer to a Window structure: */
struct Window *my_window;

/* Declare and initialize your NewWindow structure: */
struct NewWindow my_new_window=
{
  50,             /* LeftEdge    x position of the window. */
  25,             /* TopEdge     y positio of the window. */
  320,            /* Width       320 pixels wide. */
  100,            /* Height      100 lines high. */
  0,              /* DetailPen   Text should be drawn with colour reg. 0 */
  1,              /* BlockPen    Blocks should be drawn with colour r. 1 */
  CLOSEWINDOW|    /* IDCMPFlags  We will recieve a message when the user */
                  /*             selects the Close window gad.           */

  INTUITICKS,     /*             We will also recieve simple time */
                  /*             events. (Around 10 times / second.) */

  SMART_REFRESH|  /* Flags       Intuition should refresh the window. */
  WINDOWCLOSE|    /*             Close Gadget. */
  WINDOWDRAG|     /*             Drag gadget. */
  WINDOWDEPTH|    /*             Depth arrange Gadgets. */
  WINDOWSIZING|   /*             Sizing Gadget. */
  ACTIVATE,       /*             The window should be Active when opened. */
  NULL,           /* FirstGadget No gadgets connected to this window. */
  NULL,           /* CheckMark   Use Intuition's default CheckMark. */
  "THE TIME IS SHORT",/* Title   Title of the window. */
  NULL,           /* Screen      Connected to the Workbench Screen. */
  NULL,           /* BitMap      No Custom BitMap. */
  100,            /* MinWidth    We will not allow the window to become */
  50,             /* MinHeight   smaller than 100 x 50, and not bigger */
  400,            /* MaxWidth    than 400 x 200. */
  200,            /* MaxHeight */
  WBENCHSCREEN    /* Type        Connected to the Workbench Screen. */
};



/***********************************************************************/
/* Extra information:                                                  */
/* INTUITICKS is a simply way of getting time messages. We will get     */
/* around (!) 10 messages per second. The nice thing with these messages */
/* is that it will not fill up the input buffer if you do not collect   */
/* them fast enough. If Intuition finds an INTUITICK message already    */
/* waiting in the port, no new messages will be created.               */
/***********************************************************************/
```

```
main()
{
  /* Boolean variable used for the while loop: */
  BOOL close_me;

  ULONG class;   /* IDCMP flag. */

  ULONG seconds; /* Seconds, copy of the system clock. */
  ULONG micros;  /* Micros,           - " -            */

  /* Pointer to an IntuiMessage structure: */
  struct IntuiMessage *my_message;



  /* Before we can use Intuition we need to open the Intuition Library: */
  IntuitionBase = (struct IntuitionBase *)
    OpenLibrary( "intuition.library", 0 );

  if( IntuitionBase == NULL )
    exit(); /* Could NOT open the Intuition Library! */



  /* We will now try to open the window: */
  my_window = (struct Window *) OpenWindow( &my_new_window );

  /* Have we opened the window succesfully? */
  if(my_window == NULL)
  {
    /* Could NOT open the Window! */

    /* Close the Intuition Library since we have opened it: */
    CloseLibrary( IntuitionBase );

    exit();
  }



  /* We have opened the window, and everything seems to be OK. */



  close_me = FALSE;

  /* Stay in the while loop until the user has selected the Close window */
  /* gadget: */
  while( close_me == FALSE )
  {
    /* Wait until we have recieved a message: */
    Wait( 1 << my_window->UserPort->mp_SigBit );


    /* As long as we can collect messages successfully we stay in the */
    /* while-loop: */
    while(my_message = (struct IntuiMessage *) GetMsg(my_window->UserPort))
    {
      /* After we have successfully collected the message we can read */
      /* it, and save any important values which we maybe want to check */
      /* later: */
      class = my_message->Class;     /* IDCMP flag. */
      /* Copies of the system clock when this message was created: */
      seconds = my_message->Seconds; /* Seconds. */
      micros = my_message->Micros;   /* Micros. */


      /* After we have read it we reply as fast as possible: */
      /* REMEMBER! Do never try to read a message after you have replied! */
      /* (Some other process has maybe changed it.) */
      ReplyMsg( my_message );


      /* Check which IDCMP flag was sent: */
      switch( class )
      {
        case CLOSEWINDOW:    /* The user selected the Close window gad. */
               close_me=TRUE;
               break;

        case INTUITICKS:     /* Time event. */
               printf("Seconds: %6d  Micros: %6d\n", seconds, micros);
               break;
      }
    }
  }



  /* Close the window: */
  CloseWindow( my_window );



  /* Close the Intuition Library: */
  CloseLibrary( IntuitionBase );

  /* THE END */
}
```

Example9

This program explains how to use the IDCMP flag REFRESHWINDOW, and how to optimize the redrawing of the window.

```
/* Example9                                                          */
/* This program explains how to use the IDCMP flag REFRESHWINDOW, and */
/* how to optimize the redrawing of the window.                      */



#include <intuition/intuition.h>



struct IntuitionBase *IntuitionBase;



/* Declare a pointer to a Window structure: */
struct Window *my_window;

/* Declare and initialize your NewWindow structure: */
struct NewWindow my_new_window=
{
  50,             /* LeftEdge    x position of the window. */
  25,             /* TopEdge     y positio of the window. */
  320,            /* Width       320 pixels wide. */
  100,            /* Height      100 lines high. */
  0,              /* DetailPen   Text should be drawn with colour reg. 0 */
  1,              /* BlockPen    Blocks should be drawn with colour r. 1 */
  CLOSEWINDOW|    /* IDCMPFlags  We will recieve a message when the user */
                  /*             selects the Close window gad.           */

  REFRESHWINDOW,  /*             We will recieve a message whenever we */
                  /*             need to refresh (redraw) the window. */

  SIMPLE_REFRESH| /* Flags       Your program has to refresh the window. */
  WINDOWCLOSE|    /*             Close Gadget. */
  WINDOWDRAG|     /*             Drag gadget. */
  WINDOWDEPTH|    /*             Depth arrange Gadgets. */
  WINDOWSIZING|   /*             Sizing Gadget. */
  ACTIVATE,       /*             The window should be Active when opened. */
  NULL,           /* FirstGadget No gadgets connected to this window. */
  NULL,           /* CheckMark   Use Intuition's default CheckMark. */
  "UPDATE ME",    /* Title       Title of the window. */
  NULL,           /* Screen      Connected to the Workbench Screen. */
  NULL,           /* BitMap      No Custom BitMap. */
  100,            /* MinWidth    We will not allow the window to become */
  50,             /* MinHeight   smaller than 100 x 50, and not bigger */
  400,            /* MaxWidth    than 400 x 200. */
  200,            /* MaxHeight */
  WBENCHSCREEN    /* Type        Connected to the Workbench Screen. */
};



/*********************************************************************/
/* Extra information:                                                */
/* We will recieve a REFRESHWINDOW message whenever we need to redraw    */
/* the window's display. If the window is a SuperBitmap window you never */
/* need to redraw it since it has its own BitMap. However, if the window */
/* is of the type SIMPLE_REFRESH or SMART_REFRESH it can happen that     */
/* your program need to redraw the window.                           */
/* SIMPLE_REFRESH: You need to update the window if it is resized,   */
/*                 pushed from behind to the front, or is moved.     */
/* SMART_REFRESH:  You need to update the display if it is resized.  */
/*                                                                   */
/* Once you recieve the message you should redraw the window. However,   */
/* before you start to redraw you need to call the function:         */
/* BeginRefresh(), and when you have finished you should call the    */
/* function EndRefresh(). (Even if you do not redraw anything, you   */
/* should call these functions.) The functions will improve the speed of */
/* the redrawing since only the trashed parts will be redrawed.      */
/*********************************************************************/



main()
{
  /* Boolean variable used for the while loop: */
  BOOL close_me;

  ULONG class;   /* IDCMP flag. */

  /* Pointer to an IntuiMessage structure: */
  struct IntuiMessage *my_message;



  /* Before we can use Intuition we need to open the Intuition Library: */
  IntuitionBase = (struct IntuitionBase *)
    OpenLibrary( "intuition.library", 0 );

  if( IntuitionBase == NULL )
    exit(); /* Could NOT open the Intuition Library! */



  /* We will now try to open the window: */
  my_window = (struct Window *) OpenWindow( &my_new_window );

  /* Have we opened the window succesfully? */
  if(my_window == NULL)
  {
    /* Could NOT open the Window! */

    /* Close the Intuition Library since we have opened it: */
    CloseLibrary( IntuitionBase );

    exit();
  }



  /* We have opened the window, and everything seems to be OK. */



  close_me = FALSE;

  /* Stay in the while loop until the user has selected the Close window */
  /* gadget: */
  while( close_me == FALSE )
  {
    /* Wait until we have recieved a message: */
    Wait( 1 << my_window->UserPort->mp_SigBit );
    /* As long as we can collect messages successfully we stay in the */
    /* while-loop: */
    while(my_message = (struct IntuiMessage *) GetMsg(my_window->UserPort))
    {
      /* After we have successfully collected the message we can read */
      /* it, and save any important values which we maybe want to check */
      /* later: */
      class = my_message->Class;     /* IDCMP flag. */


      /* After we have read it we reply as fast as possible: */
      /* REMEMBER! Do never try to read a message after you have replied! */
      /* (Some other process has maybe changed it.) */
      ReplyMsg( my_message );


      /* Check which IDCMP flag was sent: */
      switch( class )
      {
        case CLOSEWINDOW:    /* The user selected the Close window gad. */
               close_me=TRUE;
               break;

        case REFRESHWINDOW:  /* You need to update the window. */
               printf("We need to redraw the window! (Well almost)\n");

               /* Start the redrawing: */
               BeginRefresh( my_window );

               /* Redraw the window. For example call the function       */
               /* RefreshGadgets(), DrawImage(), DrawBorder() etc...     */
               /* In this example we do not redraw anything (there does  */
               /* not exist anything to redraw). However, even if you do */
               /* nothing you need to call the functions BeginRefresh()  */
               /* and EndRefresh().                                      */

               /* End the redrawing: */
               EndRefresh( my_window );
               break;
      }
    }
  }



  /* Close the window: */
  CloseWindow( my_window );



  /* Close the Intuition Library: */
  CloseLibrary( IntuitionBase );

  /* THE END */
}
```

A.9 MISCELLANEOUS

Example1

This example shows how to allocate, and deallocate memory.

```
/* Example1                                                  */
/* This example shows how to allocate, and deallocate memory. */


#include <exec/types.h>  /* Declares CPTR. */
#include <exec/memory.h> /* Declares MEMF_CHIP. */


main()
{
  /* Declare a pointer to the memory you are going to allocate: */
  CPTR memory;


  /* Allocate 150 bytes of Chip memory: */
  memory = (CPTR) AllocMem( 150, MEMF_CHIP );


  /* Have we allocated the memory successfully? */
  if( memory == NULL )
  {
    printf("Could not allocate the memory!\n");
    exit();
  }


  /* Do whatever you want to do with the memory! */


  /* Free the memory we have previously allocated: */
  FreeMem( memory, 150 );
}
```

Example2

This example shows how to allocate and deallocate memory with help of the functions AllocRemember(), and FreeRemember().

```
/* Example2                                                         */
/* This example shows how to allocate and deallocate memory with help of */
/* the functions AllocRemember(), and FreeRemember().                */


#include <intuition/intuition.h>
#include <exec/memory.h>


struct IntuitionBase *IntuitionBase;


main()
{
  /* Declare and initialize a pointer to the first Remember structure: */
  struct Remember *remember = NULL;

  /* Declare three memory pointers: */
  CPTR memory1, memory2, memory3;



  /* Open the Intuition Library: */
  IntuitionBase = (struct IntuitionBase *)
    OpenLibrary( "intuition.library", 0 );

  if( IntuitionBase == NULL )
    exit(); /* Could NOT open the Intuition Library! */



  /* Allocate 350 bytes of Chip memory, which is cleared: */
  memory1 = AllocRemember( &remember, 350, MEMF_CHIP|MEMF_CLEAR );

  if( memory1 == NULL )
  {
    CloseLibrary( IntuitionBase );
    exit();
  }


  /* Allocate 900 bytes of memory (any type, Fast if possible): */
  memory2 = AllocRemember( &remember, 900, MEMF_PUBLIC );

  if( memory2 == NULL )
  {
    FreeRemember( &remember, TRUE );
    CloseLibrary( IntuitionBase );
    exit();
  }


  /* Allocate 100 bytes of Chip memory: *
  memory3 = AllocRemember( &remember, 100, MEMF_CHIP );

  if( memory3 == NULL )
  {
    FreeRemember( &remember, TRUE );
    CloseLibrary( IntuitionBase );
    exit();
  }


  /* Do whatever you want to do with the memory. */



  /* Deallocate all memory with one single call: */
  FreeRemember( &remember, TRUE );

  /* Close the Intuition Library: */
  CloseLibrary( IntuitionBase );
}
```

Example3

This example shows how to get a copy of the preferences.

```
/* Example3                                                */
/* This example shows how to get a copy of the preferences. */


#include <intuition/intuition.h>



struct IntuitionBase *IntuitionBase;



main()
{
  /* Declare a preferences structure: */
  struct Preferences pref;



  /* Open the Intuition Library: */
  IntuitionBase = (struct IntuitionBase *)
    OpenLibrary( "intuition.library", 0 );

  if( IntuitionBase == NULL )
    exit(); /* Could NOT open the Intuition Library! */



  /* Try to get a copy of the current preferences (whole): */
  if( GetPrefs( &pref, sizeof(pref) ) == NULL )
  {
    /* Could not get a copy of the preferences! */
    CloseLibrary( IntuitionBase );
    exit();
  }



  /* We have now a copy of the preferences. */
  /* Do what ever you want...               */

  /* Why not print out the workbench clours? */
  printf( "\nWorkbench Screen Colours:\n");
  printf( "             RGB\n" );
  printf( "Colour 0: 0x%04x\n", pref.color0 );
  printf( "Colour 1: 0x%04x\n", pref.color1 );
  printf( "Colour 2: 0x%04x\n", pref.color2 );
  printf( "Colour 3: 0x%04x\n\n", pref.color3 );



  /* Close the Intuition Library: */
  CloseLibrary( IntuitionBase );
}
```

Example4

This example shows how to handle double mouse button events.

```
/* Example4                                                  */
/* This example shows how to handle double mouse button events. */



#include <intuition/intuition.h>



struct IntuitionBase *IntuitionBase;



/* Declare a pointer to a Window structure: */
struct Window *my_window;

/* Declare and initialize your NewWindow structure: */
struct NewWindow my_new_window=
{
  50,            /* LeftEdge    x position of the window. */
  25,            /* TopEdge     y positio of the window. */
  400,           /* Width       400 pixels wide. */
  100,           /* Height      100 lines high. */
  0,             /* DetailPen   Text should be drawn with colour reg. 0 */
  1,             /* BlockPen    Blocks should be drawn with colour reg. 1 */
  CLOSEWINDOW|   /* IDCMPFlags  We will recieve a message when the user:  */
                 /*             selects the Close window gad, or when the */
  MOUSEBUTTONS,  /*             user presses/releases the mouse buttons.  */
  SMART_REFRESH| /* Flags       Intuition should refresh the window. */
  WINDOWCLOSE|   /*             Close Gadget. */
  WINDOWDRAG|    /*             Drag gadget. */
  WINDOWDEPTH|   /*             Depth arrange Gadgets. */
  WINDOWSIZING|  /*             Sizing Gadget. */
  ACTIVATE,      /*             The window should be Active when opened. */
  NULL,          /* FirstGadget No gadgets connected to this window. */
  NULL,          /* CheckMark   Use Intuition's default CheckMark. */
  "DOUBLE CLICK ON THE LEFT MOUSE BUTTON", /* Title Title of the window. */
  NULL,          /* Screen      Connected to the Workbench Screen. */
  NULL,          /* BitMap      No Custom BitMap. */
  100,           /* MinWidth    We will not allow the window to become */
  50,            /* MinHeight   smaller than 100 x 50, and not bigger */
  400,           /* MaxWidth    than 400 x 200. */
  200,           /* MaxHeight */
  WBENCHSCREEN   /* Type        Connected to the Workbench Screen. */
};



main()
{
  /* Boolean variable used for the while loop: */
  BOOL close_me;

  /* Store some data copied from the IntuitionMessage in these variables: */
  ULONG class;           /* IDCMP flag. */
  USHORT code;           /* Code. */
  ULONG seconds, micros; /* Time. */

  /* Pointer to an IntuiMessage structure: */
  struct IntuiMessage *my_message;

  /* Declare and initialize the time stamps: */
```

```
  ULONG sec1 = 0;
  ULONG mic1 = 0;
  ULONG sec2 = 0;
  ULONG mic2 = 0;



  /* Before we can use Intuition we need to open the Intuition Library: */
  IntuitionBase = (struct IntuitionBase *)
    OpenLibrary( "intuition.library", 0 );

  if( IntuitionBase == NULL )
    exit(); /* Could NOT open the Intuition Library! */



  /* We will now try to open the window: */
  my_window = (struct Window *) OpenWindow( &my_new_window );

  /* Have we opened the window succesfully? */
  if(my_window == NULL)
  {
    /* Could NOT open the Window! */

    /* Close the Intuition Library since we have opened it: */
    CloseLibrary( IntuitionBase );

    exit();
  }



  /* We have opened the window, and everything seems to be OK. */



  close_me = FALSE;

  /* Stay in the while loop until the user has selected the Close window */
  /* gadget: */
  while( close_me == FALSE )
  {
    /* Wait until we have recieved a message: */
    Wait( 1 << my_window->UserPort->mp_SigBit );

    /* As long as we can collect messages successfully we stay in the */
    /* while-loop: */
    while(my_message = (struct IntuiMessage *) GetMsg(my_window->UserPort))
    {
      /* After we have successfully collected the message we can read */
      /* it, and save any important values which we maybe want to check */
      /* later: */
      class   = my_message->Class;
      code    = my_message->Code;
      seconds = my_message->Seconds;
      micros  = my_message->Micros;


      /* After we have read it we reply as fast as possible: */
      /* REMEMBER! Do never try to read a message after you have replied! */
      /* (Some other process has maybe changed it.) */
      ReplyMsg( my_message );
      /* Check which IDCMP flag was sent: */
      switch( class )
      {
        case CLOSEWINDOW:  /* The user selected the Close window gadget! */
               close_me=TRUE;
               break;

        case MOUSEBUTTONS: /* The user pressed/released a mouse button. */
               if( code == SELECTDOWN )
               {
                 /* Left button pressed. */

                 /* Save the old time: */
                 sec2 = sec1;
                 mic2 = mic1;

                 /* Get the new time: */
                 sec1 = seconds;
                 mic1 = micros;

                 /* Check if it was a double-click or not: */
                 if( DoubleClick( sec2, mic2, sec1, mic1 ) )
                 {
                   printf("Double-Click!\n");
                   /* Reset the values: */
                   sec1 = 0;
                   mic1 = 0;
                 }
               }
               break;
      }
    }
  }



  /* Close the window: */
  CloseWindow( my_window );



  /* Close the Intuition Library: */
  CloseLibrary( IntuitionBase );
}
```

Example5
This example prints out the current time.

```
/* Example5                                   */
/* This example prints out the current time. */


#include <intuition/intuition.h>


struct IntuitionBase *IntuitionBase;


main()
{
  ULONG seconds, micros;



  /* Open the Intuition Library: */
  IntuitionBase = (struct IntuitionBase *)
    OpenLibrary( "intuition.library", 0 );

  if( IntuitionBase == NULL )
    exit(); /* Could NOT open the Intuition Library! */



  /* Get the current time: */
  CurrentTime( &seconds, &micros );

  /* Print out the current time: */
  printf( "Seconds: %d\n", seconds );
  printf( "Micros:  %d\n", micros );



  /* Close the Intuition Library: */
  CloseLibrary( IntuitionBase );
}
```

A.10 SPRITES

Example1

This program shows how to declare and initialize some sprite data and a SimpleSprite structure. It also shows how to reserve a sprite (sprite 2), and how to move it around. The user moves the sprite by pressing the arrow keys.

```
/* Example1                                                           */
/* This program shows how to declare and initialize some sprite data   */
/* and a SimpleSprite structure. It also shows how to reserve a sprite */
/* (sprite 2), and how to move it around. The user moves the sprite by */
/* pressing the arrow keys.                                            */



#include <intuition/intuition.h>
/* Include this file since you are using sprites: */
#include <graphics/sprite.h>



/* Declare the functions we are going to use: */
void main();
void free_memory();



struct IntuitionBase *IntuitionBase = NULL;
/* We need to open the Graphics library since we are using sprites: */
struct GfxBase *GfxBase = NULL;



/* Declare a pointer to a Window structure: */
struct Window *my_window = NULL;

/* Declare and initialize your NewWindow structure: */
struct NewWindow my_new_window=
{
  40,            /* LeftEdge    x position of the window. */
  20,            /* TopEdge     y positio of the window. */
  200,           /* Width       200 pixels wide. */
  100,           /* Height      100 lines high. */
  0,             /* DetailPen   Text should be drawn with colour reg. 0 */
  1,             /* BlockPen    Blocks should be drawn with colour reg. 1 */
  CLOSEWINDOW|   /* IDCMPFlags  The window will give us a message if the */
  RAWKEY,        /*             user has selected the Close window gad, */
                 /*             or if the user has pressed a key. */
  SMART_REFRESH| /* Flags       Intuition should refresh the window. */
  WINDOWCLOSE|   /*             Close Gadget. */
  WINDOWDRAG|    /*             Drag gadget. */
  WINDOWDEPTH|   /*             Depth arrange Gadgets. */
  WINDOWSIZING|  /*             Sizing Gadget. */
  ACTIVATE,      /*             The window should be Active when opened. */
  NULL,          /* FirstGadget No Custom gadgets. */
  NULL,          /* CheckMark   Use Intuition's default CheckMark. */
  "SPRITES",     /* Title       Title of the window. */
  NULL,          /* Screen      Connected to the Workbench Screen. */
  NULL,          /* BitMap      No Custom BitMap. */
  80,            /* MinWidth    We will not allow the window to become */
  30,            /* MinHeight   smaller than 80 x 30, and not bigger */
  300,           /* MaxWidth    than 300 x 200. */
  200,           /* MaxHeight */
  WBENCHSCREEN   /* Type        Connected to the Workbench Screen. */
};



/**********************************************************************/
```

```
/* Extra information:                                                 */
/* When we declare the window pointer, the intuition library pointer */
/* etc, we initialize them to point to NULL:                         */
/* struct Window *my_window = NULL;                                  */
/* Since we then know that all of the pointers will point to NULL    */
/* when we start, we can check if they still point to NULL when we   */
/* quit. If they do not point to NULL anymore, we close that window, */
/* library etc.                                                      */
/*********************************************************************/



/*******************************************************/
/* 1. Declare and initialize some sprite graphics data: */
/*******************************************************/
UWORD chip my_sprite_data[36]=
{
  0x0000, 0x0000,

  0x0180, 0x0000,
  0x03C0, 0x0000,
  0x07E0, 0x0000,
  0x0FF0, 0x0000,
  0x1FF8, 0x0000,
  0x3FFC, 0x0000,
  0x7FFE, 0x0000,
  0x0000, 0xFFFF,
  0x0000, 0xFFFF,
  0x7FFE, 0x7FFE,
  0x3FFC, 0x3FFC,
  0x1FF8, 0x1FF8,
  0x0FF0, 0x0FF0,
  0x07E0, 0x07E0,
  0x03C0, 0x03C0,
  0x0180, 0x0180,

  0x0000, 0x0000
};



/*****************************************************/
/* 2. Declare and initialize a SimpleSprite structure: */
/*****************************************************/
struct SimpleSprite my_sprite=
{
  my_sprite_data, /* posctldata, pointer to the sprite data. */
  16,             /* height, 16 lines tall. */
  40, 60,         /* x, y, position on the screen. */
  -1,             /* num, this field is automatically initialized when  */
                  /* you call the GetSprite() function, so we set it to */
                  /* -1 for the moment.                                 */
};



void main()
{
  /* Sprite position: */
  WORD x = my_sprite.x;
  WORD y = my_sprite.y;
```

```
/* Direction of the sprite: */
WORD x_direction = 0;
WORD y_direction = 0;

/* Boolean variable used for the while loop: */
BOOL close_me = FALSE;

ULONG class; /* IDCMP */
USHORT code; /* Code */

/* Declare a pointer to an IntuiMessage structure: */
struct IntuiMessage *my_message;



/* Open the Intuition Library: */
IntuitionBase = (struct IntuitionBase *)
  OpenLibrary( "intuition.library", 0 );

if( IntuitionBase == NULL )
  free_memory(); /* Could NOT open the Intuition Library! */



/* Since we are using sprites we need to open the Graphics Library: */
/* Open the Graphics Library: */
GfxBase = (struct GfxBase *)
  OpenLibrary( "graphics.library", 0);

if( GfxBase == NULL )
  free_memory(); /* Could NOT open the Graphics Library! */



/* We will now try to open the window: */
my_window = (struct Window *) OpenWindow( &my_new_window );

/* Have we opened the window succesfully? */
if(my_window == NULL)
  free_memory(); /* Could NOT open the Window! */



/* Change the colour register 21 - 23: */
SetRGB4( &my_window->WScreen->ViewPort, 21, 0xF, 0x0, 0x0 ); /* Red */
SetRGB4( &my_window->WScreen->ViewPort, 22, 0xF, 0xF, 0x0 ); /* Yellow */
SetRGB4( &my_window->WScreen->ViewPort, 23, 0x0, 0xF, 0x0 ); /* Green */



/********************************/
/* 3. Try to reserve sprite 2: */
/********************************/
if( GetSprite( &my_sprite, 2 ) != 2 )
  free_memory(); /* Could not reserve sprite number 2. */



/* Stay in the while loop until the user has selected the Close window */
/* gadget: */
while( close_me == FALSE )
{
```

```
/* Stay in the while loop as long as we can collect messages */
/* sucessfully: */
while(my_message = (struct IntuiMessage *) GetMsg(my_window->UserPort))
{
  /* After we have collected the message we can read it, and save any */
  /* important values which we maybe want to check later: */
  class = my_message->Class;
  code  = my_message->Code;


  /* After we have read it we reply as fast as possible: */
  /* REMEMBER! Do never try to read a message after you have replied! */
  /* Some other process has maybe changed it. */
  ReplyMsg( my_message );


  /* Check which IDCMP flag was sent: */
  switch( class )
  {
    case CLOSEWINDOW:     /* Quit! */
           close_me=TRUE;
           break;

    case RAWKEY:          /* A key was pressed! */
           /* Check which key was pressed: */
           switch( code )
           {
             /* Up Arrow: */
             case 0x4C:      y_direction = -1; break; /* Pressed */
             case 0x4C+0x80: y_direction = 0;  break; /* Released */

             /* Down Arrow: */
             case 0x4D:      y_direction = 1; break; /* Pressed */
             case 0x4D+0x80: y_direction = 0; break; /* Released */

             /* Right Arrow: */
             case 0x4E:      x_direction = 1; break; /* Pressed */
             case 0x4E+0x80: x_direction = 0; break; /* Released */

             /* Left Arrow: */
             case 0x4F:      x_direction = -1; break; /* Pressed */
             case 0x4F+0x80: x_direction = 0;  break; /* Released */
           }
           break;
  }
}



/* Change the x/y position: */
x += x_direction;
y += y_direction;

/* Check that the sprite does not move outside the screen: */
if(x > 320)
  x = 320;
if(x < 0)
  x = 0;
if(y > 200)
  y = 200;
if(y < 0)
  y = 0;
```

```
    /* Move the sprite: */
    MoveSprite( 0, &my_sprite, x, y );



    /* Wait for the videobeam to reach the top of the display: (This */
    /* will slow down the animation so the user can see the sprite)  */
    WaitTOF();
  }



  /* Free all allocated memory: (Close the window, libraries etc) */
  free_memory();

  /* THE END */
}



/* This function frees all allocated memory. */
void free_memory()
{
  if( my_sprite.num != -1 )
    FreeSprite( my_sprite.num );

  if( my_window )
    CloseWindow( my_window );

  if( GfxBase )
    CloseLibrary( GfxBase );

  if( IntuitionBase )
    CloseLibrary( IntuitionBase );

  exit();
}
```

Example2

This program shows how to declare and initialize some sprite data and a SimpleSprite structure. It also shows how to reserve a sprite (sprite 2), and how to move it around. The user moves the sprite by pressing the arrow keys. In this example we animate the sprite (6 frames, taken from the arcade game Miniblast).

```
/* Example2                                                        */
/* This program shows how to declare and initialize some sprite data   */
/* and a SimpleSprite structure. It also shows how to reserve a sprite */
/* (sprite 2), and how to move it around. The user moves the sprite by */
/* pressing the arrow keys. In this example we animate the sprite (6   */
/* frames taken from Miniblast).                                       */



#include <intuition/intuition.h>
/* Include this file since you are using sprites: */
#include <graphics/sprite.h>



/* Declare the functions we are going to use: */
void main();
void free_memory();



struct IntuitionBase *IntuitionBase = NULL;
/* We need to open the Graphics library since we are using sprites: */
struct GfxBase *GfxBase = NULL;



/* Declare a pointer to a Window structure: */
struct Window *my_window = NULL;

/* Declare and initialize your NewWindow structure: */
struct NewWindow my_new_window=
{
  50,            /* LeftEdge    x position of the window. */
  25,            /* TopEdge     y positio of the window. */
  320,           /* Width       320 pixels wide. */
  100,           /* Height      100 lines high. */
  0,             /* DetailPen   Text should be drawn with colour reg. 0 */
  1,             /* BlockPen    Blocks should be drawn with colour reg. 1 */
  CLOSEWINDOW|   /* IDCMPFlags  The window will give us a message if the */
  RAWKEY,        /*             user has selected the Close window gad, */
                 /*             or if the user has pressed a key. */
  SMART_REFRESH| /* Flags       Intuition should refresh the window. */
  WINDOWCLOSE|   /*             Close Gadget. */
  WINDOWDRAG|    /*             Drag gadget. */
  WINDOWDEPTH|   /*             Depth arrange Gadgets. */
  WINDOWSIZING|  /*             Sizing Gadget. */
  ACTIVATE,      /*             The window should be Active when opened. */
  NULL,          /* FirstGadget No Custom gadgets. */
  NULL,          /* CheckMark   Use Intuition's default CheckMark. */
  "MICROBLAST",  /* Title       Title of the window. */
  NULL,          /* Screen      Connected to the Workbench Screen. */
  NULL,          /* BitMap      No Custom BitMap. */
  80,            /* MinWidth    We will not allow the window to become */
  30,            /* MinHeight   smaller than 80 x 30, and not bigger */
  300,           /* MaxWidth    than 300 x 200. */
  200,           /* MaxHeight */
  WBENCHSCREEN   /* Type        Connected to the Workbench Screen. */
};
```

```
/********************************************************************/
/* Extra information:                                               */
/* When we declare the window pointer, the intuition library pointer */
/* etc, we initialize them to point to NULL:                        */
/* struct Window *my_window = NULL;                                 */
/* Since we then know that all of the pointers will point to NULL    */
/* when we start, we can check if they still point to NULL when we   */
/* quit. If they do not point to NULL anymore, we close that window, */
/* library etc.                                                     */
/********************************************************************/



/********************************************************/
/* 1. Declare and initialize some sprite graphics data: */
/********************************************************/
/* Sprite data for a ship: */
/* (6 frames, 4 different images: 1 2 3 4 3 2) */
UWORD chip ship_data[6][28]=
{
  {
          0x0000, 0x0000, /* Ship 1 */

          0xFFF8, 0x0000,
          0x0200, 0x0000,
          0x877C, 0x0000,
          0x8786, 0x027C,
          0xBFBF, 0x02C6,
          0xEDFF, 0x1AC2,
          0xA57D, 0x1AFE,
          0xBF19, 0x02FE,
          0x8F12, 0x00FC,
          0x04FC, 0x0000,
          0x0809, 0x0000,
          0x3FFE, 0x0000,

          0x0000, 0x0000,
  },
  {
          0x0000, 0x0000, /* Ship 2 */

          0x7FF0, 0x0000,
          0x0200, 0x0000,
          0x077C, 0x0000,
          0x8786, 0x027C,
          0xBFBF, 0x02C6,
          0xEDFF, 0x1AC2,
          0xA57D, 0x1AFE,
          0xBF19, 0x02FE,
          0x0F12, 0x00FC,
          0x04FC, 0x0000,
          0x0809, 0x0000,
          0x3FFE, 0x0000,

          0x0000, 0x0000,
  },
  {
          0x0000, 0x0000, /* Ship 3 */

          0x3FE0, 0x0000,
          0x0200, 0x0000,
          0x877C, 0x0000,
        0x8786, 0x027C,
        0xBFBF, 0x02C6,
        0xEDFF, 0x1AC2,
        0xA57D, 0x1AFE,
        0xBF19, 0x02FE,
        0x8F12, 0x00FC,
        0x04FC, 0x0000,
        0x0809, 0x0000,
        0x3FFE, 0x0000,

        0x0000, 0x0000,
},
{
        0x0000, 0x0000, /* Ship 4 */

        0x1FC0, 0x0000,
        0x0200, 0x0000,
        0x077C, 0x0000,
        0x8786, 0x027C,
        0xBFBF, 0x02C6,
        0xEDFF, 0x1AC2,
        0xA57D, 0x1AFE,
        0xBF19, 0x02FE,
        0x0F12, 0x00FC,
        0x04FC, 0x0000,
        0x0809, 0x0000,
        0x3FFE, 0x0000,

        0x0000, 0x0000,
},
{
        0x0000, 0x0000, /* Ship 5 (3) */

        0x3FE0, 0x0000,
        0x0200, 0x0000,
        0x877C, 0x0000,
        0x8786, 0x027C,
        0xBFBF, 0x02C6,
        0xEDFF, 0x1AC2,
        0xA57D, 0x1AFE,
        0xBF19, 0x02FE,
        0x8F12, 0x00FC,
        0x04FC, 0x0000,
        0x0809, 0x0000,
        0x3FFE, 0x0000,

        0x0000, 0x0000,
},
{
        0x0000, 0x0000, /* Ship 6 (2) */

        0x7FF0, 0x0000,
        0x0200, 0x0000,
        0x077C, 0x0000,
        0x8786, 0x027C,
        0xBFBF, 0x02C6,
        0xEDFF, 0x1AC2,
        0xA57D, 0x1AFE,
        0xBF19, 0x02FE,
        0x0F12, 0x00FC,
        0x04FC, 0x0000,
        0x0809, 0x0000,
```

```
          0x3FFE, 0x0000,

          0x0000, 0x0000,
  }
};



/********************************************************/
/* 2. Declare and initialize a SimpleSprite structure: */
/********************************************************/
struct SimpleSprite my_sprite=
{
  ship_data[0],   /* posctldata, pointer to the sprite data. (Frame 0) */
  12,             /* height, 12 lines tall. */
  40, 80,         /* x, y, position on the screen. */
  -1,             /* num, this field is automatically initialized when  */
                  /* you call the GetSprite() function, so we set it to */
                  /* -1 for the moment.                                 */
};



void main()
{
  /* Sprite position: */
  WORD x = my_sprite.x;
  WORD y = my_sprite.y;

  /* Direction of the sprite: */
  WORD x_direction = 0;
  WORD y_direction = 0;

  UWORD frame = 0; /* Frame 0 */

  /* Boolean variable used for the while loop: */
  BOOL close_me = FALSE;

  ULONG class; /* IDCMP */
  USHORT code; /* Code */

  /* Declare a pointer to an IntuiMessage structure: */
  struct IntuiMessage *my_message;



  /* Open the Intuition Library: */
  IntuitionBase = (struct IntuitionBase *)
    OpenLibrary( "intuition.library", 0 );

  if( IntuitionBase == NULL )
    free_memory(); /* Could NOT open the Intuition Library! */



  /* Since we are using sprites we need to open the Graphics Library: */
  /* Open the Graphics Library: */
  GfxBase = (struct GfxBase *)
    OpenLibrary( "graphics.library", 0);

  if( GfxBase == NULL )
    free_memory(); /* Could NOT open the Graphics Library! */
```

```
/* We will now try to open the window: */
my_window = (struct Window *) OpenWindow( &my_new_window );

/* Have we opened the window succesfully? */
if(my_window == NULL)
  free_memory(); /* Could NOT open the Window! */



/* Change the colour register 21 - 23: */
SetRGB4( &my_window->WScreen->ViewPort, 21, 0x0, 0x0, 0x0 ); /* Black */
SetRGB4( &my_window->WScreen->ViewPort, 22, 0x0, 0x8, 0x0 ); /* DGreen */
SetRGB4( &my_window->WScreen->ViewPort, 23, 0x0, 0xD, 0x0 ); /* Green */



/********************************/
/* 3. Try to reserve sprite 2: */
/********************************/
if( GetSprite( &my_sprite, 2 ) != 2 )
  free_memory(); /* Could not reserve sprite number 2. */



/* Stay in the while loop until the user has selected the Close window */
/* gadget: */
while( close_me == FALSE )
{
  /* Stay in the while loop as long as we can collect messages */
  /* sucessfully: */
  while(my_message = (struct IntuiMessage *) GetMsg(my_window->UserPort))
  {
    /* After we have collected the message we can read it, and save any */
    /* important values which we maybe want to check later: */
    class = my_message->Class;
    code  = my_message->Code;


    /* After we have read it we reply as fast as possible: */
    /* REMEMBER! Do never try to read a message after you have replied! */
    /* Some other process has maybe changed it. */
    ReplyMsg( my_message );


    /* Check which IDCMP flag was sent: */
    switch( class )
    {
      case CLOSEWINDOW:     /* Quit! */
             close_me=TRUE;
             break;

      case RAWKEY:          /* A key was pressed! */
             /* Check which key was pressed: */
             switch( code )
             {
               /* Up Arrow: */
               case 0x4C:      y_direction = -1; break; /* Pressed */
               case 0x4C+0x80: y_direction = 0;  break; /* Released */
```

```
               /* Down Arrow: */
               case 0x4D:      y_direction = 1; break; /* Pressed */
               case 0x4D+0x80: y_direction = 0; break; /* Released */

               /* Right Arrow: */
               case 0x4E:      x_direction = 1; break; /* Pressed */
               case 0x4E+0x80: x_direction = 0; break; /* Released */

               /* Left Arrow: */
               case 0x4F:      x_direction = -1; break; /* Pressed */
               case 0x4F+0x80: x_direction = 0;  break; /* Released */
             }
             break;
    }
  }



  /* Change the x/y position: */
  x += x_direction;
  y += y_direction;

  /* Check that the sprite does not move outside the screen: */
  if(x > 320)
    x = 320;
  if(x < 0)
    x = 0;
  if(y > 200)
    y = 200;
  if(y < 0)
    y = 0;

  /* Move the sprite: */
  MoveSprite( 0, &my_sprite, x, y );



  /* Change frame: */
  frame++;

  /* 6 frames: */
  if( frame > 5 )
    frame = 0;

  /* Change the sprite data: */
  ChangeSprite( 0, &my_sprite, ship_data[ frame ] );



  /* Wait for the videobeam to reach the top of the display: (This */
  /* will slow down the animation so the user can see the sprite)  */
  /* (If you want to have some "action" you can take it away...) */
  WaitTOF();
}



/* Free all allocated memory: (Close the window, libraries etc) */
free_memory();

/* THE END */
}
```

```
/* This function frees all allocated memory. */
void free_memory()
{
  if( my_sprite.num != -1 )
    FreeSprite( my_sprite.num );

  if( my_window )
    CloseWindow( my_window );

  if( GfxBase )
    CloseLibrary( GfxBase );

  if( IntuitionBase )
    CloseLibrary( IntuitionBase );

  exit();
}
```

Example3

This program shows how to set up a 15 coloured sprite, and how to move it around.

```
/* Example3                                                 */
/* This program shows how to set up a 15 coloured sprite, and how to */
/* move it around.                                          */



#include <intuition/intuition.h>
/* Include this file since you are using sprites: */
#include <graphics/sprite.h>



/* Declare the functions we are going to use: */
void main();
void free_memory();



struct IntuitionBase *IntuitionBase = NULL;
/* We need to open the Graphics library since we are using sprites: */
struct GfxBase *GfxBase = NULL;



/* Declare a pointer to a Window structure: */
struct Window *my_window = NULL;

/* Declare and initialize your NewWindow structure: */
struct NewWindow my_new_window=
{
  50,            /* LeftEdge    x position of the window. */
  25,            /* TopEdge     y positio of the window. */
  320,           /* Width       320 pixels wide. */
  100,           /* Height      100 lines high. */
  0,             /* DetailPen   Text should be drawn with colour reg. 0 */
  1,             /* BlockPen    Blocks should be drawn with colour reg. 1 */
  CLOSEWINDOW|   /* IDCMPFlags  The window will give us a message if the */
  RAWKEY,        /*             user has selected the Close window gad, */
                 /*             or if the user has pressed a key. */
  SMART_REFRESH| /* Flags       Intuition should refresh the window. */
  WINDOWCLOSE|   /*             Close Gadget. */
  WINDOWDRAG|    /*             Drag gadget. */
  WINDOWDEPTH|   /*             Depth arrange Gadgets. */
  WINDOWSIZING|  /*             Sizing Gadget. */
  ACTIVATE,      /*             The window should be Active when opened. */
  NULL,          /* FirstGadget No Custom gadgets. */
  NULL,          /* CheckMark   Use Intuition's default CheckMark. */
  "15-COLOURED SPRITE",/* Title Title of the window. */
  NULL,          /* Screen      Connected to the Workbench Screen. */
  NULL,          /* BitMap      No Custom BitMap. */
  80,            /* MinWidth    We will not allow the window to become */
  30,            /* MinHeight   smaller than 80 x 30, and not bigger */
  300,           /* MaxWidth    than 300 x 200. */
  200,           /* MaxHeight */
  WBENCHSCREEN   /* Type        Connected to the Workbench Screen. */
};



/*********************************************************/
/* 1. Declare and initialize some sprite graphics data: */
/*********************************************************/
```

```
/* Sprite Data for the Bottom Sprite: */
UWORD chip bottom_sprite_data[36]=
{
  0x0000, 0x0000,

  /* Bitplane */
  /* ZERO ONE */

  0x0000, 0x0000,
  0xFFFF, 0x0000,
  0x0000, 0xFFFF,
  0xFFFF, 0xFFFF,

  0x0000, 0x0000,
  0xFFFF, 0x0000,
  0x0000, 0xFFFF,
  0xFFFF, 0xFFFF,

  0x0000, 0x0000,
  0xFFFF, 0x0000,
  0x0000, 0xFFFF,
  0xFFFF, 0xFFFF,

  0x0000, 0x0000,
  0xFFFF, 0x0000,
  0x0000, 0xFFFF,
  0xFFFF, 0xFFFF,

  0x0000, 0x0000
};

/* Sprite Data for the Top Sprite: */
UWORD chip top_sprite_data[36]=
{
  0x0000, SPRITE_ATTACHED, /* We attach the Top Sprite to the Bottom */
                           /* Sprite.                                */

  /*  Bitplane  */
  /* TWO  THREE */
  0x0000, 0x0000,
  0x0000, 0x0000,
  0x0000, 0x0000,
  0x0000, 0x0000,

  0xFFFF, 0x0000,
  0xFFFF, 0x0000,
  0xFFFF, 0x0000,
  0xFFFF, 0x0000,

  0x0000, 0xFFFF,
  0x0000, 0xFFFF,
  0x0000, 0xFFFF,
  0x0000, 0xFFFF,

  0xFFFF, 0xFFFF,
  0xFFFF, 0xFFFF,
  0xFFFF, 0xFFFF,
  0xFFFF, 0xFFFF,

  0x0000, 0x0000
};
```

```
/********************************************************/
/* 2. Declare and initialize two SimpleSprite structure: */
/********************************************************/

/* Bottom sprite: */
struct SimpleSprite bottom_sprite=
{
  bottom_sprite_data, /* posctldata, pointer to the sprite data. */
  16,                 /* height, 16 lines tall. */
  40, 80,             /* x, y, position on the screen. */
  -1,                 /* num, this field is automatically initialized */
                      /* when you call the GetSprite() function, so   */
                      /* we set it to -1 for the moment.              */
};

/* Top sprite: */
struct SimpleSprite top_sprite=
{
  top_sprite_data, /* posctldata, pointer to the sprite data. */
  16,              /* height, 16 lines tall. */
  40, 80,          /* x, y, position on the screen. */
  -1,              /* num, this field is automatically initialized */
                   /* when you call the GetSprite() function, so   */
                   /* we set it to -1 for the moment.              */
};



void main()
{
  /* Sprite position: (We use only one pair of coordinates since the */
  /* two sprites will be attached to each other.)                    */
  WORD x = bottom_sprite.x;
  WORD y = bottom_sprite.y;

  /* Direction of the sprite: */
  WORD x_direction = 0;
  WORD y_direction = 0;

  /* Boolean variable used for the while loop: */
  BOOL close_me = FALSE;

  ULONG class; /* IDCMP */
  USHORT code; /* Code */

  /* Declare a pointer to an IntuiMessage structure: */
  struct IntuiMessage *my_message;



  /* Open the Intuition Library: */
  IntuitionBase = (struct IntuitionBase *)
    OpenLibrary( "intuition.library", 0 );

  if( IntuitionBase == NULL )
    free_memory("Could NOT open the Intuition Library!");
```

```
/* Since we are using sprites we need to open the Graphics Library: */
/* Open the Graphics Library: */
GfxBase = (struct GfxBase *)
  OpenLibrary( "graphics.library", 0);

if( GfxBase == NULL )
  free_memory("Could NOT open the Graphics Library!");



/* We will now try to open the window: */
my_window = (struct Window *) OpenWindow( &my_new_window );

/* Have we opened the window succesfully? */
if(my_window == NULL)
  free_memory("Could NOT open the Window!");



/* Change the colour register 17 - 31: */
/* NOTE! Since we change colour register 17, 18 and 19 we will */
/* change the colour of Intuition's Pointer (Sprite 0). We do  */
/* not bother about that in this Example but you should be     */
/* careful with these three colour registers.                  */
SetRGB4( &my_window->WScreen->ViewPort, 17, 0x0, 0xF, 0x0 );
SetRGB4( &my_window->WScreen->ViewPort, 18, 0x0, 0xD, 0x0 );
SetRGB4( &my_window->WScreen->ViewPort, 19, 0x0, 0xB, 0x0 );
SetRGB4( &my_window->WScreen->ViewPort, 20, 0x0, 0x9, 0x0 );
SetRGB4( &my_window->WScreen->ViewPort, 21, 0x0, 0x7, 0x1 );
SetRGB4( &my_window->WScreen->ViewPort, 22, 0x0, 0x5, 0x3 );
SetRGB4( &my_window->WScreen->ViewPort, 23, 0x0, 0x3, 0x5 );
SetRGB4( &my_window->WScreen->ViewPort, 24, 0x1, 0x1, 0x7 );
SetRGB4( &my_window->WScreen->ViewPort, 25, 0x3, 0x0, 0x5 );
SetRGB4( &my_window->WScreen->ViewPort, 26, 0x5, 0x0, 0x3 );
SetRGB4( &my_window->WScreen->ViewPort, 27, 0x7, 0x0, 0x1 );
SetRGB4( &my_window->WScreen->ViewPort, 28, 0x9, 0x0, 0x0 );
SetRGB4( &my_window->WScreen->ViewPort, 29, 0xB, 0x0, 0x0 );
SetRGB4( &my_window->WScreen->ViewPort, 30, 0xD, 0x0, 0x0 );
SetRGB4( &my_window->WScreen->ViewPort, 31, 0xF, 0x0, 0x0 );



/* Reserve sprite 2 as Bottom Sprite: */
if( GetSprite( &bottom_sprite, 2 ) != 2 )
  free_memory("Could NOT reserve Hardware Sprite 2!"); /* Error! */

/* Reserve sprite 3 as Top Sprite: */
if( GetSprite( &top_sprite, 3 ) != 3 )
  free_memory("Could NOT reserve Hardware Sprite 3!"); /* Error! */



/* We will now move the two sprites so that we can see them: */
/* (After you have reserved a sprite you need to call either */
/* MoveSprite() or ChangeSprite() inorder to display the     */
/* sprite.)                                                  */
MoveSprite( 0, &bottom_sprite, x, y );
MoveSprite( 0, &top_sprite, x, y );



/* Stay in the while loop until the user has selected the Close window */
/* gadget: */
while( close_me == FALSE )
{
  /* Stay in the while loop as long as we can collect messages */
  /* sucessfully: */
  while(my_message = (struct IntuiMessage *) GetMsg(my_window->UserPort))
  {
    /* After we have collected the message we can read it, and save any */
    /* important values which we maybe want to check later: */
    class = my_message->Class;
    code  = my_message->Code;


    /* After we have read it we reply as fast as possible: */
    /* REMEMBER! Do never try to read a message after you have replied! */
    /* Some other process has maybe changed it. */
    ReplyMsg( my_message );


    /* Check which IDCMP flag was sent: */
    switch( class )
    {
      case CLOSEWINDOW:     /* Quit! */
             close_me=TRUE;
             break;

      case RAWKEY:          /* A key was pressed! */
             /* Check which key was pressed: */
             switch( code )
             {
               /* Up Arrow: */
               case 0x4C:      y_direction = -1; break; /* Pressed */
               case 0x4C+0x80: y_direction = 0;  break; /* Released */

               /* Down Arrow: */
               case 0x4D:      y_direction = 1; break; /* Pressed */
               case 0x4D+0x80: y_direction = 0; break; /* Released */

               /* Right Arrow: */
               case 0x4E:      x_direction = 1; break; /* Pressed */
               case 0x4E+0x80: x_direction = 0; break; /* Released */

               /* Left Arrow: */
               case 0x4F:      x_direction = -1; break; /* Pressed */
               case 0x4F+0x80: x_direction = 0;  break; /* Released */
             }
             break;
    }
  }



  /* Change the x/y position: */
  x += x_direction;
  y += y_direction;

  /* Check that the sprite does not move outside the screen: */
  if(x > 320)
    x = 320;
  if(x < 0)
    x = 0;
  if(y > 200)
    y = 200;
  if(y < 0)
    y = 0;



  /* Move the bottom sprite: */
  /* IMPORTANT! If you move the Bottom Sprite the Top Sprite will      */
  /* automatically be moved too. However, if you move the Top Sprite   */
  /* the Bottom Sprite will not be moved, and the Attach function will */
  /* not work any more. (You then get two 3-coloured sprites.)         */
  MoveSprite( 0, &bottom_sprite, x, y );



  /* Wait for the videobeam to reach the top of the display: (This */
  /* will slow down the animation so the user can see the sprite)  */
  /* (If you want to have some "action" you can take it away...)   */
  WaitTOF();
}



/* Free all allocated memory: (Close the window, libraries etc) */
free_memory("THE END");

/* THE END */
}



/* This function frees all allocated memory. */
void free_memory( message )
STRPTR message;
{
  printf( "%s\n", message );

  if( bottom_sprite.num != -1 )
    FreeSprite( bottom_sprite.num );

  if( top_sprite.num != -1 )
    FreeSprite( top_sprite.num );

  if( my_window )
    CloseWindow( my_window );

  if( GfxBase )
    CloseLibrary( GfxBase );

  if( IntuitionBase )
    CloseLibrary( IntuitionBase );

  exit();
}
```

A.11 AMIGADOS

Example1

This program collects ten integer values from the user, and saves them in a file ("HighScore.dat") on the RAM disk. The memory is then cleared, and the file cursor is moved to the beginning of the file. The file is then loaded into the memory again, and printed out.

```
/* Example1                                                        */
/* This program collects ten integer values from the user, and saves    */
/* them in a file ("HighScore.dat") on the RAM disk. The memory is then */
/* cleared, and the file cursor is moved to the beginning of the file.  */
/* The file is then loaded into the memory again, and printed out.      */



#include <libraries/dos.h>


void main();

void main()
{
  struct FileHandle *file_handle;
  int highscore[ 10 ];
  long bytes_written;
  long bytes_read;
  int loop;



  /* Let the user enter ten integer values: */
  for( loop=0; loop < 10; loop++ )
  {
    printf("Highscore[%d]: ", loop );
    scanf("%d", &highscore[ loop ] );
  }



  /* Try to open file "HighScore.dat" as a new file:  */
  /* (If the file does not exist, it will be created. */
  /* If it, on the the other hand, exist, it will be  */
  /* overwritten.)                                    */
  file_handle = (struct FileHandle *)
    Open( "RAM:HighScore.dat", MODE_NEWFILE );

  /* Have we opened the file successfully? */
  if( file_handle == NULL )
  {
    printf("Could not open the file!\n");
    exit();
  }



  /* We have now opened a file, and are ready to start writing: */
  bytes_written = Write( file_handle, highscore, sizeof( highscore ) );

  if( bytes_written != sizeof( highscore ) )
  {
    printf("Could not save the Highscore list!\n");
    Close( file_handle );
    exit();
  }
  else
    printf("Highscore saved successfully!\n");
```

```
  printf("Memory cleared!\n");

  for( loop=0; loop < 10; loop++ )
    highscore[ loop ] = 0;



  printf("Loading Highscore!\n");

  Seek( file_handle, 0, OFFSET_BEGINNING );

  bytes_read = Read( file_handle, highscore, sizeof( highscore ) );

  if( bytes_written != sizeof( highscore ) )
  {
    printf("Could not read the Highscore list!\n");
    Close( file_handle );
    exit();
  }



  /* Print out the numbers: */
  for( loop=0; loop < 10; loop++ )
    printf("Highscore[%d] = %5d\n", loop, highscore[ loop ] );



  /* Close the file: */
  Close( file_handle );
}
```

Example2

This example demonstrates how to create a directory called "MyDirectory" on the RAM disk.

```
/* Example 2                                              */
/* This example demonstrates how to create a directory called */
/* "MyDirectory" on the RAM disk.                         */


#include <libraries/dos.h>


void main();

void main()
{
  /* Declare a FileLock structure: */
  struct FileLock *lock;

  /* Create a directory on the RAM disk: (The directory will */
  /* be locked with an exclusive lock, and must therefore be */
  /* unlocked before the program terminates.)                */
  lock = (struct FileLock *) CreateDir( "RAM:MyDirectory" );

  /* If there is no lock, no directory has been created. In  */
  /* that case, inform the user about the problem and leave: */
  if( lock == NULL )
  {
    printf( "ERROR Could NOT create the new directory!\n" );
    exit( 0 );
  }

  /* Unlock the directory: */
  UnLock( lock );
}
```

Example3

This example demonstrates how to rename files and directories. It will rename the file Example 1 created (called "HighScore.dat") to "Numbers.dat". It will also rename the directory Example 2 created ("MyDirectory") to "NewDirectory".

```
/* Example 3                                                          */
/* This example demonstrates how to rename files and directories. It  */
/* will rename the file Example 1 created (called "HighScore.dat") to */
/* "Numbers.dat". It will also rename the directory Example 2 created */
/* ("MyDirectory") to "NewDirectory".                                 */


/* This file declares the type BOOL: */
#include <exec/types.h>


void main();

void main()
{
  BOOL ok;


  /* Rename the file: */
  ok = Rename( "RAM:HighScore.dat", "RAM:Numbers.dat" );

  /* Check if the file was successfully renamed: */
  if( !ok )
    printf( "The file could not be renamed!\n" );


  /* Rename the directory: */
  ok = Rename( "RAM:MyDirectory", "RAM:NewDirectory" );

  /* Check if the directory was successfully renamed: */
  if( !ok )
    printf( "The directory could not be renamed!\n" );
}
```

Example4

This example demonstrates how to delete files and directories. It will delete the file Example 1 and directory Example 2 created. (The file and directory are supposed to have been renamed by Example 3.)

```
/* Example 4                                                         */
/* This example demonstrates how to delete files and directories. It */
/* will delete the file Example 1 and directory Example 2 created.   */
/* (The file and directory are supposed to have been renamed by      */
/* Example 3.)                                                       */


/* This file declares the type BOOL: */
#include <exec/types.h>


void main();

void main()
{
  BOOL ok;


  /* Delete the file: */
  ok = DeleteFile( "RAM:Numbers.dat" );

  /* Check if the file was successfully deleted: */
  if( !ok )
    printf( "The file could not be deleted!\n" );


  /* Delete the directory: */
  ok = DeleteFile( "RAM:NewDirectory" );

  /* Check if the directory was successfully deleted: */
  if( !ok )
    printf( "The directory could not be deleted!\n" );
}
```

Example5

This example demonstrates how to attach a short comment to a file. A short file called "Letter.doc" will be created, and a short comment will be attached. To see the comment use the CLI command "List".

```
/* Example5                                                        */
/* This example demonstrates how to attach a short comment to a file. */
/* A short file called "Letter.doc" will be created, and a short      */
/* comment will be attached.                                          */
/* To see the comment use the CLI command "List".                     */

/* Declares BOOL: */
#include <exec/types.h>
/* Declares the FileHandle structure: */
#include <libraries/dos.h>


void main();

void main()
{
  struct FileHandle *file_handle;
  char letter[ 8 ] = { 'D', 'e', 'a', 'r', ' ', 'S', 'i', 'r' };
  long bytes_written;
  BOOL ok;


  /* Try to open file "Letter.doc" as a new file:     */
  /* (If the file does not exist, it will be created. */
  /* If it, on the the other hand, exist, it will be  */
  /* overwritten.)                                    */
  file_handle = (struct FileHandle *)
    Open( "RAM:Letter.doc", MODE_NEWFILE );

  /* Have we opened the file successfully? */
  if( file_handle == NULL )
  {
    printf( "Could not open the file!\n" );
    exit();
  }


  /* We have now opened a file, and are ready to start writing: */
  bytes_written = Write( file_handle, letter, sizeof( letter ) );

  if( bytes_written != sizeof( letter ) )
  {
    printf( "Could not save the docummment!\n" );
    exit();
  }
  else
    printf( "The docummment was successfully saved!\n" );


  /* Attach a short comment: */
  ok = SetComment( "RAM:Letter.doc", "A very short letter" );

  /* Check if the comment was successfully attached: */
  if( !ok )
    printf( "Could not attach the comment!\n" );
  else
    printf( "The comment was successfull attached to the file!\n" );

  /* Close the file: */
  Close( file_handle );
}
```

Example6

This example demonstrates how to protect and unprotect files. The file Example 5 created ("Letter.doc") will be protected, and we will then try to delete it (unsuccessfully). We will then unprotect the file and then try to delete it (successfully).

```
/* Example6                                                   */
/* This example demonstrates how to protect and unprotect files. */
/* The file Example 5 created ("Letter.doc") will be protected,  */
/* and we will then try to delete it (unsuccessfully). We will   */
/* then unprotect the file and then try to delete it             */
/* (successfully).                                               */


/* Declares BOOL: */
#include <exec/types.h>
/* Declares the FileHandle structure: */
#include <libraries/dos.h>


void main();

void main()
{
  BOOL ok;


  /* Protect the file: */
  ok = SetProtection( "RAM:Letter.doc", FIBF_DELETE );

  /* Check if the file was successfully protected: */
  if( !ok )
    printf( "Could not protect the file!\n" );


  /* Try to delete the file: */
  ok = DeleteFile( "RAM:Letter.doc" );

  /* Check if the file was successfully deleteted: */
  if( !ok )
    printf( "Could not delete the file!\n" );
  else
    printf( "File deleted!\n" );



  /* Unprotect the file: */
  ok = SetProtection( "RAM:Letter.doc", NULL );

  /* Check if the file was successfully unprotected: */
  if( !ok )
    printf( "Could not unprotect the file!\n" );


  /* Try to delete the file: */
  ok = DeleteFile( "RAM:Letter.doc" );

  /* Check if the file was successfully deleteted: */
  if( !ok )
    printf( "Could not delete the file!\n" );
  else
    printf( "File deleted!\n" );
}
```

Example7

This program takes a file/directory/device name as parameter, and prints out some interesting information about it.

```c
/* Example7                                                     */
/* This program takes a file/directory/device name as parameter, and */
/* prints out some interesting information about it.            */


#include <libraries/dos.h>
#include <exec/memory.h>

main( argc, argv )
int argc;
char *argv[];
{
  struct FileLock *lock;
  struct FileInfoBlock *fib_ptr; /* Declare a FileInfoBlock */
                              /* pointer called fib_ptr. */

  if( argc < 2 )
  {
    /* No file/directory specified! */
    printf("What file/directory do you actually want to examine?\n");
    exit();
  }


  /* 1. Allocate enough memory for a FileInfoBlock structure:       */
  /* (Here is some casting again. AllocMem() returns a CPTR memory  */
  /* pointer, while fib_ptr is a pointer to a FileInfoBlock. It is  */
  /* actually the same thing, but to not make the compiler upset we */
  /* tell it that AllocMem() returns a pointer to a FileInfoBlock.) */
  fib_ptr = (struct FileInfoBlock *)
            AllocMem( sizeof( struct FileInfoBlock ),
                      MEMF_PUBLIC | MEMF_CLEAR );

  /* MEMF_PUBLIC: Any type of memory (chip/fast) */
  /* MEMF_CLEAR:  Clear the allocated memory.    */

  /* Check if we have allocated the memory successfully: */
  if( fib_ptr == NULL )
  {
    printf("Not enough memory!\n");
    exit();
  };


  /* 2. Try to lock the file: */
  /* (Casting again! We tell the compiler that Lock() returns a pointer */
  /* to a FileLock structure.)                                          */
  lock = (struct FileLock *) Lock( argv[ 1 ], SHARED_LOCK );

  /* Colud we lock the file? */
  if( lock == NULL )
  {
    printf("Could not lock the file/directory!\n");

    /* Deallocate the memory we have allocated: */
    FreeMem( fib_ptr, sizeof( struct FileInfoBlock ) );

    exit();
  }

  /* 3. Try to get some information about the file: */
  if( Examine( lock, fib_ptr ) == NULL )
  {
    printf("Could not examine the file/directory!\n");

    /* Deallocate the memory we have allocated: */
    FreeMem( fib_ptr, sizeof( struct FileInfoBlock ) );

    /* Unlock the file: */
    UnLock( lock );

    exit();
  }

  /* 4. You may now examine the FileInfoBlock structure! */

  if( fib_ptr->fib_DirEntryType < 0 )
    printf("Type:       File\n");
  else
    printf("Type:       Directory\n");

  printf("Name:       %s\n", fib_ptr->fib_FileName );
  printf("Size:       %d\n", fib_ptr->fib_Size );
  printf("Blocks:     %d\n", fib_ptr->fib_NumBlocks );
  printf("Comment:    %s\n",
    fib_ptr->fib_Comment[0] != '\0' ? fib_ptr->fib_Comment : "No comment" );

  printf("Deletable:  %s\n",
    fib_ptr->fib_Protection & FIBF_DELETE ? "On" : "Off" );

  printf("Executable: %s\n",
    fib_ptr->fib_Protection & FIBF_EXECUTE ? "On" : "Off" );

  printf("Writable:   %s\n",
    fib_ptr->fib_Protection & FIBF_WRITE ? "On" : "Off" );

  printf("Readable:   %s\n",
    fib_ptr->fib_Protection & FIBF_READ ? "On" : "Off" );

  printf("Archive:    %s\n",
    fib_ptr->fib_Protection & FIBF_ARCHIVE ? "On" : "Off" );

  printf("Pure:       %s\n",
    fib_ptr->fib_Protection & FIBF_PURE ? "On" : "Off" );

  printf("Script:     %s\n",
    fib_ptr->fib_Protection & FIBF_SCRIPT ? "On" : "Off" );

  printf("Days:       %d\n", fib_ptr->fib_Date.ds_Days );
  printf("Minutes:    %d\n", fib_ptr->fib_Date.ds_Minute );
  printf("Ticks:      %d\n", fib_ptr->fib_Date.ds_Tick );



  /* 5. Unlock the file: */
  UnLock( lock );


  /* 6. Deallocate the memory we have allocated: */
  FreeMem( fib_ptr, sizeof( struct FileInfoBlock ) );
}
```

Example8

This program takes a directory/device name as parameter, and prints out all the file/directory-names inside it. This example describes how to use Examine() and ExNext().

```
/* Example8                                                     */
/* This program takes a directory/device name as parameter, and    */
/* prints out all the file/directory-names inside it. This example */
/* describes how to use Examine() and ExNext().                    */


#include <libraries/dos.h>
#include <exec/memory.h>


main( argc, argv )
int argc;
char *argv[];
{
  struct FileLock *lock;
  struct FileInfoBlock *fib_ptr; /* Declare a FileInfoBlock */
                                 /* pointer called fib_ptr. */


  if( argc < 2 )
  {
    /* No directory/device specified! */
    printf("Which directory/device do you actually want to examine?\n");
    exit();
  }



  /* Allocate enough memory for a FileInfoBlock structure: */
  fib_ptr = (struct FileInfoBlock *)
            AllocMem( sizeof( struct FileInfoBlock ),
                      MEMF_PUBLIC | MEMF_CLEAR );

  /* Check if we have allocated the memory successfully: */
  if( fib_ptr == NULL )
  {
    printf("Not enough memory!\n");
    exit();
  };



  /* Try to lock the file: */
  lock = (struct FileLock *) Lock( argv[ 1 ], SHARED_LOCK );

  /* Colud we lock the file? */
  if( lock == NULL )
  {
    printf("Could not lock the file/directory!\n");

    /* Deallocate the memory we have allocated: */
    FreeMem( fib_ptr, sizeof( struct FileInfoBlock ) );

    exit();
  }



  /* Try to examine the directory/device/(file): */
  if( Examine( lock, fib_ptr ) )
  {
    /* Check if it is a directory/device: */
    if( fib_ptr->fib_DirEntryType > 0 )
    {
      /* Print out the directory/device name with underlined characters: */
      /* \033[4m : Underline */
      /* \033[0m : Normal    */
      printf("\033[4m%s\033[0m\n", fib_ptr->fib_FileName );



      /* As long as we can examine files/directories we continue: */
      while( ExNext( lock, fib_ptr ) )
      {
        /* If it is a file we print out the name with white characters. */
        /* However, if it is a (sub)directory we use orange:            */
        if( fib_ptr->fib_DirEntryType < 0 )
          printf("%s\n", fib_ptr->fib_FileName ); /* File */
        else
          printf("\033[33m%s\033[31m\n", fib_ptr->fib_FileName ); /* Dir */

        /* \033[33m : Orange (Colour 3) */
        /* \033[31m : White  (Colour 1) */
      }



      /* Check what went wrong. If it was not because there were no more */
      /* files in the directory (ERROR_NO_MORE_ENTRIES), something       */
      /* terrible has happened!                                          */
      if( IoErr() != ERROR_NO_MORE_ENTRIES )
        printf("ERROR WHILE READING!!!\n");
    }
    else
      printf("%s is a file!\n", argv[1] );
  }
  else
    printf("Could not examine %s!\n", argv[ 1 ] );



  /* Unlock the file: */
  UnLock( lock );



  /* Deallocate the memory we have allocated: */
  FreeMem( fib_ptr, sizeof( struct FileInfoBlock ) );
}
```

A.12 LOW LEVEL GRAPHICS ROUTINES

Example1

This example shows how to create your own display, and fill it with a lot of pixels in seven different colours.

```
/* Example 1                                                       */
/* This example shows how to create your own display, and fill it with */
/* a lot of pixels in seven different colours.                     */


#include <intuition/intuition.h>
#include <graphics/gfxbase.h>


#define WIDTH  640 /* 640 pixels wide (high resolution)              */
#define HEIGHT 200 /* 200 lines high (non interlaced NTSC display)   */
#define DEPTH    3 /* 3 BitPlanes should be used, gives eight colours. */
#define COLOURS  8 /* 2^3 = 8                                        */


struct IntuitionBase *IntuitionBase;
struct GfxBase *GfxBase;


struct View my_view;
struct View *my_old_view;
struct ViewPort my_view_port;
struct RasInfo my_ras_info;
struct BitMap my_bit_map;
struct RastPort my_rast_port;


UWORD my_color_table[] =
{
  0x000, /* Colour 0, Black       */
  0x800, /* Colour 1, Red         */
  0xF00, /* Colour 2, Light red   */
  0x080, /* Colour 3, Green       */
  0x0F0, /* Colour 4, Light green */
  0x008, /* Colour 5, Blue        */
  0x00F, /* Colour 6, Light Blue  */
  0xFFF, /* Colour 7, White       */
};


void clean_up();
void main();


void main()
{
  UWORD *pointer;
  int loop;


  /* Open the Intuition library: */
  IntuitionBase = (struct IntuitionBase *)
    OpenLibrary( "intuition.library", 0 );
  if( !IntuitionBase )
    clean_up( "Could NOT open the Intuition library!" );

  /* Open the Graphics library: */
  GfxBase = (struct GfxBase *)
    OpenLibrary( "graphics.library", 0 );
  if( !GfxBase )
    clean_up( "Could NOT open the Graphics library!" );
```

```
  /* Save the current View, so we can restore it later: */
  my_old_view = GfxBase->ActiView;


  /* 1. Prepare the View structure, and give it a pointer to */
  /*    the first ViewPort:                                  */
  InitView( &my_view );
  my_view.ViewPort = &my_view_port;


  /* 2. Prepare the ViewPort structure, and set some important values: */
  InitVPort( &my_view_port );
  my_view_port.DWidth = WIDTH;         /* Set the width.                */
  my_view_port.DHeight = HEIGHT;       /* Set the height.               */
  my_view_port.RasInfo = &my_ras_info; /* Give it a pointer to RasInfo. */
  my_view_port.Modes = HIRES;          /* High resolution.              */


  /* 3. Get a colour map, link it to the ViewPort, and prepare it: */
  my_view_port.ColorMap = (struct ColorMap *) GetColorMap( COLOURS );
  if( my_view_port.ColorMap == NULL )
    clean_up( "Could NOT get a ColorMap!" );

  /* Get a pointer to the colour map: */
  pointer = (UWORD *) my_view_port.ColorMap->ColorTable;

  /* Set the colours: */
  for( loop = 0; loop < COLOURS; loop++ )
    *pointer++ = my_color_table[ loop ];


  /* 4. Prepare the BitMap: */
  InitBitMap( &my_bit_map, DEPTH, WIDTH, HEIGHT );

  /* Allocate memory for the Raster: */
  for( loop = 0; loop < DEPTH; loop++ )
  {
    my_bit_map.Planes[ loop ] = (PLANEPTR) AllocRaster( WIDTH, HEIGHT );
    if( my_bit_map.Planes[ loop ] == NULL )
      clean_up( "Could NOT allocate enough memory for the raster!" );

    /* Clear the display memory with help of the Blitter: */
    BltClear( my_bit_map.Planes[ loop ], RASSIZE( WIDTH, HEIGHT ), 0 );
  }


  /* 5. Prepare the RasInfo structure: */
  my_ras_info.BitMap = &my_bit_map; /* Pointer to the BitMap structure.  */
  my_ras_info.RxOffset = 0;         /* The top left corner of the Raster */
  my_ras_info.RyOffset = 0;         /* should be at the top left corner  */
                                    /* of the display.                   */
  my_ras_info.Next = NULL;          /* Single playfield - only one       */
                                    /* RasInfo structure is necessary.   */

  /* 6. Create the display: */
  MakeVPort( &my_view, &my_view_port );
  MrgCop( &my_view );


  /* 7. Prepare the RastPort, and give it a pointer to the BitMap. */
  InitRastPort( &my_rast_port );
  my_rast_port.BitMap = &my_bit_map;


  /* 8. Show the new View: */
  LoadView( &my_view );


  /* Set the draw mode to JAM1. FgPen's colour will be used. */
  SetDrMd( &my_rast_port, JAM1 );
  /* Draw 10000 pixels in seven different colours, randomly. */
  for( loop = 0; loop < 10000; loop++ )
  {
    /* Set FgPen's colour (1-7, 0 used for the the background). */
    SetAPen( &my_rast_port, rand() % (COLOURS-1) + 1 );
    /* Write a pixel somewere on the display: */
    WritePixel( &my_rast_port, rand() % WIDTH, rand() % HEIGHT );
  }


  /* 9. Restore the old View: */
  LoadView( my_old_view );


  /* Free all allocated resources and leave. */
  clean_up( "THE END" );
}


/* Returns all allocated resources: */
void clean_up( message )
STRPTR message;
{
  int loop;

  /* Free automatically allocated display structures: */
  FreeVPortCopLists( &my_view_port );
  FreeCprList( my_view.LOFCprList );

  /* Deallocate the display memory, BitPlane for BitPlane: */
  for( loop = 0; loop < DEPTH; loop++ )
    if( my_bit_map.Planes[ loop ] )
      FreeRaster( my_bit_map.Planes[ loop ], WIDTH, HEIGHT );

  /* Deallocate the ColorMap: */
  if( my_view_port.ColorMap ) FreeColorMap( my_view_port.ColorMap );

  /* Close the Graphics library: */
  if( GfxBase ) CloseLibrary( GfxBase );

  /* Close the Intuition library: */
  if( IntuitionBase ) CloseLibrary( IntuitionBase );

  /* Print the message and leave: */
  printf( "%s\n", message );
  exit();
}
```

Example2

This example shows how to create a large Raster and a smaller display. We fill the Raster with a lot of pixels in seven different colours and by altering the RxOffset and RyOffset values in the RasInfo structure, the Raster is scrolled in all directions. This method to scroll a large drawing in full speed is used in many games and was even used in my own racing game "Car".

```
/* Example 2                                                         */
/* This example shows how to create a large Raster and a smaller     */
/* display. We fill the Raster with a lot of pixels in seven different  */
/* colours and by altering the RxOffset and RyOffset values in the   */
/* RasInfo structure, the Raster is scrolled in all directions. This */
/* method to scroll a large drawing in full speed is used in many games */
/* and was even used in my own racing game "Car".                    */


#include <intuition/intuition.h>
#include <graphics/gfxbase.h>


#define RWIDTH   450 /* Raster 450 pixels wide.  */
#define RHEIGHT  250 /* Raster 250 lines high.   */

/* The ViewPort is quite small, and is placed in the middle of the View: */
#define DWIDTH   200 /* Display 200 pixels wide. */
#define DHEIGHT  100 /* Display 100 lines high.  */
#define DXOFFSET  60 /* DxOffset 60 pixels.      */
#define DYOFFSET  50 /* DyOffset 50 lines.       */

#define DEPTH      3 /* 3 BitPlanes should be used, gives eight colours. */
#define COLOURS    8 /* 2^3 = 8                                         */

#define SPEED      1 /* How many pixels the Raster should be scrolled */
                     /* every time.                                   */


struct IntuitionBase *IntuitionBase;
struct GfxBase *GfxBase;


struct View my_view;
struct View *my_old_view;
struct ViewPort my_view_port;
struct RasInfo my_ras_info;
struct BitMap my_bit_map;
struct RastPort my_rast_port;


UWORD my_color_table[] =
{
  0x000, /* Colour 0, Black       */
  0x800, /* Colour 1, Red         */
  0xF00, /* Colour 2, Light red   */
  0x080, /* Colour 3, Green       */
  0x0F0, /* Colour 4, Light green */
  0x008, /* Colour 5, Blue        */
  0x00F, /* Colour 6, Light Blue  */
  0xFFF, /* Colour 7, White       */
};


void clean_up();
void main();


void main()
{
  SHORT deltaX = SPEED;
  SHORT deltaY = SPEED;
```

```
UWORD *pointer;
int loop;



/* Open the Intuition library: */
IntuitionBase = (struct IntuitionBase *)
  OpenLibrary( "intuition.library", 0 );
if( !IntuitionBase )
  clean_up( "Could NOT open the Intuition library!" );

/* Open the Graphics library: */
GfxBase = (struct GfxBase *)
  OpenLibrary( "graphics.library", 0 );
if( !GfxBase )
  clean_up( "Could NOT open the Graphics library!" );


/* Save the current View, so we can restore it later: */
my_old_view = GfxBase->ActiView;


/* 1. Prepare the View structure, and give it a pointer to */
/*    the first ViewPort:                                  */
InitView( &my_view );
my_view.ViewPort = &my_view_port;


/* 2. Prepare the ViewPort structure, and set some important values: */
InitVPort( &my_view_port );
my_view_port.DWidth = DWIDTH;        /* Set the width.                */
my_view_port.DHeight = DHEIGHT;      /* Set the height.               */
my_view_port.DxOffset = DXOFFSET;    /* Set the display X offset.     */
my_view_port.DyOffset = DYOFFSET;    /* Set the display Y offset.     */
my_view_port.RasInfo = &my_ras_info; /* Give it a pointer to RasInfo. */
my_view_port.Modes = NULL;           /* Low resolution.               */


/* 3. Get a colour map, link it to the ViewPort, and prepare it: */
my_view_port.ColorMap = (struct ColorMap *) GetColorMap( COLOURS );
if( my_view_port.ColorMap == NULL )
  clean_up( "Could NOT get a ColorMap!" );

/* Get a pointer to the colour map: */
pointer = (UWORD *) my_view_port.ColorMap->ColorTable;

/* Set the colours: */
for( loop = 0; loop < COLOURS; loop++ )
  *pointer++ = my_color_table[ loop ];


/* 4. Prepare the BitMap: */
InitBitMap( &my_bit_map, DEPTH, RWIDTH, RHEIGHT );

/* Allocate memory for the Raster: */
for( loop = 0; loop < DEPTH; loop++ )
{
  my_bit_map.Planes[ loop ] = (PLANEPTR) AllocRaster( RWIDTH, RHEIGHT );
  if( my_bit_map.Planes[ loop ] == NULL )
    clean_up( "Could NOT allocate enough memory for the raster!" );

  /* Clear the display memory with help of the Blitter: */
  BltClear( my_bit_map.Planes[ loop ], RASSIZE( RWIDTH, RHEIGHT ), 0 );
}


/* 5. Prepare the RasInfo structure: */
my_ras_info.BitMap = &my_bit_map; /* Pointer to the BitMap structure.  */
my_ras_info.RxOffset = 0;         /* The top left corner of the Raster */
my_ras_info.RyOffset = 0;         /* should be at the top left corner  */
                                  /* of the display.                   */
my_ras_info.Next = NULL;          /* Single playfield - only one       */
                                  /* RasInfo structure is necessary.   */

/* 6. Create the display: */
MakeVPort( &my_view, &my_view_port );
MrgCop( &my_view );


/* 7. Prepare the RastPort, and give it a pointer to the BitMap. */
InitRastPort( &my_rast_port );
my_rast_port.BitMap = &my_bit_map;


/* 8. Show the new View: */
LoadView( &my_view );


/* Set the draw mode to JAM1. FgPen's colour will be used. */
SetDrMd( &my_rast_port, JAM1 );
/* Draw 10000 pixels in seven different colours, randomly. */
for( loop = 0; loop < 10000; loop++ )
{
  /* Set FgPen's colour (1-7, 0 used for the the background). */
  SetAPen( &my_rast_port, rand() % (COLOURS-1) + 1 );
  /* Write a pixel somewere on the display: */
  WritePixel( &my_rast_port, rand() % RWIDTH, rand() % RHEIGHT );
}


/* Scroll the Raster in all directions for a little while: */
for( loop = 0; loop < 5000; loop++ )
{
  my_ras_info.RxOffset += deltaX;
  my_ras_info.RyOffset += deltaY;

  /* The Raster is moved in one direction until the other side is */
  /* reached were we change the direction:                         */

  /* Have we reached the left side? */
  if( my_ras_info.RxOffset <= 0 )
    deltaX = SPEED;
  /* Have we reached the right (Raster width - Display width) side? */
  if( my_ras_info.RxOffset >= RWIDTH - DWIDTH )
    deltaX = -SPEED;

  /* Have we reached the top side? */
  if( my_ras_info.RyOffset <= 0 )
    deltaY = SPEED;
  /* Have we reached the bottom (Raster height - Display height) side? */
  if( my_ras_info.RyOffset >= RHEIGHT - DHEIGHT )
    deltaY = -SPEED;
```

```
    /* Recalculate the display instructions: (If you change any values */
    /* in the display structures the Amiga have to recalculate the     */
    /* entire display instructions. You must therefore call all three  */
    /* display functions: MakeVPort(), MrgCop() and LoadView().)       */
    MakeVPort( &my_view, &my_view_port );
    MrgCop( &my_view );
    LoadView( &my_view );
  }


  /* 9. Restore the old View: */
  LoadView( my_old_view );


  /* Free all allocated resources and leave. */
  clean_up( "THE END" );
}


/* Returns all allocated resources: */
void clean_up( message )
STRPTR message;
{
  int loop;

  /* Free automatically allocated display structures: */
  FreeVPortCopLists( &my_view_port );
  FreeCprList( my_view.LOFCprList );

  /* Deallocate the display memory, BitPlane for BitPlane: */
  for( loop = 0; loop < DEPTH; loop++ )
    if( my_bit_map.Planes[ loop ] )
      FreeRaster( my_bit_map.Planes[ loop ], RWIDTH, RHEIGHT );

  /* Deallocate the ColorMap: */
  if( my_view_port.ColorMap ) FreeColorMap( my_view_port.ColorMap );

  /* Close the Graphics library: */
  if( GfxBase ) CloseLibrary( GfxBase );

  /* Close the Intuition library: */
  if( IntuitionBase ) CloseLibrary( IntuitionBase );

  /* Print the message and leave: */
  printf( "%s\n", message );
  exit();
}
```

Example3

This example shows how to create a display that covers the entire display. This method is called "Overscan", and is primarily used in video and graphics programs, but can also be used in games etc to make the display more interesting.

```
/* Example 3                                                       */
/* This example shows how to create a display that covers the entire  */
/* display. This method is called "Overscan", and is primarly used in */
/* video and graphics programs, but can also be used in games etc to  */
/* make the display more interesting.                                 */

/* EXTRA INFORMATION                                                  */
/* If you want your programs to work on both American (NTSC) and      */
/* European (PAL) machines you must either:                           */
/* 1. Not make the display taller than 200 lines. The program will    */
/*    then run perfectly on both types of machines, BUT the European  */
/*    user would be very annoyed since the last 56 lines could not    */
/*    be used.                                                        */
/* 2. Look at the GfxBase structure and see if the program is running */
/*    on an American machine, set the height to max 200 lines, or if  */
/*    the program is running on a European machine, set the height    */
/*    to max 256 lines. (For interlaced displays: 400 or 512 lines)   */
/*    Example:  if( GfxBase->DisplayFlags & NTSC )                    */
/*                Height=200;                                         */
/*              if( GfxBase->DisplayFlags & PAL )                     */
/*                Height=256;                                         */


#include <intuition/intuition.h>
#include <graphics/gfxbase.h>


#define WIDTH       352 /* Display 352 pixels wide. [Overscan]       */
#define NTSC_HEIGHT 262 /* Display 262 lines high. [NTSC - Overscan] */
#define PAL_HEIGHT  287 /* Display 287 lines high. [PAL - Overscan]  */

/* The ViewPort should be placed above and more to the    */
/* left than what is normally used:                       */
#define DXOFFSET -16 /* DxOffset -16 pixels.              */
#define DYOFFSET -31 /* DyOffset -31 lines.               */

#define DEPTH      2 /* 2 BitPlanes should be used, gives four colours. */
#define COLOURS    8 /* 2^2 = 4                                          */


struct IntuitionBase *IntuitionBase;
struct GfxBase *GfxBase;


struct View my_view;
struct View *my_old_view;
struct ViewPort my_view_port;
struct RasInfo my_ras_info;
struct BitMap my_bit_map;
struct RastPort my_rast_port;


UWORD my_color_table[] =
{
  0x000, /* Colour 0, Black */
  0xF00, /* Colour 1, Red   */
  0x0F0, /* Colour 2, Green */
  0x00F, /* Colour 3, Blue  */
};


SHORT height;
```

```
void clean_up();
void main();


void main()
{
  UWORD *pointer;
  int loop;


  /* Open the Intuition library: */
  IntuitionBase = (struct IntuitionBase *)
    OpenLibrary( "intuition.library", 0 );
  if( !IntuitionBase )
    clean_up( "Could NOT open the Intuition library!" );

  /* Open the Graphics library: */
  GfxBase = (struct GfxBase *)
    OpenLibrary( "graphics.library", 0 );
  if( !GfxBase )
    clean_up( "Could NOT open the Graphics library!" );


  /* Check if the program is running on a PAL or NTSC machine: */
  if( GfxBase->DisplayFlags & PAL )
  {
    height = PAL_HEIGHT;
    printf( "You have an European (PAL) machine!\n" );
  }
  else
  {
    height = NTSC_HEIGHT;
    printf( "You have an American (NTSC) machine!\n" );
  }


  /* Save the current View, so we can restore it later: */
  my_old_view = GfxBase->ActiView;


  /* 1. Prepare the View structure, and give it a pointer to */
  /*    the first ViewPort:                                  */
  InitView( &my_view );
  my_view.ViewPort = &my_view_port;


  /* 2. Prepare the ViewPort structure, and set some important values:   */
  InitVPort( &my_view_port );
  my_view_port.DWidth = WIDTH;         /* Set the width.                */
  my_view_port.DHeight = height;       /* Set the height.               */
  my_view_port.DxOffset = DXOFFSET;    /* Set the display X offset.     */
  my_view_port.DyOffset = DYOFFSET;    /* Set the display Y offset.     */
  my_view_port.RasInfo = &my_ras_info; /* Give it a pointer to RasInfo. */
  my_view_port.Modes = NULL;           /* Low resolution.               */


  /* 3. Get a colour map, link it to the ViewPort, and prepare it: */
  my_view_port.ColorMap = (struct ColorMap *) GetColorMap( COLOURS );
  if( my_view_port.ColorMap == NULL )
    clean_up( "Could NOT get a ColorMap!" );
```

```
/* Get a pointer to the colour map: */
pointer = (UWORD *) my_view_port.ColorMap->ColorTable;

/* Set the colours: */
for( loop = 0; loop < COLOURS; loop++ )
  *pointer++ = my_color_table[ loop ];


/* 4. Prepare the BitMap: */
InitBitMap( &my_bit_map, DEPTH, WIDTH, height );

/* Allocate memory for the Raster: */
for( loop = 0; loop < DEPTH; loop++ )
{
  my_bit_map.Planes[ loop ] = (PLANEPTR) AllocRaster( WIDTH, height );
  if( my_bit_map.Planes[ loop ] == NULL )
    clean_up( "Could NOT allocate enough memory for the raster!" );

  /* Clear the display memory with help of the Blitter: */
  BltClear( my_bit_map.Planes[ loop ], RASSIZE( WIDTH, height ), 0 );
}


/* 5. Prepare the RasInfo structure: */
my_ras_info.BitMap = &my_bit_map; /* Pointer to the BitMap structure.  */
my_ras_info.RxOffset = 0;         /* The top left corner of the Raster */
my_ras_info.RyOffset = 0;         /* should be at the top left corner  */
                                  /* of the display.                   */
my_ras_info.Next = NULL;          /* Single playfield - only one       */
                                  /* RasInfo structure is necessary.   */

/* 6. Create the display: */
MakeVPort( &my_view, &my_view_port );
MrgCop( &my_view );


/* 7. Prepare the RastPort, and give it a pointer to the BitMap. */
InitRastPort( &my_rast_port );
my_rast_port.BitMap = &my_bit_map;


/* 8. Show the new View: */
LoadView( &my_view );


/* Set the draw mode to JAM1. FgPen's colour will be used. */
SetDrMd( &my_rast_port, JAM1 );
/* Draw 10000 pixels in seven different colours, randomly. */
for( loop = 0; loop < 10000; loop++ )
{
  /* Set FgPen's colour (1-7, 0 used for the the background). */
  SetAPen( &my_rast_port, rand() % (COLOURS-1) + 1 );
  /* Write a pixel somewere on the display: */
  WritePixel( &my_rast_port, rand() % WIDTH, rand() % height );
}


/* 9. Restore the old View: */
LoadView( my_old_view );
```

```
  /* Free all allocated resources and leave. */
  clean_up( "THE END" );
}


/* Returns all allocated resources: */
void clean_up( message )
STRPTR message;
{
  int loop;

  /* Free automatically allocated display structures: */
  FreeVPortCopLists( &my_view_port );
  FreeCprList( my_view.LOFCprList );

  /* Deallocate the display memory, BitPlane for BitPlane: */
  for( loop = 0; loop < DEPTH; loop++ )
    if( my_bit_map.Planes[ loop ] )
      FreeRaster( my_bit_map.Planes[ loop ], WIDTH, height );

  /* Deallocate the ColorMap: */
  if( my_view_port.ColorMap ) FreeColorMap( my_view_port.ColorMap );

  /* Close the Graphics library: */
  if( GfxBase ) CloseLibrary( GfxBase );

  /* Close the Intuition library: */
  if( IntuitionBase ) CloseLibrary( IntuitionBase );

  /* Print the message and leave: */
  printf( "%s\n", message );
  exit();
}
```

Example4

This example demonstrates how to open two different ViewPorts on the same display. The first ViewPort is in low resolution and use 32 colours, while the second ViewPort is in high resolution and only use 2 colours.

```
/* Example 4                                                        */
/* This example demonstrates how to open two different ViewPorts on the */
/* same display. The first ViewPort is in low resolution and use 32     */
/* colours, while the second ViewPort is in high resolution and only    */
/* use 2 colours.                                                       */


#include <intuition/intuition.h>
#include <graphics/gfxbase.h>


/* ViewPort 1 */
#define WIDTH1   320 /* 320 pixels wide.                             */
#define HEIGHT1  150 /* 150 lines high.                              */
#define DEPTH1     5 /* 5 BitPlanes should be used, gives 32 colours. */
#define COLOURS1  32 /* 2^5 = 32                                     */

/* ViewPort 2 */
#define WIDTH2   640 /* 640 pixels wide.                            */
#define HEIGHT2   45 /* 45 lines high.                              */
#define DEPTH2     1 /* 1 BitPlanes should be used, gives 2 colours. */
#define COLOURS2   2 /* 2^1 = 2                                     */


struct IntuitionBase *IntuitionBase;
struct GfxBase *GfxBase;


struct View my_view;
struct View *my_old_view;


/* ViewPort 1 */
struct ViewPort view_port1;
struct RasInfo ras_info1;
struct BitMap bit_map1;
struct RastPort rast_port1;
UWORD color_table1[] =
{
  0x000, 0xFFF, 0xDDD, 0xBBB, 0x999, 0x777, 0x555, 0x333,
  0xF00, 0xD00, 0xB00, 0x900, 0x700, 0x500, 0x300, 0x100,
  0x0F0, 0x0D0, 0x0B0, 0x090, 0x070, 0x050, 0x030, 0x010,
  0x00F, 0x00D, 0x00B, 0x009, 0x007, 0x005, 0x003, 0x001
};


/* ViewPort 2 */
struct ViewPort view_port2;
struct RasInfo ras_info2;
struct BitMap bit_map2;
struct RastPort rast_port2;
UWORD color_table2[] = { 0x000, 0xFFF };

void clean_up();
void main();


void main()
{
  UWORD *pointer;
  int loop;
```

```
  /* Open the Intuition library: */
  IntuitionBase = (struct IntuitionBase *)
    OpenLibrary( "intuition.library", 0 );
  if( !IntuitionBase )
    clean_up( "Could NOT open the Intuition library!" );

  /* Open the Graphics library: */
  GfxBase = (struct GfxBase *)
    OpenLibrary( "graphics.library", 0 );
  if( !GfxBase )
    clean_up( "Could NOT open the Graphics library!" );



  /* Save the current View, so we can restore it later: */
  my_old_view = GfxBase->ActiView;



  /* 1. Prepare the View structure, and give it a pointer to */
  /*    the first ViewPort:                                  */
  InitView( &my_view );
  my_view.ViewPort = &view_port1;



  /* 2. Prepare the ViewPort structures, and set some important values: */

  /* ViewPort 1 */
  InitVPort( &view_port1 );
  view_port1.DWidth = WIDTH1;      /* Set the width.                */
  view_port1.DHeight = HEIGHT1;    /* Set the height.               */
  view_port1.DxOffset = 0;         /* X position.                   */
  view_port1.DyOffset = 0;         /* Y position.                   */
  view_port1.RasInfo = &ras_info1; /* Give it a pointer to RasInfo. */
  view_port1.Modes = NULL;         /* Low resolution.               */
  view_port1.Next = &view_port2;   /* Pointer to next ViewPort.     */

  /* ViewPort 2 */
  InitVPort( &view_port2 );
  view_port2.DWidth = WIDTH2;      /* Set the width.                */
  view_port2.DHeight = HEIGHT2;    /* Set the height.               */
  view_port2.DxOffset = 0;         /* X position.                   */
  view_port2.DyOffset = HEIGHT1+5; /* Y position (5 lines under).   */
  view_port2.RasInfo = &ras_info2; /* Give it a pointer to RasInfo. */
  view_port2.Modes = HIRES;        /* High resolution.              */
  view_port2.Next = NULL;          /* Last ViewPort in the list.    */



  /* 3. Get a colour map, link it to the ViewPort, and prepare it: */

  /* ViewPort 1 */
  view_port1.ColorMap = (struct ColorMap *) GetColorMap( COLOURS1 );
  if( view_port1.ColorMap == NULL )
    clean_up( "Could NOT get a ColorMap!" );
  /* Get a pointer to the colour map: */
  pointer = (UWORD *) view_port1.ColorMap->ColorTable;
  /* Set the colours: */
  for( loop = 0; loop < COLOURS1; loop++ )
```

```
    *pointer++ = color_table1[ loop ];

  /* ViewPort 2 */
  view_port2.ColorMap = (struct ColorMap *) GetColorMap( COLOURS2 );
  if( view_port2.ColorMap == NULL )
    clean_up( "Could NOT get a ColorMap!" );
  /* Get a pointer to the colour map: */
  pointer = (UWORD *) view_port2.ColorMap->ColorTable;
  /* Set the colours: */
  for( loop = 0; loop < COLOURS2; loop++ )
    *pointer++ = color_table2[ loop ];


  /* 4. Prepare the BitMap: */

  /* ViewPort 1 */
  InitBitMap( &bit_map1, DEPTH1, WIDTH1, HEIGHT1 );
  /* Allocate memory for the Raster: */
  for( loop = 0; loop < DEPTH1; loop++ )
  {
    bit_map1.Planes[ loop ] = (PLANEPTR) AllocRaster( WIDTH1, HEIGHT1 );
    if( bit_map1.Planes[ loop ] == NULL )
      clean_up( "Could NOT allocate enough memory for the raster!" );
    /* Clear the display memory with help of the Blitter: */
    BltClear( bit_map1.Planes[ loop ], RASSIZE( WIDTH1, HEIGHT1 ), 0 );
  }

  /* ViewPort 2 */
  InitBitMap( &bit_map2, DEPTH2, WIDTH2, HEIGHT2 );
  /* Allocate memory for the Raster: */
  for( loop = 0; loop < DEPTH2; loop++ )
  {
    bit_map2.Planes[ loop ] = (PLANEPTR) AllocRaster( WIDTH2, HEIGHT2 );
    if( bit_map2.Planes[ loop ] == NULL )
      clean_up( "Could NOT allocate enough memory for the raster!" );
    /* Clear the display memory with help of the Blitter: */
    BltClear( bit_map2.Planes[ loop ], RASSIZE( WIDTH2, HEIGHT2 ), 0 );
  }

  /* 5. Prepare the RasInfo structure: */

  /* ViewPort 1 */
  ras_info1.BitMap = &bit_map1; /* Pointer to the BitMap structure.  */
  ras_info1.RxOffset = 0;       /* The top left corner of the Raster */
  ras_info1.RyOffset = 0;       /* should be at the top left corner  */
                                /* of the display.                   */
  ras_info1.Next = NULL;        /* Single playfield - only one       */
                                /* RasInfo structure is necessary.   */

  /* ViewPort 2 */
  ras_info2.BitMap = &bit_map2; /* Pointer to the BitMap structure.  */
  ras_info2.RxOffset = 0;       /* The top left corner of the Raster */
  ras_info2.RyOffset = 0;       /* should be at the top left corner  */
                                /* of the display.                   */
  ras_info2.Next = NULL;        /* Single playfield - only one       */
                                /* RasInfo structure is necessary.   */



  /* 6. Create the display: */
  MakeVPort( &my_view, &view_port1 ); /* Prepare ViewPort 1 */
  MakeVPort( &my_view, &view_port2 ); /* Prepare ViewPort 2 */
```

```
  MrgCop( &my_view );


  /* 7. Prepare the RastPort, and give it a pointer to the BitMap. */

  /* ViewPort 1 */
  InitRastPort( &rast_port1 );
  rast_port1.BitMap = &bit_map1;

  /* ViewPort 2 */
  InitRastPort( &rast_port2 );
  rast_port2.BitMap = &bit_map2;


  /* 8. Show the new View: */
  LoadView( &my_view );



  /* Set the draw mode to JAM1. FgPen's colour will be used. */
  SetDrMd( &rast_port1, JAM1 );
  SetDrMd( &rast_port2, JAM1 );

  /* Set FgPen's colour to 1 (white). */
  SetAPen( &rast_port2, 1 );
  /* Draw some pixels in the second ViewPort: */
  for( loop = 0; loop < 500; loop++ )
    WritePixel( &rast_port2, rand() % WIDTH2, rand() % HEIGHT2 );

  /* Print some text into the second ViewPort: */
  Move( &rast_port2, 0, 10 );
  Text( &rast_port2, "This text is written on a single high resolution BitMap. The
ViewPort above use ", 80 );
  Move( &rast_port2, 0, 20 );
  Text( &rast_port2, "a 32-colour low resolution BitMap.
", 80 );

  /* Draw 10000 pixels in seven different colours, randomly. */
  for( loop = 0; loop < 10000; loop++ )
  {
    /* Set FgPen's colour (1-31, 0 used for the the background). */
    SetAPen( &rast_port1, rand() % (COLOURS1-1) + 1 );
    /* Write a pixel somewere on the display: */
    WritePixel( &rast_port1, rand() % WIDTH1, rand() % HEIGHT1 );
  }


  /* 9. Restore the old View: */
  LoadView( my_old_view );


  /* Free all allocated resources and leave. */
  clean_up( "THE END" );
}


/* Returns all allocated resources: */
void clean_up( message )
STRPTR message;
{
  int loop;
  /* Free automatically allocated display structures: */
  FreeVPortCopLists( &view_port1 );
  FreeVPortCopLists( &view_port2 );
  FreeCprList( my_view.LOFCprList );

  /* Deallocate the display memory, BitPlane for BitPlane: */
  for( loop = 0; loop < DEPTH1; loop++ )
    if( bit_map1.Planes[ loop ] )
      FreeRaster( bit_map1.Planes[ loop ], WIDTH1, HEIGHT1 );
  for( loop = 0; loop < DEPTH2; loop++ )
    if( bit_map2.Planes[ loop ] )
      FreeRaster( bit_map2.Planes[ loop ], WIDTH2, HEIGHT2 );

  /* Deallocate the ColorMap: */
  if( view_port1.ColorMap ) FreeColorMap( view_port1.ColorMap );
  if( view_port2.ColorMap ) FreeColorMap( view_port2.ColorMap );

  /* Close the Graphics library: */
  if( GfxBase ) CloseLibrary( GfxBase );

  /* Close the Intuition library: */
  if( IntuitionBase ) CloseLibrary( IntuitionBase );

  /* Print the message and leave: */
  printf( "%s\n", message );
  exit();
}
```

Example5

This example demonstrates how to open a ViewPort in interlace mode.

```
/* Example 5                                                         */
/* This example demonstrates how to open a ViewPort in interlace mode. */


#include <intuition/intuition.h>
#include <graphics/gfxbase.h>


#define WIDTH  640 /* 640 pixels wide (high resolution)                */
#define HEIGHT 400 /* 400 lines high (interlaced NTSC display)         */
#define DEPTH    3 /* 3 BitPlanes should be used, gives eight colours. */
#define COLOURS  8 /* 2^3 = 8                                          */


struct IntuitionBase *IntuitionBase;
struct GfxBase *GfxBase;


struct View my_view;
struct View *my_old_view;
struct ViewPort my_view_port;
struct RasInfo my_ras_info;
struct BitMap my_bit_map;
struct RastPort my_rast_port;


UWORD my_color_table[] =
{
  0x000, /* Colour 0, Black       */
  0x800, /* Colour 1, Red         */
  0xF00, /* Colour 2, Light red   */
  0x080, /* Colour 3, Green       */
  0x0F0, /* Colour 4, Light green */
  0x008, /* Colour 5, Blue        */
  0x00F, /* Colour 6, Light Blue  */
  0xFFF, /* Colour 7, White       */
};


void clean_up();
void main();


void main()
{
  UWORD *pointer;
  int loop;


  /* Open the Intuition library: */
  IntuitionBase = (struct IntuitionBase *)
    OpenLibrary( "intuition.library", 0 );
  if( !IntuitionBase )
    clean_up( "Could NOT open the Intuition library!" );

  /* Open the Graphics library: */
  GfxBase = (struct GfxBase *)
    OpenLibrary( "graphics.library", 0 );
  if( !GfxBase )
    clean_up( "Could NOT open the Graphics library!" );
```

```
  /* Save the current View, so we can restore it later: */
  my_old_view = GfxBase->ActiView;


  /* 1. Prepare the View structure, and give it a pointer to */
  /*    the first ViewPort:                                  */
  InitView( &my_view );
  my_view.ViewPort = &my_view_port;
  /* The View should be interlaced: */
  my_view.Modes = LACE;


  /* 2. Prepare the ViewPort structure, and set some important values: */
  InitVPort( &my_view_port );
  my_view_port.DWidth = WIDTH;         /* Set the width.                */
  my_view_port.DHeight = HEIGHT;       /* Set the height.               */
  my_view_port.RasInfo = &my_ras_info; /* Give it a pointer to RasInfo. */
  my_view_port.Modes = HIRES|LACE;     /* High resolution interlace.    */


  /* IMPORTANT! If you want a ViewPort to be interlaced you have to set */
  /* the LACE flag in both the ViewPort as well as in the View          */
  /* structure. If the ViewPort is interlaced but the View is non-      */
  /* interlaced, only every second line in the ViewPort would be drawn. */
  /* If the ViewPort is non-interlaced but the View is interlaced,      */
  /* each line in the ViewPort would be drawn twice.                    */


  /* 3. Get a colour map, link it to the ViewPort, and prepare it: */
  my_view_port.ColorMap = (struct ColorMap *) GetColorMap( COLOURS );
  if( my_view_port.ColorMap == NULL )
    clean_up( "Could NOT get a ColorMap!" );

  /* Get a pointer to the colour map: */
  pointer = (UWORD *) my_view_port.ColorMap->ColorTable;

  /* Set the colours: */
  for( loop = 0; loop < COLOURS; loop++ )
    *pointer++ = my_color_table[ loop ];


  /* 4. Prepare the BitMap: */
  InitBitMap( &my_bit_map, DEPTH, WIDTH, HEIGHT );

  /* Allocate memory for the Raster: */
  for( loop = 0; loop < DEPTH; loop++ )
  {
    my_bit_map.Planes[ loop ] = (PLANEPTR) AllocRaster( WIDTH, HEIGHT );
    if( my_bit_map.Planes[ loop ] == NULL )
      clean_up( "Could NOT allocate enough memory for the raster!" );

    /* Clear the display memory with help of the Blitter: */
    BltClear( my_bit_map.Planes[ loop ], RASSIZE( WIDTH, HEIGHT ), 0 );
  }


  /* 5. Prepare the RasInfo structure: */
  my_ras_info.BitMap = &my_bit_map; /* Pointer to the BitMap structure.  */
  my_ras_info.RxOffset = 0;         /* The top left corner of the Raster */
  my_ras_info.RyOffset = 0;         /* should be at the top left corner  */
                                    /* of the display.                   */
  my_ras_info.Next = NULL;          /* Single playfield - only one       */
```

```
                                  /* RasInfo structure is necessary.   */

  /* 6. Create the display: */
  MakeVPort( &my_view, &my_view_port );
  MrgCop( &my_view );


  /* 7. Prepare the RastPort, and give it a pointer to the BitMap. */
  InitRastPort( &my_rast_port );
  my_rast_port.BitMap = &my_bit_map;


  /* 8. Show the new View: */
  LoadView( &my_view );


  /* Set the draw mode to JAM1. FgPen's colour will be used. */
  SetDrMd( &my_rast_port, JAM1 );

  /* Draw 10000 lines in eight different colours, randomly. */
  /* Position the pen: */
  Move( &my_rast_port, rand() % WIDTH, rand() % HEIGHT );
  for( loop = 0; loop < 10000; loop++ )
  {
    /* Set FgPen's colour (0-7): */
    SetAPen( &my_rast_port, rand() % COLOURS );
    /* Draw a line: */
    Draw( &my_rast_port, rand() % WIDTH, rand() % HEIGHT );
  }


  /* 9. Restore the old View: */
  LoadView( my_old_view );


  /* Free all allocated resources and leave. */
  clean_up( "THE END" );
}


/* Returns all allocated resources: */
void clean_up( message )
STRPTR message;
{
  int loop;

  /* Free automatically allocated display structures: */
  FreeVPortCopLists( &my_view_port );
  FreeCprList( my_view.LOFCprList );
  FreeCprList( my_view.SHFCprList ); /* ! */

  /* An interlaced display use two copper lists (the normal LOF plus   */
  /* the special SHF). When your program closes an interlaced ViewPort */
  /* you must therefore deallocate both lists!                         */


  /* Deallocate the display memory, BitPlane for BitPlane: */
  for( loop = 0; loop < DEPTH; loop++ )
    if( my_bit_map.Planes[ loop ] )
      FreeRaster( my_bit_map.Planes[ loop ], WIDTH, HEIGHT );

  /* Deallocate the ColorMap: */
  if( my_view_port.ColorMap ) FreeColorMap( my_view_port.ColorMap );

  /* Close the Graphics library: */
  if( GfxBase ) CloseLibrary( GfxBase );

  /* Close the Intuition library: */
  if( IntuitionBase ) CloseLibrary( IntuitionBase );

  /* Print the message and leave: */
  printf( "%s\n", message );
  exit();
}
```

Example6

This example demonstrates how to create a ViewPort in dual playfield mode. Playfield 1 use four colours and is placed behind playfield 2 which only use two colours (transparent and grey). Playfield 1 is filled with a lot of dots and is scrolled around while playfield 2 is is not moved and is filled with only five grey rectangles.

```
/* Example 6                                                         */
/* This example demonstrates how to create a ViewPort in dual playfield  */
/* mode. Playfield 1 use four colours and is placed behind playfield 2   */
/* which only use two colours (transparent and grey). Playfield 1 is     */
/* filled with a lot of dots and is scrolled around while playfield 2 is */
/* is not moved and is filled with only five grey rectangles.            */


#include <intuition/intuition.h>
#include <graphics/gfxbase.h>


#define DWIDTH   320 /* Display 320 pixels wide (low resolution).     */
#define DHEIGHT  200 /* Display 200 lines tall (NTSC non interlaced). */

#define RWIDTH1  600 /* 600 pixels wide.                              */
#define RHEIGHT1 300 /* 300 lines high.                               */
#define DEPTH1     2 /* Playfield one should use 2 BitPlanes.         */

#define RWIDTH2  320 /* 320 pixels wide.                              */
#define RHEIGHT2 200 /* 200 lines high.                               */
#define DEPTH2     1 /* Playfield two should use 1 BitPlane.          */

#define COLOURS   10 /* PF1: colours 0-3, PF2: colours 8 and 9. (0-9) */

#define SPEED      1 /* How many pixels the Raster should be scrolled */
                     /* every time.                                   */

#define BOXES      5 /* Draw 5 rectangles in the second playfield.    */


struct IntuitionBase *IntuitionBase;
struct GfxBase *GfxBase;


struct View view;
struct View *old_view;
struct ViewPort view_port;

/* Playfield 1: */
struct RasInfo ras_info1;
struct BitMap bit_map1;
struct RastPort rast_port1;

/* Playfield 2: */
struct RasInfo ras_info2;
struct BitMap bit_map2;
struct RastPort rast_port2;


UWORD color_table[] =
{
  0x000, /* Colour 0, Black       */
  0xF00, /* Colour 1, Red         */
  0x0F0, /* Colour 2, Green       */
  0x00F, /* Colour 3, Blue        */
  0x000, /* Colour 4, Not used    */
  0x000, /* Colour 5,   - " -     */
  0x000, /* Colour 6,   - " -     */
  0x000, /* Colour 7,   - " -     */
  0x000, /* Colour 8, Transparent */
  0x888  /* Colour 9, Grey        */
};


UWORD box[ BOXES ][ 4 ] =
{
  /*  Minimum  Maximum */
  /*  X    Y    X    Y */
  {    0,   0,  50,  20 },
  {  150,  30, 260,  50 },
  {  290, 100, 319, 150 },
  {  150, 170, 210, 199 },
  {   20,  70,  90, 170 }
};


void clean_up();
void main();


void main()
{
  SHORT deltaX = SPEED;
  SHORT deltaY = SPEED;
  UWORD *pointer;
  int loop;


  /* Open the Intuition library: */
  IntuitionBase = (struct IntuitionBase *)
    OpenLibrary( "intuition.library", 0 );
  if( !IntuitionBase )
    clean_up( "Could NOT open the Intuition library!" );

  /* Open the Graphics library: */
  GfxBase = (struct GfxBase *)
    OpenLibrary( "graphics.library", 0 );
  if( !GfxBase )
    clean_up( "Could NOT open the Graphics library!" );



  /* Save the current View, so we can restore it later: */
  old_view = GfxBase->ActiView;



  /* 1. Prepare the View structure, and give it a pointer to */
  /*    the first ViewPort:                                   */
  InitView( &view );
  view.ViewPort = &view_port;



  /* 2. Prepare the ViewPort structure, and set some important values: */
  InitVPort( &view_port );
  view_port.DWidth = DWIDTH;        /* Set the width.                  */
  view_port.DHeight = DHEIGHT;      /* Set the height.                 */
  view_port.RasInfo = &ras_info1;   /* Give it a pointer to RasInfo.   */
  view_port.Modes = DUALPF|PFBA;    /* Dual playfields, 2 on top of 1. */


  /* 3. Get a colour map, link it to the ViewPort, and prepare it: */
```

```
view_port.ColorMap = (struct ColorMap *) GetColorMap( COLOURS );
if( view_port.ColorMap == NULL )
  clean_up( "Could NOT get a ColorMap!" );

/* Get a pointer to the colour map: */
pointer = (UWORD *) view_port.ColorMap->ColorTable;

/* Set the colours: */
for( loop = 0; loop < COLOURS; loop++ )
  *pointer++ = color_table[ loop ];



/* 4. Prepare the BitMaps: */

/* Playfield 1: */
InitBitMap( &bit_map1, DEPTH1, RWIDTH1, RHEIGHT1 );
/* Allocate memory for the Raster: */
for( loop = 0; loop < DEPTH1; loop++ )
{
  bit_map1.Planes[ loop ] = (PLANEPTR) AllocRaster( RWIDTH1, RHEIGHT1 );
  if( bit_map1.Planes[ loop ] == NULL )
    clean_up( "Could NOT allocate enough memory for the raster!" );

  /* Clear the display memory with help of the Blitter: */
  BltClear( bit_map1.Planes[ loop ], RASSIZE( RWIDTH1, RHEIGHT1 ), 0 );
}

/* Playfield 2: */
InitBitMap( &bit_map2, DEPTH2, RWIDTH2, RHEIGHT2 );
/* Allocate memory for the Raster: */
for( loop = 0; loop < DEPTH2; loop++ )
{
  bit_map2.Planes[ loop ] = (PLANEPTR) AllocRaster( RWIDTH2, RHEIGHT2 );
  if( bit_map2.Planes[ loop ] == NULL )
    clean_up( "Could NOT allocate enough memory for the raster!" );

  /* Clear the display memory with help of the Blitter: */
  BltClear( bit_map2.Planes[ loop ], RASSIZE( RWIDTH2, RHEIGHT2 ), 0 );
}



/* 5. Prepare the RasInfo structures: */

/* Playfield 1: */
ras_info1.BitMap = &bit_map1; /* Pointer to the BitMap structure.  */
ras_info1.RxOffset = 0;       /* The top left corner of the Raster */
ras_info1.RyOffset = 0;       /* should be at the top left corner  */
                              /* of the display.                   */
ras_info1.Next = &ras_info2;  /* Link RasInfo1 to RasInfo2.        */

/* Playfield 2: */
ras_info2.BitMap = &bit_map2; /* Pointer to the BitMap structure.  */
ras_info2.RxOffset = 0;       /* The top left corner of the Raster */
ras_info2.RyOffset = 0;       /* should be at the top left corner  */
                              /* of the display.                   */
ras_info2.Next = NULL;        /* Last RasInfo structure.           */



/* 6. Create the display: */
```

```
MakeVPort( &view, &view_port );
MrgCop( &view );



/* 7. Prepare the RastPorts, and give them a pointer to each BitMap. */

/* Playfield 1: */
InitRastPort( &rast_port1 );
rast_port1.BitMap = &bit_map1;

/* Playfield 2: */
InitRastPort( &rast_port2 );
rast_port2.BitMap = &bit_map2;



/* 8. Show the new View: */
LoadView( &view );



/* Playfield 2: */
/* Set the draw mode to JAM1. FgPen's colour will be used. */
SetDrMd( &rast_port2, JAM1 );
/* Use colour 9 (grey): */
SetAPen( &rast_port2, 9 );
/* Draw five grey boxes: */
for( loop = 0; loop < BOXES; loop++ )
  RectFill( &rast_port2, box[ loop ][ 0 ],
                         box[ loop ][ 1 ],
                         box[ loop ][ 2 ],
                         box[ loop ][ 3 ] );


/* Playfield 1: */
/* Set the draw mode to JAM1. FgPen's colour will be used. */
SetDrMd( &rast_port1, JAM1 );
/* PF1: Draw 5000 pixels in four different colours, randomly. */
for( loop = 0; loop < 5000; loop++ )
{
  /* Set FgPen's colour (0-3): */
  SetAPen( &rast_port1, rand() % 4 );
  /* Write a pixel somewere on the display: */
  WritePixel( &rast_port1, rand() % RWIDTH1, rand() % RHEIGHT1 );
}

/* Scroll the Raster (PF 1) in all directions for a little while: */
for( loop = 0; loop < 5000; loop++ )
{
  ras_info1.RxOffset += deltaX;
  ras_info1.RyOffset += deltaY;

  /* The Raster is moved in one direction until the other side is */
  /* reached were we change the direction:                        */

  /* Have we reached the left side? */
  if( ras_info1.RxOffset <= 0 )
    deltaX = SPEED;
  /* Have we reached the right (Raster width - Display width) side? */
  if( ras_info1.RxOffset >= RWIDTH1 - DWIDTH )
    deltaX = -SPEED;
    /* Have we reached the top side? */
    if( ras_info1.RyOffset <= 0 )
      deltaY = SPEED;
    /* Have we reached the bottom (Raster height - Display height) side? */
    if( ras_info1.RyOffset >= RHEIGHT1 - DHEIGHT )
      deltaY = -SPEED;


    /* Recalculate the display instructions: (If you change any values */
    /* in the display structures the Amiga have to recalculate the     */
    /* entire display instructions. You must therefore call all three  */
    /* display functions: MakeVPort(), MrgCop() and LoadView().)       */
    MakeVPort( &view, &view_port );
    MrgCop( &view );
    LoadView( &view );
  }



  /* 9. Restore the old View: */
  LoadView( old_view );


  /* Free all allocated resources and leave. */
  clean_up( "THE END" );
}


/* Returns all allocated resources: */
void clean_up( message )
STRPTR message;
{
  int loop;

  /* Free automatically allocated display structures: */
  FreeVPortCopLists( &view_port );
  FreeCprList( view.LOFCprList );

  /* Deallocate the display memory, BitPlane for BitPlane: */
  /* Playfield 1: */
  for( loop = 0; loop < DEPTH1; loop++ )
    if( bit_map1.Planes[ loop ] )
      FreeRaster( bit_map1.Planes[ loop ], RWIDTH1, RHEIGHT1 );
  /* Playfield 2: */
  for( loop = 0; loop < DEPTH2; loop++ )
    if( bit_map2.Planes[ loop ] )
      FreeRaster( bit_map2.Planes[ loop ], RWIDTH2, RHEIGHT2 );

  /* Deallocate the ColorMap: */
  if( view_port.ColorMap ) FreeColorMap( view_port.ColorMap );

  /* Close the Graphics library: */
  if( GfxBase ) CloseLibrary( GfxBase );

  /* Close the Intuition library: */
  if( IntuitionBase ) CloseLibrary( IntuitionBase );

  /* Print the message and leave: */
  printf( "%s\n", message );
  exit();
}
```

Example7

This example demonstrates how to create a ViewPort with the special display mode "Hold and Modify".

```
/* Example 7                                                         */
/* This example demonstrates how to create a ViewPort with the special */
/* display mode "Hold and Modify".                                   */
/*                                                                   */
/* ------------------------------------------------------------      */
/* BitPlane                                                          */
/*  543210   Description                                             */
/* ------------------------------------------------------------      */
/*  00XXXX   One of the base colours will be used.                   */
/*  01XXXX   The pixel to left will be dublicated, and the blue      */
/*           value will be set by the first four bits (XXXX).        */
/*  10XXXX   The pixel to left will be dublicated, and the red       */
/*           value will be set by the first four bits (XXXX).        */
/*  01XXXX   The pixel to left will be dublicated, and the green     */
/*           value will be set by the first four bits (XXXX).        */
/* ------------------------------------------------------------      */


#include <intuition/intuition.h>
#include <graphics/gfxbase.h>


/* Whith help of this macro we can write numbers in binary, and it */
/* will be translated to normal decimal numbers. For example:      */
/* BIN(0,1,0,1,0,1) will be 21 (010101[b] -> 21[d])                */
#define BIN(a,b,c,d,e,f) ((a)<<5|(b)<<4|(c)<<3|(d)<<2|(e)<<1|(f))


#define WIDTH   320 /* 320 pixels wide (low resolution)             */
#define HEIGHT  200 /* 200 lines high (non interlaced NTSC display) */
#define DEPTH     6 /* 6 BitPlanes + HAM = 4096 colours.            */
#define COLOURS  16 /* 16 base colours.                             */


struct IntuitionBase *IntuitionBase;
struct GfxBase *GfxBase;


struct View my_view;
struct View *my_old_view;
struct ViewPort my_view_port;
struct RasInfo my_ras_info;
struct BitMap my_bit_map;
struct RastPort my_rast_port;


/* The base colours: */
UWORD my_color_table[] =
{
  0x000, /* Colour  0, Black */
  0x111, /* Colour  1,       */
  0x222, /* Colour  2,   |   */
  0x333, /* Colour  3,   |   */
  0x444, /* Colour  4,   |   */
  0x555, /* Colour  5,   |   */
  0x666, /* Colour  6,   |   */
  0x777, /* Colour  7,   |   */
  0x888, /* Colour  8,   |   */
  0x999, /* Colour  9,   |   */
  0xAAA, /* Colour 10,   |   */
  0xBBB, /* Colour 11,   |   */
  0xCCC, /* Colour 12,   |   */
  0xDDD, /* Colour 13,   V   */
  0xEEE, /* Colour 14,       */
  0xFFF, /* Colour 15, White */
};


void clean_up();
void main();


void main()
{
  UWORD *pointer;
  int loop;


  /* Open the Intuition library: */
  IntuitionBase = (struct IntuitionBase *)
    OpenLibrary( "intuition.library", 0 );
  if( !IntuitionBase )
    clean_up( "Could NOT open the Intuition library!" );

  /* Open the Graphics library: */
  GfxBase = (struct GfxBase *)
    OpenLibrary( "graphics.library", 0 );
  if( !GfxBase )
    clean_up( "Could NOT open the Graphics library!" );


  /* Save the current View, so we can restore it later: */
  my_old_view = GfxBase->ActiView;


  /* 1. Prepare the View structure, and give it a pointer to */
  /*    the first ViewPort:                                  */
  InitView( &my_view );
  my_view.ViewPort = &my_view_port;


  /* 2. Prepare the ViewPort structure, and set some important values: */
  InitVPort( &my_view_port );
  my_view_port.DWidth = WIDTH;         /* Set the width.                */
  my_view_port.DHeight = HEIGHT;       /* Set the height.               */
  my_view_port.RasInfo = &my_ras_info; /* Give it a pointer to RasInfo. */
  my_view_port.Modes = HAM;            /* Hold And Moduify.             */


  /* 3. Get a colour map, link it to the ViewPort, and prepare it: */
  my_view_port.ColorMap = (struct ColorMap *) GetColorMap( COLOURS );
  if( my_view_port.ColorMap == NULL )
    clean_up( "Could NOT get a ColorMap!" );

  /* Get a pointer to the colour map: */
  pointer = (UWORD *) my_view_port.ColorMap->ColorTable;

  /* Set the colours: */
  for( loop = 0; loop < COLOURS; loop++ )
    *pointer++ = my_color_table[ loop ];


  /* 4. Prepare the BitMap: */
  InitBitMap( &my_bit_map, DEPTH, WIDTH, HEIGHT );
```

```
/* Allocate memory for the Raster: */
for( loop = 0; loop < DEPTH; loop++ )
{
  my_bit_map.Planes[ loop ] = (PLANEPTR) AllocRaster( WIDTH, HEIGHT );
  if( my_bit_map.Planes[ loop ] == NULL )
    clean_up( "Could NOT allocate enough memory for the raster!" );

  /* Clear the display memory with help of the Blitter: */
  BltClear( my_bit_map.Planes[ loop ], RASSIZE( WIDTH, HEIGHT ), 0 );
}


/* 5. Prepare the RasInfo structure: */
my_ras_info.BitMap = &my_bit_map; /* Pointer to the BitMap structure.  */
my_ras_info.RxOffset = 0;         /* The top left corner of the Raster */
my_ras_info.RyOffset = 0;         /* should be at the top left corner  */
                                  /* of the display.                   */
my_ras_info.Next = NULL;          /* Single playfield - only one       */
                                  /* RasInfo structure is necessary.   */

/* 6. Create the display: */
MakeVPort( &my_view, &my_view_port );
MrgCop( &my_view );


/* 7. Prepare the RastPort, and give it a pointer to the BitMap. */
InitRastPort( &my_rast_port );
my_rast_port.BitMap = &my_bit_map;


/* 8. Show the new View: */
LoadView( &my_view );


/* Set the draw mode to JAM1. FgPen's colour will be used. */
SetDrMd( &my_rast_port, JAM1 );



/* Base colour 0: */
SetAPen( &my_rast_port, BIN(0,0,0,0,0,0) );
RectFill( &my_rast_port, 10, 10, 30, 130 );



/* Change R to 3 (0011 = 3): */
SetAPen( &my_rast_port, BIN(1,0,0,0,1,1) );
RectFill( &my_rast_port, 30, 10, 50, 50 );

/* Change R to 7 (0111 = 7): */
SetAPen( &my_rast_port, BIN(1,0,0,1,1,1) );
RectFill( &my_rast_port, 50, 10, 70, 50 );

/* Change R to 11 (1011 = 11): */
SetAPen( &my_rast_port, BIN(1,0,1,0,1,1) );
RectFill( &my_rast_port, 70, 10, 90, 50 );

/* Change R to 13 (1101 = 13): */
SetAPen( &my_rast_port, BIN(1,0,1,1,0,1) );
RectFill( &my_rast_port, 90, 10, 110, 50 );
```

```
/* Change R to 15 (1111 = 15): */
SetAPen( &my_rast_port, BIN(1,0,1,1,1,1) );
RectFill( &my_rast_port, 110, 10, 130, 50 );



/* Change B to 3 (0011 = 3): */
SetAPen( &my_rast_port, BIN(0,1,0,0,1,1) );
RectFill( &my_rast_port, 30, 50, 50, 90 );

/* Change B to 7 (0111 = 7): */
SetAPen( &my_rast_port, BIN(0,1,0,1,1,1) );
RectFill( &my_rast_port, 50, 50, 70, 90 );

/* Change B to 11 (1011 = 11): */
SetAPen( &my_rast_port, BIN(0,1,1,0,1,1) );
RectFill( &my_rast_port, 70, 50, 90, 90 );

/* Change B to 13 (1101 = 13): */
SetAPen( &my_rast_port, BIN(0,1,1,1,0,1) );
RectFill( &my_rast_port, 90, 50, 110, 90 );

/* Change B to 15 (1111 = 15): */
SetAPen( &my_rast_port, BIN(0,1,1,1,1,1) );
RectFill( &my_rast_port, 110, 50, 130, 90 );



/* Change G to 3 (0011 = 3): */
SetAPen( &my_rast_port, BIN(1,1,0,0,1,1) );
RectFill( &my_rast_port, 30, 90, 50, 130 );

/* Change G to 7 (0111 = 7): */
SetAPen( &my_rast_port, BIN(1,1,0,1,1,1) );
RectFill( &my_rast_port, 50, 90, 70, 130 );

/* Change G to 11 (1011 = 11): */
SetAPen( &my_rast_port, BIN(1,1,1,0,1,1) );
RectFill( &my_rast_port, 70, 90, 90, 130 );

/* Change G to 13 (1101 = 13): */
SetAPen( &my_rast_port, BIN(1,1,1,1,0,1) );
RectFill( &my_rast_port, 90, 90, 110, 130 );

/* Change G to 15 (1111 = 15): */
SetAPen( &my_rast_port, BIN(1,1,1,1,1,1) );
RectFill( &my_rast_port, 110, 90, 130, 130 );



/* Change the basecolour: (Black, dark grey, ... light grey, white) */
/* As you will notice, not only the base colour will change! Since  */
/* all rectangles' colours are modified versions of the base colour */
/* they will also change as the base colour change.                 */
for( loop = 0; loop < COLOURS; loop++ )
{
  Delay( 50 );
  SetAPen( &my_rast_port, loop );
  RectFill( &my_rast_port, 10, 10, 30, 130 );
}
Delay( 50 );
```

Example8

This example shows how to use the functions: SetAPen(), SetBPen(), SetOPen(), SetDrMd(), SetDrPt(), WritePixel(), ReadPixel(), Move(), Draw(), Text() and finally PolyDraw().

```
/* Example 8                                                         */
/* This example shows how to use the functions: SetAPen(), SetBPen(),  */
/* SetOPen(), SetDrMd(), SetDrPt(), WritePixel(), ReadPixel(), Move(), */
/* Draw(), Text() and finally PolyDraw().                              */


#include <intuition/intuition.h>
#include <graphics/gfxbase.h>
#include <graphics/gfxmacros.h>


/* NOTE! We must include the file "gfxmacros.h" inorder to be able to */
/* use the function (macro) SetDrPt().                                */


#define WIDTH  320 /* 320 pixels wide (low resolution)                */
#define HEIGHT 200 /* 200 lines high (non interlaced NTSC display)    */
#define DEPTH    2 /* 2 BitPlanes should be used, gives four colours. */
#define COLOURS  4 /* 2^2 = 4                                         */


struct IntuitionBase *IntuitionBase;
struct GfxBase *GfxBase;


struct View my_view;
struct View *my_old_view;
struct ViewPort my_view_port;
struct RasInfo my_ras_info;
struct BitMap my_bit_map;
struct RastPort my_rast_port;


UWORD my_color_table[] =
{
  0x000, /* Colour 0, Black */
  0xF00, /* Colour 1, Red   */
  0x0F0, /* Colour 2, Green */
  0x00F  /* Colour 3, Blue  */
};


/* The coordinates for the PolyDraw() function: (Creates a small box) */
WORD coordinates[] =
{
  100, 10,
  140, 10,
  140, 50,
  100, 50,
  100, 10
};


void clean_up();
void main();


void main()
{
  UWORD *pointer;
  int loop;
```

```
  /* Open the Intuition library: */
  IntuitionBase = (struct IntuitionBase *)
    OpenLibrary( "intuition.library", 0 );
  if( !IntuitionBase )
    clean_up( "Could NOT open the Intuition library!" );

  /* Open the Graphics library: */
  GfxBase = (struct GfxBase *)
    OpenLibrary( "graphics.library", 0 );
  if( !GfxBase )
    clean_up( "Could NOT open the Graphics library!" );


  /* Save the current View, so we can restore it later: */
  my_old_view = GfxBase->ActiView;


  /* 1. Prepare the View structure, and give it a pointer to */
  /*    the first ViewPort:                                  */
  InitView( &my_view );
  my_view.ViewPort = &my_view_port;


  /* 2. Prepare the ViewPort structure, and set some important values: */
  InitVPort( &my_view_port );
  my_view_port.DWidth = WIDTH;         /* Set the width.                */
  my_view_port.DHeight = HEIGHT;       /* Set the height.               */
  my_view_port.RasInfo = &my_ras_info; /* Give it a pointer to RasInfo. */
  my_view_port.Modes = NULL;           /* Low resolution.               */


  /* 3. Get a colour map, link it to the ViewPort, and prepare it: */
  my_view_port.ColorMap = (struct ColorMap *) GetColorMap( COLOURS );
  if( my_view_port.ColorMap == NULL )
    clean_up( "Could NOT get a ColorMap!" );

  /* Get a pointer to the colour map: */
  pointer = (UWORD *) my_view_port.ColorMap->ColorTable;

  /* Set the colours: */
  for( loop = 0; loop < COLOURS; loop++ )
    *pointer++ = my_color_table[ loop ];


  /* 4. Prepare the BitMap: */
  InitBitMap( &my_bit_map, DEPTH, WIDTH, HEIGHT );

  /* Allocate memory for the Raster: */
  for( loop = 0; loop < DEPTH; loop++ )
  {
    my_bit_map.Planes[ loop ] = (PLANEPTR) AllocRaster( WIDTH, HEIGHT );
    if( my_bit_map.Planes[ loop ] == NULL )
      clean_up( "Could NOT allocate enough memory for the raster!" );

    /* Clear the display memory with help of the Blitter: */
    BltClear( my_bit_map.Planes[ loop ], RASSIZE( WIDTH, HEIGHT ), 0 );
  }


  /* 5. Prepare the RasInfo structure: */
  my_ras_info.BitMap = &my_bit_map; /* Pointer to the BitMap structure.  */
  my_ras_info.RxOffset = 0;         /* The top left corner of the Raster */
  my_ras_info.RyOffset = 0;         /* should be at the top left corner  */
                                    /* of the display.                   */
  my_ras_info.Next = NULL;          /* Single playfield - only one       */
                                    /* RasInfo structure is necessary.   */

  /* 6. Create the display: */
  MakeVPort( &my_view, &my_view_port );
  MrgCop( &my_view );


  /* 7. Prepare the RastPort, and give it a pointer to the BitMap. */
  InitRastPort( &my_rast_port );
  my_rast_port.BitMap = &my_bit_map;


  /* 8. Show the new View: */
  LoadView( &my_view );


  SetDrMd( &my_rast_port, JAM1 ); /* Use FgPen only. */
  SetAPen( &my_rast_port, 2 );    /* FgPen: Green    */
  SetBPen( &my_rast_port, 1 );    /* BgPen: Red      */


  /* Write a pixel: */
  WritePixel( &my_rast_port, 10, 10 );

  /* Check what colour the pixel was drawn with: */
  printf( "Colour: %d\n", ReadPixel( &my_rast_port, 10, 10 ) );


  /* Move the cursor to (20, 10) and draw a simple line to (20, 100): */
  Move( &my_rast_port, 20, 10 );
  Draw( &my_rast_port, 20, 100 );


  /* Move the cursor to (25, 10) and draw a patterned line to (25, 100): */
  /* Pattern: 1111 0110 1111 0110 1111 = F6F6 (hexadecimal)              */
  SetDrPt( &my_rast_port, 0xF6F6 );
  Move( &my_rast_port, 25, 10 );
  Draw( &my_rast_port, 25, 100 );


  /* Write "Hello!" with FgPen (green), do not change the background: */
  Move( &my_rast_port, 30, 10 );
  Text( &my_rast_port, "Hello!", 6 );

  /* Write "Hello!" with FgPen and change background to BgPen: */
  /* (Green text on red background.)                           */
  SetDrMd( &my_rast_port, JAM2 );
  Move( &my_rast_port, 30, 20 );
  Text( &my_rast_port, "Hello!", 6 );

  /* Inversed JAM1. Black text on green background: */
  SetDrMd( &my_rast_port, JAM1|INVERSVID );
  Move( &my_rast_port, 30, 30 );
  Text( &my_rast_port, "Hello!", 6 );

  /* Inversed JAM2. Red text on black background: */
  SetDrMd( &my_rast_port, JAM2|INVERSVID );
  Move( &my_rast_port, 30, 40 );
```

```
  Text( &my_rast_port, "Hello!", 6 );

  /* Print the text in red with a green shadow: */
  /* JAM1, green text background unchanged (black): */
  SetDrMd( &my_rast_port, JAM1 );
  Move( &my_rast_port, 30, 50 );
  Text( &my_rast_port, "Hello!", 6 );
  /* Change FgPen to red: */
  SetAPen( &my_rast_port, 1 );
  Move( &my_rast_port, 31, 51 );
  Text( &my_rast_port, "Hello!", 6 );


  /* Draw a small red box:  */
  /* Move to the start position. (Otherwise there would be a line from */
  /* were the cursor is for the moment up to the start position.)      */
  Move( &my_rast_port, 100, 10 );
  PolyDraw( &my_rast_port, 5, coordinates ); /* (5 : Five coordinates) */


  /* Wait 20 seconds: */
  Delay( 50 * 20 );


  /* 9. Restore the old View: */
  LoadView( my_old_view );


  /* Free all allocated resources and leave. */
  clean_up( "THE END" );
}


/* Returns all allocated resources: */
void clean_up( message )
STRPTR message;
{
  int loop;

  /* Free automatically allocated display structures: */
  FreeVPortCopLists( &my_view_port );
  FreeCprList( my_view.LOFCprList );

  /* Deallocate the display memory, BitPlane for BitPlane: */
  for( loop = 0; loop < DEPTH; loop++ )
    if( my_bit_map.Planes[ loop ] )
      FreeRaster( my_bit_map.Planes[ loop ], WIDTH, HEIGHT );

  /* Deallocate the ColorMap: */
  if( my_view_port.ColorMap ) FreeColorMap( my_view_port.ColorMap );

  /* Close the Graphics library: */
  if( GfxBase ) CloseLibrary( GfxBase );

  /* Close the Intuition library: */
  if( IntuitionBase ) CloseLibrary( IntuitionBase );

  /* Print the message and leave: */
  printf( "%s\n", message );
  exit();
}
```

Example9

This example shows how to flood fill a figure, and how to draw filled rectangles (both solid as well as filled with single and multi coloured patterns).

```
/* Example 9                                                        */
/* This example shows how to flood fill a figure, and how to draw filled */
/* rectangles (both solid as well as filled with single and multi        */
/* coloured patterns).                                                */


#include <intuition/intuition.h>
#include <graphics/gfxbase.h>
#include <graphics/gfxmacros.h>


/* NOTE! We must include the file "gfxmacros.h" inorder to be able to */
/* use the functions (macros) SetOPen() and SetAfPt().               */


#define WIDTH  320 /* 320 pixels wide (low resolution)                */
#define HEIGHT 200 /* 200 lines high (non interlaced NTSC display)    */
#define DEPTH    2 /* 2 BitPlanes should be used, gives four colours. */
#define COLOURS  4 /* 2^2 = 4                                         */


#define OUTLINE_MODE 0 /* Fill until we find same colour as Outline Pen. */
#define COLOUR_MODE  1 /* Fill until we find another colour.             */


struct IntuitionBase *IntuitionBase;
struct GfxBase *GfxBase;


struct View my_view;
struct View *my_old_view;
struct ViewPort my_view_port;
struct RasInfo my_ras_info;
struct BitMap my_bit_map;
struct RastPort my_rast_port;


UWORD my_color_table[] =
{
  0x000, /* Colour 0, Black */
  0xF00, /* Colour 1, Red   */
  0x0F0, /* Colour 2, Green */
  0x00F  /* Colour 3, Blue  */
};


/* The coordinates (25) for the PolyDraw() function: */
WORD coordinates[] =
{
    0,   0,
  120,   0,
  120,  40,
  180,  40,
  180,   0,
  300,   0,
  300,  20,
  200,  20,
  200,  60,
  160,  60,
  160, 100,
  280, 100,
  280,  40,
  300,  40,
  300, 120,
    0, 120,
    0,  40,
   20,  40,
   20, 100,
  140, 100,
  140,  60,
  100,  60,
  100,  20,
    0,  20,
    0,   0
};


/* A heart (1 BitPlane):                                                */
/* An area pattern is always 16 bits wide, and the hight is some power */
/* of two (1, 2, 4, 8, 16, 32, and so on ).                             */
UWORD pattern[] =
{
  0x38E0, /* 0011 1000 1110 0000 */
  0x7DF0, /* 0111 1101 1111 0000 */
  0xFFF8, /* 1111 1111 1111 1000 */
  0xFFF8, /* 1111 1111 1111 1000 */
  0xFFF8, /* 1111 1111 1111 1000 */
  0x7FF0, /* 0111 1111 1111 0000 */
  0x3FE0, /* 0011 1111 1110 0000 */
  0x1FC0, /* 0001 1111 1100 0000 */
  0x0F80, /* 0000 1111 1000 0000 */
  0x0700, /* 0000 0111 0000 0000 */
  0x0200, /* 0000 0010 0000 0000 */

  0x0000, /* 0000 0000 0000 0000 */
  0x0000, /* 0000 0000 0000 0000 */
  0x0000, /* 0000 0000 0000 0000 */
  0x0000, /* 0000 0000 0000 0000 */
  0x0000, /* 0000 0000 0000 0000 */
};


/* A four-coloured pattern: (Black, red, green and blue lines) */
UWORD coloured_pattern[][] =
{
  {
    0x00FF, /* BitPlane 0 */
    0xFF00,
    0x00FF,
    0xFF00
  },
  {
    0x00FF, /* BitPlane 1 */
    0x00FF,
    0xFF00,
    0xFF00
  }
};


void clean_up();
void main();
void main()
{
  UWORD *pointer;
  int loop;


  /* Open the Intuition library: */
  IntuitionBase = (struct IntuitionBase *)
    OpenLibrary( "intuition.library", 0 );
  if( !IntuitionBase )
    clean_up( "Could NOT open the Intuition library!" );

  /* Open the Graphics library: */
  GfxBase = (struct GfxBase *)
    OpenLibrary( "graphics.library", 0 );
  if( !GfxBase )
    clean_up( "Could NOT open the Graphics library!" );


  /* Save the current View, so we can restore it later: */
  my_old_view = GfxBase->ActiView;


  /* 1. Prepare the View structure, and give it a pointer to */
  /*    the first ViewPort:                                  */
  InitView( &my_view );
  my_view.ViewPort = &my_view_port;


  /* 2. Prepare the ViewPort structure, and set some important values: */
  InitVPort( &my_view_port );
  my_view_port.DWidth = WIDTH;         /* Set the width.                */
  my_view_port.DHeight = HEIGHT;       /* Set the height.               */
  my_view_port.RasInfo = &my_ras_info; /* Give it a pointer to RasInfo. */
  my_view_port.Modes = NULL;           /* Low resolution.               */


  /* 3. Get a colour map, link it to the ViewPort, and prepare it: */
  my_view_port.ColorMap = (struct ColorMap *) GetColorMap( COLOURS );
  if( my_view_port.ColorMap == NULL )
    clean_up( "Could NOT get a ColorMap!" );

  /* Get a pointer to the colour map: */
  pointer = (UWORD *) my_view_port.ColorMap->ColorTable;

  /* Set the colours: */
  for( loop = 0; loop < COLOURS; loop++ )
    *pointer++ = my_color_table[ loop ];


  /* 4. Prepare the BitMap: */
  InitBitMap( &my_bit_map, DEPTH, WIDTH, HEIGHT );

  /* Allocate memory for the Raster: */
  for( loop = 0; loop < DEPTH; loop++ )
  {
    my_bit_map.Planes[ loop ] = (PLANEPTR) AllocRaster( WIDTH, HEIGHT );
    if( my_bit_map.Planes[ loop ] == NULL )
      clean_up( "Could NOT allocate enough memory for the raster!" );

    /* Clear the display memory with help of the Blitter: */
    BltClear( my_bit_map.Planes[ loop ], RASSIZE( WIDTH, HEIGHT ), 0 );
```

```
  }


  /* 5. Prepare the RasInfo structure: */
  my_ras_info.BitMap = &my_bit_map; /* Pointer to the BitMap structure.  */
  my_ras_info.RxOffset = 0;         /* The top left corner of the Raster */
  my_ras_info.RyOffset = 0;         /* should be at the top left corner  */
                                    /* of the display.                   */
  my_ras_info.Next = NULL;          /* Single playfield - only one       */
                                    /* RasInfo structure is necessary.   */

  /* 6. Create the display: */
  MakeVPort( &my_view, &my_view_port );
  MrgCop( &my_view );


  /* 7. Prepare the RastPort, and give it a pointer to the BitMap. */
  InitRastPort( &my_rast_port );
  my_rast_port.BitMap = &my_bit_map;


  /* 8. Show the new View: */
  LoadView( &my_view );



  SetDrMd( &my_rast_port, JAM2 ); /* Use Fg and Bg Pen. */
  SetAPen( &my_rast_port, 3 );    /* FgPen:  Blue       */
  SetBPen( &my_rast_port, 2 );    /* BgPen:  Green      */
  SetOPen( &my_rast_port, 3 );    /* BgPen:  Blue       */


  /* Draw a funny figure in blue colour: */
  Move( &my_rast_port, 0, 0 );
  PolyDraw( &my_rast_port, 25, coordinates );


  /* Wait 5 seconds: */
  Delay( 50 * 5 );


  /* Change FgPen colour to red, and fill the figure: */
  SetAPen( &my_rast_port, 1 );    /* FgPen: Red       */
  Flood( &my_rast_port, OUTLINE_MODE, 10, 10 );


  /* Wait 5 seconds: */
  Delay( 50 * 5 );


  /* Draw a filled rectangle at the bottom of the display: */
  RectFill( &my_rast_port, 0, 150, 150, 190 );


  /* Wait 5 seconds: */
  Delay( 50 * 5 );


  /* Set the are pattern. We will now draw a rectangle filled with a */
  /* lot of  hearts. (The pattern is 16 lines tall which is 2 to the  */
  /* power of 4.)                                                     */
  SetAfPt( &my_rast_port, (USHORT *) pattern, 4);
  /* Draw a rectangle filled with hearts at the bottom of the display: */
  RectFill( &my_rast_port, 150, 150, 300, 190 );

  /* Wait 5 seconds: */
  Delay( 50 * 5 );


  /* Prepare to fill with a coloured pattern: */
  /* Drawmode JAM2, FgPen colour 255, BgPen 0 */
  SetDrMd( &my_rast_port, JAM2 );
  SetAPen( &my_rast_port, 255 );
  SetBPen( &my_rast_port, 0 );
  SetAfPt( &my_rast_port, (USHORT *) coloured_pattern, -2);
  /* 4 lines = 2^2 -> 2 : Multicolour: -2 */

  /* Draw a rectangle filled with four colours: */
  RectFill( &my_rast_port, 0, 150, 300, 190 );

  /* Wait 5 seconds: */
  Delay( 50 * 5 );



  /* 9. Restore the old View: */
  LoadView( my_old_view );


  /* Free all allocated resources and leave. */
  clean_up( "THE END" );
}


/* Returns all allocated resources: */
void clean_up( message )
STRPTR message;
{
  int loop;

  /* Free automatically allocated display structures: */
  FreeVPortCopLists( &my_view_port );
  FreeCprList( my_view.LOFCprList );

  /* Deallocate the display memory, BitPlane for BitPlane: */
  for( loop = 0; loop < DEPTH; loop++ )
    if( my_bit_map.Planes[ loop ] )
      FreeRaster( my_bit_map.Planes[ loop ], WIDTH, HEIGHT );

  /* Deallocate the ColorMap: */
  if( my_view_port.ColorMap ) FreeColorMap( my_view_port.ColorMap );

  /* Close the Graphics library: */
  if( GfxBase ) CloseLibrary( GfxBase );

  /* Close the Intuition library: */
  if( IntuitionBase ) CloseLibrary( IntuitionBase );

  /* Print the message and leave: */
  printf( "%s\n", message );
  exit();
}
```

Example10

This example demonstrate how to use the Area Fill functions.
[AreaMove(), AreaDraw() and AreaEnd().]

```
/* Example 10                                                */
/* This example demonstrates how to use the Area Fill functions. */
/* [ AreaMove(), AreaDraw() and AreaEnd(). ]                     */


#include <intuition/intuition.h>
#include <graphics/gfxbase.h>
#include <graphics/gfxmacros.h>


#define WIDTH  320 /* 320 pixels wide (low resolution)                */
#define HEIGHT 200 /* 200 lines high (non interlaced NTSC display)    */
#define DEPTH    2 /* 2 BitPlanes should be used, gives four colours. */
#define COLOURS  4 /* 2^2 = 4                                         */


#define MAX_VERTICES 100 /* 100 vertices, 5 bytes each = 500 bytes. */
#define BUFFERT_SIZE 250 /* 500 bytes = 250 words.                  */


struct IntuitionBase *IntuitionBase;
struct GfxBase *GfxBase;


struct View my_view;
struct View *my_old_view;
struct ViewPort my_view_port;
struct RasInfo my_ras_info;
struct BitMap my_bit_map;

struct RastPort my_rast_port;
struct TmpRas  my_temp_ras;
struct AreaInfo my_area_info;


UWORD my_color_table[] =
{
  0x000, /* Colour 0, Black */
  0xF00, /* Colour 1, Red   */
  0x0F0, /* Colour 2, Green */
  0x00F  /* Colour 3, Blue  */
};


/* The buffert must start on a word boundary: */
UWORD buffert[ BUFFERT_SIZE ];
PLANEPTR extra_space;


void clean_up();
void main();


void main()
{
  UWORD *pointer;
  int loop;


  /* Open the Intuition library: */
  IntuitionBase = (struct IntuitionBase *)
    OpenLibrary( "intuition.library", 0 );
```

```
  if( !IntuitionBase )
    clean_up( "Could NOT open the Intuition library!" );

  /* Open the Graphics library: */
  GfxBase = (struct GfxBase *)
    OpenLibrary( "graphics.library", 0 );
  if( !GfxBase )
    clean_up( "Could NOT open the Graphics library!" );


  /* Save the current View, so we can restore it later: */
  my_old_view = GfxBase->ActiView;


  /* Prepare the View structure, and give it a pointer to */
  /* the first ViewPort:                                  */
  InitView( &my_view );
  my_view.ViewPort = &my_view_port;


  /* Prepare the ViewPort structure, and set some important values:     */
  InitVPort( &my_view_port );
  my_view_port.DWidth = WIDTH;         /* Set the width.                */
  my_view_port.DHeight = HEIGHT;       /* Set the height.               */
  my_view_port.RasInfo = &my_ras_info; /* Give it a pointer to RasInfo. */
  my_view_port.Modes = NULL;           /* Low resolution.               */


  /* Get a colour map, link it to the ViewPort, and prepare it: */
  my_view_port.ColorMap = (struct ColorMap *) GetColorMap( COLOURS );
  if( my_view_port.ColorMap == NULL )
    clean_up( "Could NOT get a ColorMap!" );

  /* Get a pointer to the colour map: */
  pointer = (UWORD *) my_view_port.ColorMap->ColorTable;

  /* Set the colours: */
  for( loop = 0; loop < COLOURS; loop++ )
    *pointer++ = my_color_table[ loop ];


  /* Prepare the BitMap: */
  InitBitMap( &my_bit_map, DEPTH, WIDTH, HEIGHT );

  /* Allocate memory for the Raster: */
  for( loop = 0; loop < DEPTH; loop++ )
  {
    my_bit_map.Planes[ loop ] = (PLANEPTR) AllocRaster( WIDTH, HEIGHT );
    if( my_bit_map.Planes[ loop ] == NULL )
      clean_up( "Could NOT allocate enough memory for the raster!" );

    /* Clear the display memory with help of the Blitter: */
    BltClear( my_bit_map.Planes[ loop ], RASSIZE( WIDTH, HEIGHT ), 0 );
  }


  /* Prepare the RasInfo structure: */
  my_ras_info.BitMap = &my_bit_map; /* Pointer to the BitMap structure.  */
  my_ras_info.RxOffset = 0;         /* The top left corner of the Raster */
  my_ras_info.RyOffset = 0;         /* should be at the top left corner  */
                                    /* of the display.                   */
  my_ras_info.Next = NULL;          /* Single playfield - only one       */
```

```
                                  /* RasInfo structure is necessary.   */

  /* Create the display: */
  MakeVPort( &my_view, &my_view_port );
  MrgCop( &my_view );


  /* Prepare the RastPort, and give it a pointer to the BitMap. */
  InitRastPort( &my_rast_port );
  my_rast_port.BitMap = &my_bit_map;



  /* 1. Get some space for the vertices and initialize the AreaInfo ptr: */
  InitArea( &my_area_info, buffert, MAX_VERTICES );
  my_rast_port.AreaInfo = &my_area_info;


  /* 2. Allocate some space that is needed to build up the objects: */
  extra_space = (PLANEPTR) AllocRaster( WIDTH, HEIGHT );
  if( extra_space == NULL )
    clean_up( "Could NOT allocate enough memory for the temp raster!" );


  /* 3. Initialize the TmpRas structure: */
  my_rast_port.TmpRas = (struct TmpRas *)
    InitTmpRas( &my_temp_ras, extra_space, RASSIZE( WIDTH, HEIGHT ) );



  /* Show the new View: */
  LoadView( &my_view );



  SetAPen( &my_rast_port, 1 ); /* Red   */
  SetBPen( &my_rast_port, 0 ); /* Black */
  SetOPen( &my_rast_port, 2 ); /* Green */
  SetDrMd( &my_rast_port, JAM1 );



  /* New position: */
  AreaMove( &my_rast_port,  10,  10 );

  /* Add the vertices: */
  AreaDraw( &my_rast_port, 310,  10 );
  AreaDraw( &my_rast_port, 310, 100 );
  AreaDraw( &my_rast_port, 290, 100 );
  AreaDraw( &my_rast_port, 290,  30 );
  AreaDraw( &my_rast_port,  30,  30 );
  AreaDraw( &my_rast_port,  30, 100 );
  AreaDraw( &my_rast_port,  10, 100 );

  /* End this object. The last line will be set automatically in order */
  /* to close the object, and the figure will be filled. The Outline   */
  /* pen (green) will be used to draw a line around the whole object.  */
  AreaEnd( &my_rast_port );


  /* Turn off the outline function: */
  BNDRYOFF( &my_rast_port );
```

```
  /* New position: (This figure will not be outlined.) */
  AreaMove( &my_rast_port,  10,  190 );

  /* Add the vertices: */
  AreaDraw( &my_rast_port,  10, 150);
  AreaDraw( &my_rast_port, 310, 190);
  AreaDraw( &my_rast_port, 310, 150);

  /* End this object: */
  AreaEnd( &my_rast_port );



  /* Wait 10 seconds: */
  Delay( 50 * 10 );


  /* Restore the old View: */
  LoadView( my_old_view );


  /* Free all allocated resources and leave. */
  clean_up( "THE END" );
}


/* Returns all allocated resources: */
void clean_up( message )
STRPTR message;
{
  int loop;

  /* Deallocate memory used for the objects: */
  if( extra_space )
    FreeRaster( extra_space, WIDTH, HEIGHT );

  /* Free automatically allocated display structures: */
  FreeVPortCopLists( &my_view_port );
  FreeCprList( my_view.LOFCprList );

  /* Deallocate the display memory, BitPlane for BitPlane: */
  for( loop = 0; loop < DEPTH; loop++ )
    if( my_bit_map.Planes[ loop ] )
      FreeRaster( my_bit_map.Planes[ loop ], WIDTH, HEIGHT );

  /* Deallocate the ColorMap: */
  if( my_view_port.ColorMap ) FreeColorMap( my_view_port.ColorMap );

  /* Close the Graphics library: */
  if( GfxBase ) CloseLibrary( GfxBase );

  /* Close the Intuition library: */
  if( IntuitionBase ) CloseLibrary( IntuitionBase );

  /* Print the message and leave: */
  printf( "%s\n", message );
  exit();
}
```

Example11

This example demonstrate how to copy rectangular memory areas with help of the blitter.

```
/* Example 11                                                 */
/* This example demonstrate how to copy rectangular memory areas */
/* with help of the blitter.                                  */



#include <intuition/intuition.h>
#include <graphics/gfxbase.h>
#include <graphics/gfxmacros.h>


/* NOTE! We must include the file "gfxmacros.h" inorder to be able to */
/* use the function (macro) SetDrPt().                                */


#define WIDTH  320 /* 320 pixels wide (low resolution)             */
#define HEIGHT 200 /* 200 lines high (non interlaced NTSC display) */
#define DEPTH    2 /* 2 BitPlanes should be used, gives 4 colours. */
#define COLOURS  4 /* 2^2 = 4                                      */


struct IntuitionBase *IntuitionBase;
struct GfxBase *GfxBase;


struct View my_view;
struct View *my_old_view;
struct ViewPort my_view_port;
struct RasInfo my_ras_info;
struct BitMap my_bit_map;
struct RastPort my_rast_port;


UWORD my_color_table[] =
{
  0x000, /* Colour  0, Black      */
  0x555, /* Colour  1, Dark grey  */
  0x777, /* Colour  2, Grey       */
  0x999, /* Colour  3, Light grey */
};


void clean_up();
void main();


void main()
{
  UWORD *pointer;
  int loop;
  int x, y;


  /* Open the Intuition library: */
  IntuitionBase = (struct IntuitionBase *)
    OpenLibrary( "intuition.library", 0 );
  if( !IntuitionBase )
    clean_up( "Could NOT open the Intuition library!" );

  /* Open the Graphics library: */
  GfxBase = (struct GfxBase *)
    OpenLibrary( "graphics.library", 0 );
```

```
  if( !GfxBase )
    clean_up( "Could NOT open the Graphics library!" );


  /* Save the current View, so we can restore it later: */
  my_old_view = GfxBase->ActiView;


  /* 1. Prepare the View structure, and give it a pointer to */
  /*    the first ViewPort:                                  */
  InitView( &my_view );
  my_view.ViewPort = &my_view_port;


  /* 2. Prepare the ViewPort structure, and set some important values: */
  InitVPort( &my_view_port );
  my_view_port.DWidth = WIDTH;         /* Set the width.                */
  my_view_port.DHeight = HEIGHT;       /* Set the height.               */
  my_view_port.RasInfo = &my_ras_info; /* Give it a pointer to RasInfo. */
  my_view_port.Modes = NULL;           /* Low resolution.               */


  /* 3. Get a colour map, link it to the ViewPort, and prepare it: */
  my_view_port.ColorMap = (struct ColorMap *) GetColorMap( COLOURS );
  if( my_view_port.ColorMap == NULL )
    clean_up( "Could NOT get a ColorMap!" );

  /* Get a pointer to the colour map: */
  pointer = (UWORD *) my_view_port.ColorMap->ColorTable;

  /* Set the colours: */
  for( loop = 0; loop < COLOURS; loop++ )
    *pointer++ = my_color_table[ loop ];


  /* 4. Prepare the BitMap: */
  InitBitMap( &my_bit_map, DEPTH, WIDTH, HEIGHT );

  /* Allocate memory for the Raster: */
  for( loop = 0; loop < DEPTH; loop++ )
  {
    my_bit_map.Planes[ loop ] = (PLANEPTR) AllocRaster( WIDTH, HEIGHT );
    if( my_bit_map.Planes[ loop ] == NULL )
      clean_up( "Could NOT allocate enough memory for the raster!" );

    /* Clear the display memory with help of the Blitter: */
    BltClear( my_bit_map.Planes[ loop ], RASSIZE( WIDTH, HEIGHT ), 0 );
  }


  /* 5. Prepare the RasInfo structure: */
  my_ras_info.BitMap = &my_bit_map; /* Pointer to the BitMap structure.  */
  my_ras_info.RxOffset = 0;         /* The top left corner of the Raster */
  my_ras_info.RyOffset = 0;         /* should be at the top left corner  */
                                    /* of the display.                   */
  my_ras_info.Next = NULL;          /* Single playfield - only one       */
                                    /* RasInfo structure is necessary.   */

  /* 6. Create the display: */
  MakeVPort( &my_view, &my_view_port );
  MrgCop( &my_view );
  /* 7. Prepare the RastPort, and give it a pointer to the BitMap. */
  InitRastPort( &my_rast_port );
  my_rast_port.BitMap = &my_bit_map;


  /* 8. Show the new View: */
  LoadView( &my_view );


  SetDrMd( &my_rast_port, JAM1 ); /* Use FgPen only.   */

  SetAPen( &my_rast_port, 1 );    /* Dark grey */
  Move( &my_rast_port, 10, 10 );
  Draw( &my_rast_port, 26, 10 );
  Draw( &my_rast_port, 26, 26 );

  SetAPen( &my_rast_port, 3 );    /* Light grey */
  Draw( &my_rast_port, 10, 26 );
  Draw( &my_rast_port, 10, 10 );

  SetAPen( &my_rast_port, 2 );    /* Grey */
  RectFill( &my_rast_port, 11, 11, 25, 25 );
  WritePixel( &my_rast_port, 10, 10 );
  WritePixel( &my_rast_port, 26, 26 );

  SetAPen( &my_rast_port, 1 );    /* Dark grey */
  WritePixel( &my_rast_port, 13, 13 );
  WritePixel( &my_rast_port, 23, 13 );
  WritePixel( &my_rast_port, 23, 23 );
  WritePixel( &my_rast_port, 13, 23 );


  /* We will now make 150 copies of the brick: */
  for( x = 0; x < 15; x++ )
    for( y = 0; y < 10; y++ )
      BltBitMap(
        &my_bit_map, /* Source                 */
        10, 10,      /* Position, source.      */
        &my_bit_map, /* Destination.           */
        50 + 17 * x, /* Position, destination. */
        10 + 17 * y, /*          - " -         */
        17, 17,      /* Width and height.      */
        0xC0,        /* Normal copy.           */
        0xFF,        /* All bitplanes.         */
        NULL );      /* No temporary storage.  */


  /* Wait 20 seconds: */
  Delay( 50 * 20 );


  /* 9. Restore the old View: */
  LoadView( my_old_view );


  /* Free all allocated resources and leave. */
  clean_up( "THE END" );
}


/* Returns all allocated resources: */
```

```
void clean_up( message )
STRPTR message;
{
  int loop;

  /* Free automatically allocated display structures: */
  FreeVPortCopLists( &my_view_port );
  FreeCprList( my_view.LOFCprList );

  /* Deallocate the display memory, BitPlane for BitPlane: */
  for( loop = 0; loop < DEPTH; loop++ )
    if( my_bit_map.Planes[ loop ] )
      FreeRaster( my_bit_map.Planes[ loop ], WIDTH, HEIGHT );

  /* Deallocate the ColorMap: */
  if( my_view_port.ColorMap ) FreeColorMap( my_view_port.ColorMap );

  /* Close the Graphics library: */
  if( GfxBase ) CloseLibrary( GfxBase );

  /* Close the Intuition library: */
  if( IntuitionBase ) CloseLibrary( IntuitionBase );

  /* Print the message and leave: */
  printf( "%s\n", message );
  exit();
}
```

A.13 VSPRITES

Example1

This example demonstrates how to get and use a VSprite.
The VSprite can be moved around by the user by pressing
the arrow keys.

```
/* Example1                                                         */
/* This example demonstrates how to get and use a VSprite. The VSprite */
/* can be moved around by the user by pressing the arrow keys.         */



/* Since we use Intuition, include this file: */
#include <intuition/intuition.h>

/* Include this file since you are using sprites: */
#include <graphics/gels.h>



/* Declare the functions we are going to use: */
void main();
void clean_up();



struct IntuitionBase *IntuitionBase = NULL;
/* We need to open the Graphics library since we are using sprites: */
struct GfxBase *GfxBase = NULL;



/* Declare a pointer to a Screen structure: */
struct Screen *my_screen;

/* Declare and initialize your NewScreen structure: */
struct NewScreen my_new_screen=
{
  0,            /* LeftEdge  Should always be 0. */
  0,            /* TopEdge   Top of the display.*/
  640,          /* Width     We are using a high-resolution screen. */
  200,          /* Height    Non-Interlaced NTSC (American) display. */
  2,            /* Depth     4 colours. */
  0,            /* DetailPen Text should be drawn with colour reg. 0 */
  1,            /* BlockPen  Blocks should be drawn with colour reg. 1 */
  HIRES|SPRITES,/* ViewModes High resolution, sprites will be used. */
  CUSTOMSCREEN, /* Type      Your own customized screen. */
  NULL,         /* Font      Default font. */
  "VSprites!",  /* Title     The screen's title. */
  NULL,         /* Gadget    Must for the moment be NULL. */
  NULL          /* BitMap    No special CustomBitMap. */
};



/* Declare a pointer to a Window structure: */
struct Window *my_window = NULL;

/* Declare and initialize your NewWindow structure: */
struct NewWindow my_new_window=
{
  0,             /* LeftEdge    x position of the window. */
  0,             /* TopEdge     y positio of the window. */
  640,           /* Width       640 pixels wide. */
  200,           /* Height      200 lines high. */
  0,             /* DetailPen   Text should be drawn with colour reg. 0 */
  1,             /* BlockPen    Blocks should be drawn with colour reg. 1 */
  CLOSEWINDOW|   /* IDCMPFlags  The window will give us a message if the */
  RAWKEY,        /*             user has selected the Close window gad, */
                 /*             or if the user has pressed a key. */
  SMART_REFRESH| /* Flags       Intuition should refresh the window. */
  WINDOWCLOSE|   /*             Close Gadget. */
  WINDOWDRAG|    /*             Drag gadget. */
  WINDOWDEPTH|   /*             Depth arrange Gadgets. */
  WINDOWSIZING|  /*             Sizing Gadget. */
  ACTIVATE,      /*             The window should be Active when opened. */
  NULL,          /* FirstGadget No Custom gadgets. */
  NULL,          /* CheckMark   Use Intuition's default CheckMark. */
  "Use the arrow keys to move the VSprite!", /* Title */
  NULL,          /* Screen      Will later be connected to a custom scr. */
  NULL,          /* BitMap      No Custom BitMap. */
  80,            /* MinWidth    We will not allow the window to become */
  30,            /* MinHeight   smaller than 80 x 30, and not bigger */
  640,           /* MaxWidth    than 640 x 200. */
  200,           /* MaxHeight */
  CUSTOMSCREEN   /* Type        Connected to the Workbench Screen. */
};



/* 1. Declare and initialize some sprite */
/*    data for each VSprite:             */
UWORD chip vsprite_data[]=
{
  0x0180, 0x0000,
  0x03C0, 0x0000,
  0x07E0, 0x0000,
  0x0FF0, 0x0000,
  0x1FF8, 0x0000,
  0x3FFC, 0x0000,
  0x7FFE, 0x0000,
  0x0000, 0xFFFF,
  0x0000, 0xFFFF,
  0x7FFE, 0x7FFE,
  0x3FFC, 0x3FFC,
  0x1FF8, 0x1FF8,
  0x0FF0, 0x0FF0,
  0x07E0, 0x07E0,
  0x03C0, 0x03C0,
  0x0180, 0x0180,
};



/* 2. Declare three VSprite structures. One will be used, */
/*    the other two are "dummies":                        */
struct VSprite head, tail, vsprite;


/* 3. Decide the VSprite's colours:            */
/*                           RGB     RGB     RGB */
WORD colour_table[] = { 0x000F, 0x00F0, 0x0F00 };


/* 4. Declare a GelsInfo structure: */
struct GelsInfo ginfo;


/* This boolean variable will tell us if the VSprite is in */
/* the list or not:                                        */
```

```
BOOL vsprite_on = FALSE;


/* This program will not open any console window if run from */
/* Workbench, but we must therefore not print anything.      */
/* Functions like printf() must therefore not be used.       */
void _main()
{
  /* The GelsInfo structure needs the following arrays: */
  WORD nextline[ 8 ];
  WORD *lastcolor[ 8 ];

  /* Sprite position: */
  WORD x = 40;
  WORD y = 40;

  /* Direction of the sprite: */
  WORD x_direction = 0;
  WORD y_direction = 0;

  /* Boolean variable used for the while loop: */
  BOOL close_me = FALSE;

  ULONG class; /* IDCMP */
  USHORT code; /* Code */

  /* Declare a pointer to an IntuiMessage structure: */
  struct IntuiMessage *my_message;



  /* Open the Intuition Library: */
  IntuitionBase = (struct IntuitionBase *)
    OpenLibrary( "intuition.library", 0 );

  if( IntuitionBase == NULL )
    clean_up(); /* Could NOT open the Intuition Library! */



  /* 5. Open the Graphics Library:                                */
  /* Since we are using sprites we need to open the Graphics Library: */
  /* Open the Graphics Library: */
  GfxBase = (struct GfxBase *)
    OpenLibrary( "graphics.library", 0);

  if( GfxBase == NULL )
    clean_up(); /* Could NOT open the Graphics Library! */



  /* We will now try to open the screen: */
  my_screen = (struct Screen *) OpenScreen( &my_new_screen );

  /* Have we opened the screen succesfully? */
  if(my_screen == NULL)
    clean_up();


  my_new_window.Screen = my_screen;
```

```
  /* We will now try to open the window: */
  my_window = (struct Window *) OpenWindow( &my_new_window );

  /* Have we opened the window succesfully? */
  if(my_window == NULL)
    clean_up(); /* Could NOT open the Window! */



  /* 6. Initialize the GelsInfo structure: */

  /* All sprites except the first two may be used to draw */
  /* the VSprites: ( 11111100 = 0xFC )                     */
  ginfo.sprRsrvd = 0xFC;
  /* If we do not exclude the first two sprites, the mouse */
  /* pointer's colours may be affected.                     */


  /* Give the GelsInfo structure some memory: */
  ginfo.nextLine = nextline;
  ginfo.lastColor = lastcolor;


  /* Give the Rastport a pointer to the GelsInfo structure: */
  my_window->RPort->GelsInfo = &ginfo;


  /* Give the GelsInfo structure to the system: */
  InitGels( &head, &tail, &ginfo );



  /* 7. Initialize the VSprite structure: */

  vsprite.Flags = VSPRITE; /* It is a VSprite.            */
  vsprite.X = x;           /* X position.                 */
  vsprite.Y = y;           /* Y position.                 */
  vsprite.Height = 16;     /* 16 lines tall.              */
  vsprite.Width = 2;       /* Two bytes (16 pixels) wide. */
  vsprite.Depth = 2;       /* Two bitplanes, 4 colours.   */


  /* Pointer to the sprite data: */
  vsprite.ImageData = vsprite_data;

  /* Pointer to the colour table: */
  vsprite.SprColors = colour_table;



  /* 8. Add the VSprites to the VSprite list: */
  AddVSprite( &vsprite, my_window->RPort );

  /* The VSprite is in the list. */
  vsprite_on = TRUE;



  /* Stay in the while loop until the user has selected the Close window */
  /* gadget: */
  while( close_me == FALSE )
  {
```

```
/* Stay in the while loop as long as we can collect messages */
/* sucessfully: */
while(my_message = (struct IntuiMessage *) GetMsg(my_window->UserPort))
{
  /* After we have collected the message we can read it, and save any */
  /* important values which we maybe want to check later: */
  class = my_message->Class;
  code  = my_message->Code;


  /* After we have read it we reply as fast as possible: */
  /* REMEMBER! Do never try to read a message after you have replied! */
  /* Some other process has maybe changed it. */
  ReplyMsg( my_message );


  /* Check which IDCMP flag was sent: */
  switch( class )
  {
    case CLOSEWINDOW:     /* Quit! */
           close_me=TRUE;
           break;

    case RAWKEY:          /* A key was pressed! */
           /* Check which key was pressed: */
           switch( code )
           {
             /* Up Arrow: */
             case 0x4C:      y_direction = -1; break; /* Pressed */
             case 0x4C+0x80: y_direction = 0;  break; /* Released */

             /* Down Arrow: */
             case 0x4D:      y_direction = 1; break; /* Pressed */
             case 0x4D+0x80: y_direction = 0; break; /* Released */

             /* Right Arrow: */
             case 0x4E:      x_direction = 2; break; /* Pressed */
             case 0x4E+0x80: x_direction = 0; break; /* Released */

             /* Left Arrow: */
             case 0x4F:      x_direction = -2; break; /* Pressed */
             case 0x4F+0x80: x_direction = 0;  break; /* Released */
           }
           break;
  }
}



/* 12. Play around with the VSprite: */

/* Change the x/y position: */
x += x_direction;
y += y_direction;

/* Check that the sprite does not move outside the screen: */
if(x > 640)
  x = 640;
if(x < 0)
  x = 0;
if(y > 200)
  y = 200;
```

```
    if(y < 0)
      y = 0;

    vsprite.X = x;
    vsprite.Y = y;



    /* 9. Sort the Gels list: */
    SortGList( my_window->RPort );

    /* 10. Draw the Gels list: */
    DrawGList( my_window->RPort, &(my_screen->ViewPort) );

    /* 11. Set the Copper and redraw the display: */
    MakeScreen( my_screen );
    RethinkDisplay();
  }



  /* Free all allocated memory: (Close the window, libraries etc) */
  clean_up();

  /* THE END */
}



/* This function frees all allocated memory. */
void clean_up()
{
  /* 13. Remove the VSprites: */
  if( vsprite_on )
    RemVSprite( &vsprite );

  if( my_window )
    CloseWindow( my_window );

  if(my_screen )
    CloseScreen( my_screen );

  if( GfxBase )
    CloseLibrary( GfxBase );

  if( IntuitionBase )
    CloseLibrary( IntuitionBase );

  exit();
}
```

Example2

This example demonstrates how to use several VSprites each with its own colour table.

```c
/* Example 2                                              */
/* This example demonstrates how to use several VSprites */
/* each with its own colour table.                        */



/* Since we use Intuition, include this file: */
#include <intuition/intuition.h>

/* Include this file since you are using sprites: */
#include <graphics/gels.h>



/* We will use 15 VSprites: */
#define MAXVSPRITES 15

/* They will move two pixels each time: */
#define SPEED 2



/* Declare the functions we are going to use: */
void main();
void clean_up();



struct IntuitionBase *IntuitionBase = NULL;
/* We need to open the Graphics library since we are using sprites: */
struct GfxBase *GfxBase = NULL;


/* Declare a pointer to a Screen structure: */
struct Screen *my_screen;

/* Declare and initialize your NewScreen structure: */
struct NewScreen my_new_screen=
{
  0,            /* LeftEdge  Should always be 0. */
  0,            /* TopEdge   Top of the display.*/
  320,          /* Width     We are using a low-resolution screen. */
  200,          /* Height    Non-Interlaced NTSC (American) display. */
  2,            /* Depth     4 colours. */
  0,            /* DetailPen Text should be drawn with colour reg. 0 */
  1,            /* BlockPen  Blocks should be drawn with colour reg. 1 */
  SPRITES,      /* ViewModes No special modes. (Low-res, Non-Interlaced) */
  CUSTOMSCREEN, /* Type      Your own customized screen. */
  NULL,         /* Font      Default font. */
  "MY SCREEN",  /* Title     The screen' title. */
  NULL,         /* Gadget    Must for the moment be NULL. */
  NULL          /* BitMap    No special CustomBitMap. */
};



/* Declare a pointer to a Window structure: */
struct Window *my_window = NULL;

/* Declare and initialize your NewWindow structure: */
struct NewWindow my_new_window=
{
  0,              /* LeftEdge    x position of the window. */
  0,              /* TopEdge     y positio of the window. */
  320,            /* Width       320 pixels wide. */
  200,            /* Height      200 lines high. */
  0,              /* DetailPen   Text should be drawn with colour reg. 0 */
  1,              /* BlockPen    Blocks should be drawn with colour reg. 1 */
  CLOSEWINDOW,    /* IDCMPFlags  The window will give us a message if the */
                  /*             user has selected the Close window gad. */
  SMART_REFRESH| /* Flags       Intuition should refresh the window. */
  WINDOWCLOSE|   /*             Close Gadget. */
  WINDOWDRAG|    /*             Drag gadget. */
  WINDOWDEPTH|   /*             Depth arrange Gadgets. */
  WINDOWSIZING|  /*             Sizing Gadget. */
  ACTIVATE,      /*             The window should be Active when opened. */
  NULL,          /* FirstGadget No Custom gadgets. */
  NULL,          /* CheckMark   Use Intuition's default CheckMark. */
  "VSprites are great!",        /* Title */
  NULL,          /* Screen      Will later be connected to a custom scr. */
  NULL,          /* BitMap      No Custom BitMap. */
  80,            /* MinWidth    We will not allow the window to become */
  30,            /* MinHeight   smaller than 80 x 30, and not bigger */
  320,           /* MaxWidth    than 320 x 200. */
  200,           /* MaxHeight   */
  CUSTOMSCREEN   /* Type        Connected to the Workbench Screen. */
};



/* 1. Declare and initialize some sprite data: */
UWORD chip vsprite_data[]=
{
  0x0180, 0x0000,
  0x03C0, 0x0000,
  0x07E0, 0x0000,
  0x0FF0, 0x0000,
  0x1FF8, 0x0000,
  0x3FFC, 0x0000,
  0x7FFE, 0x0000,
  0x0000, 0xFFFF,
  0x0000, 0xFFFF,
  0x7FFE, 0x7FFE,
  0x3FFC, 0x3FFC,
  0x1FF8, 0x1FF8,
  0x0FF0, 0x0FF0,
  0x07E0, 0x07E0,
  0x03C0, 0x03C0,
  0x0180, 0x0180,
};



/* 2. Declare three VSprite structures. One will be used, */
/*    the other two are "dummies":                         */
struct VSprite head, tail, vsprite[ MAXVSPRITES ];


/* 3. Declare the VSprites' colour tables:     */
WORD colour_table[ MAXVSPRITES ][ 3 ];


/* 4. Declare a GelsInfo structure: */
struct GelsInfo ginfo;
```

```
/* This boolean variable will tell us if the VSprites are */
/* in the list or not:                                     */
BOOL vsprite_on = FALSE;


/* This program will not open any console window if run from */
/* Workbench, but we must therefore not print anything.      */
/* Functions like printf() must therefore not be used.       */
void _main()
{
  /* The GelsInfo structure needs the following arrays: */
  WORD nextline[ 8 ];
  WORD *lastcolor[ 8 ];

  /* Direction of the sprite: */
  WORD x_direction[ MAXVSPRITES ];
  WORD y_direction[ MAXVSPRITES ];

  /* Boolean variable used for the while loop: */
  BOOL close_me = FALSE;

  /* Declare a pointer to an IntuiMessage structure: */
  struct IntuiMessage *my_message;

  /* Used as counter in the for loop: */
  UBYTE loop;



  /* Open the Intuition Library: */
  IntuitionBase = (struct IntuitionBase *)
    OpenLibrary( "intuition.library", 0 );

  if( IntuitionBase == NULL )
    clean_up(); /* Could NOT open the Intuition Library! */



  /* 5. Open the Graphics Library:                             */
  /* Since we are using sprites we need to open the Graphics Library: */
  /* Open the Graphics Library: */
  GfxBase = (struct GfxBase *)
    OpenLibrary( "graphics.library", 0);

  if( GfxBase == NULL )
    clean_up(); /* Could NOT open the Graphics Library! */



  /* We will now try to open the screen: */
  my_screen = (struct Screen *) OpenScreen( &my_new_screen );

  /* Have we opened the screen succesfully? */
  if(my_screen == NULL)
    clean_up();


  my_new_window.Screen = my_screen;
```

```
/* We will now try to open the window: */
my_window = (struct Window *) OpenWindow( &my_new_window );

/* Have we opened the window succesfully? */
if(my_window == NULL)
  clean_up(); /* Could NOT open the Window! */



/* 6. Initialize the GelsInfo structure: */

/* All sprites except the first two may be used to draw */
/* the VSprites: ( 11111100 = 0xFC )                    */
ginfo.sprRsrvd = 0xFC;
/* If we do not exclude the first two sprites, the mouse */
/* pointer's colours may be affected.                    */


/* Give the GelsInfo structure some memory: */
ginfo.nextLine = nextline;
ginfo.lastColor = lastcolor;


/* Give the Rastport a pointer to the GelsInfo structure: */
my_window->RPort->GelsInfo = &ginfo;


/* Give the GelsInfo structure to the system: */
InitGels( &head, &tail, &ginfo );



/* 7. Initialize the VSprite structures: */

/* Set a random seed: */
srand( 64 );

for( loop = 0; loop < MAXVSPRITES; loop++ )
{
  /* Set the VSprite's colours: */
  colour_table[ loop ][ 0 ] = loop;      /* Blue  */
  colour_table[ loop ][ 1 ] = loop << 4; /* Green */
  colour_table[ loop ][ 2 ] = loop << 8; /* Red   */

  /* Set the speed and direction of the VSprite: */
  x_direction[ loop ] = SPEED;
  y_direction[ loop ] = -SPEED;

  vsprite[ loop ].Flags = VSPRITE;    /* It is a VSprite.    */
  vsprite[ loop ].X = 10 + 20 * loop; /* X position.         */
  vsprite[ loop ].Y = 10 + 20 * loop; /* Y position.         */
  vsprite[ loop ].Height = 16;        /* 16 lines tall.      */
  vsprite[ loop ].Width = 2;          /* 2 words wide.       */
  vsprite[ loop ].Depth = 2;          /* 2 bitpl, 4 colours. */

  /* Pointer to the sprite data: */
  vsprite[ loop ].ImageData = vsprite_data;

  /* Pointer to the colour table: */
  vsprite[ loop ].SprColors = colour_table[ loop ];
```

```
    /* 8. Add the VSprites to the VSprite list: */
    AddVSprite( &vsprite[ loop ], my_window->RPort );
  }


  /* The VSprites are in the list. */
  vsprite_on = TRUE;


  /* Stay in the while loop until the user has */
  /* selected the Close window gadget:         */
  while( close_me == FALSE )
  {
    /* Stay in the while loop as long as we can collect messages: */
    while(my_message = (struct IntuiMessage *) GetMsg(my_window->UserPort))
    {
      if( my_message->Class == CLOSEWINDOW)
        close_me=TRUE;

      ReplyMsg( my_message );
    }


    /* Affect all VSprites: */
    for( loop = 0; loop < MAXVSPRITES; loop++ )
    {
      /* Change the position of the VSprite: */
      vsprite[ loop ].X += x_direction[ loop ];
      vsprite[ loop ].Y += y_direction[ loop ];


      /* Check that the sprite does not move outside the screen: */
      if(vsprite[ loop ].X > 300)
        x_direction[ loop ] = -SPEED;

      if(vsprite[ loop ].X < 0)
        x_direction[ loop ] = SPEED;

      if(vsprite[ loop ].Y > 180)
        y_direction[ loop ] = -SPEED;

      if(vsprite[ loop ].Y < 4)
        y_direction[ loop ] = SPEED;
    }

    /* 9. Sort the Gels list: */
    SortGList( my_window->RPort );

    /* 10. Draw the Gels list: */
    DrawGList( my_window->RPort, &(my_screen->ViewPort) );

    /* 11. Set the Copper and redraw the display: */
    MakeScreen( my_screen );
    RethinkDisplay();
  }



  /* Free all allocated memory: (Close the window, libraries etc) */
  clean_up();
```

```
  /* THE END */
}



/* This function frees all allocated memory. */
void clean_up()
{
  UBYTE loop;

  if( vsprite_on )
    for( loop = 0; loop < MAXVSPRITES; loop++ )
      RemVSprite( &vsprite[ loop ] );

  if( my_window )
    CloseWindow( my_window );

  if(my_screen )
    CloseScreen( my_screen );

  if( GfxBase )
    CloseLibrary( GfxBase );

  if( IntuitionBase )
    CloseLibrary( IntuitionBase );

  exit();
}
```

Example3

This program demonstrates how to animate several (!) VSprites.

```
/* Example 3                                                   */
/* This program demonstrates how to animate several (!) VSprites. */



/* Since we use Intuition, include this file: */
#include <intuition/intuition.h>

/* Include this file since you are using sprites: */
#include <graphics/gels.h>



/* We will use 32 VSprites: */
#define MAXVSPRITES 32

/* They will move one pixel each time: */
#define SPEED 1



/* Declare the functions we are going to use: */
void main();
void clean_up();



struct IntuitionBase *IntuitionBase = NULL;
/* We need to open the Graphics library since we are using sprites: */
struct GfxBase *GfxBase = NULL;


/* Declare a pointer to a Screen structure: */
struct Screen *my_screen;

/* Declare and initialize your NewScreen structure: */
struct NewScreen my_new_screen=
{
  0,            /* LeftEdge  Should always be 0. */
  0,            /* TopEdge   Top of the display.*/
  320,          /* Width     We are using a low-resolution screen. */
  200,          /* Height    Non-Interlaced NTSC (American) display. */
  2,            /* Depth     4 colours. */
  0,            /* DetailPen Text should be drawn with colour reg. 0 */
  1,            /* BlockPen  Blocks should be drawn with colour reg. 1 */
  SPRITES,      /* ViewModes No special modes. (Low-res, Non-Interlaced) */
  CUSTOMSCREEN, /* Type      Your own customized screen. */
  NULL,         /* Font      Default font. */
  "MY SCREEN",  /* Title     The screen' title. */
  NULL,         /* Gadget    Must for the moment be NULL. */
  NULL          /* BitMap    No special CustomBitMap. */
};



/* Declare a pointer to a Window structure: */
struct Window *my_window = NULL;

/* Declare and initialize your NewWindow structure: */
struct NewWindow my_new_window=
{
  0,             /* LeftEdge    x position of the window. */
  0,             /* TopEdge     y positio of the window. */
  320,           /* Width       320 pixels wide. */
  200,           /* Height      200 lines high. */
  0,             /* DetailPen   Text should be drawn with colour reg. 0 */
  1,             /* BlockPen    Blocks should be drawn with colour reg. 1 */
  CLOSEWINDOW,   /* IDCMPFlags  The window will give us a message if the */
                 /*             user has selected the Close window gad. */
  SMART_REFRESH| /* Flags       Intuition should refresh the window. */
  WINDOWCLOSE|   /*             Close Gadget. */
  WINDOWDRAG|    /*             Drag gadget. */
  WINDOWDEPTH|   /*             Depth arrange Gadgets. */
  WINDOWSIZING|  /*             Sizing Gadget. */
  ACTIVATE,      /*             The window should be Active when opened. */
  NULL,          /* FirstGadget No Custom gadgets. */
  NULL,          /* CheckMark   Use Intuition's default CheckMark. */
  "VSprites with no limitations!", /* Title */
  NULL,          /* Screen      Will later be connected to a custom scr. */
  NULL,          /* BitMap      No Custom BitMap. */
  80,            /* MinWidth    We will not allow the window to become */
  30,            /* MinHeight   smaller than 80 x 30, and not bigger */
  320,           /* MaxWidth    than 320 x 200. */
  200,           /* MaxHeight */
  CUSTOMSCREEN   /* Type        Connected to the Workbench Screen. */
};



/* 1. Declare and initialize some sprite data: */
/* (6 frames, 4 different images: 1 2 3 4 3 2) */
UWORD chip ship_data[6][28]=
{
  {
    0xFFF8, 0x0000,
    0x0200, 0x0000,
    0x877C, 0x0000,
    0x8786, 0x027C,
    0xBFBF, 0x02C6,
    0xEDFF, 0x1AC2,
    0xA57D, 0x1AFE,
    0xBF19, 0x02FE,
    0x8F12, 0x00FC,
    0x04FC, 0x0000,
    0x0809, 0x0000,
    0x3FFE, 0x0000,
  },
  {
    0x7FF0, 0x0000,
    0x0200, 0x0000,
    0x077C, 0x0000,
    0x8786, 0x027C,
    0xBFBF, 0x02C6,
    0xEDFF, 0x1AC2,
    0xA57D, 0x1AFE,
    0xBF19, 0x02FE,
    0x0F12, 0x00FC,
    0x04FC, 0x0000,
    0x0809, 0x0000,
    0x3FFE, 0x0000,
  },
  {
    0x3FE0, 0x0000,
    0x0200, 0x0000,
    0x877C, 0x0000,
    0x8786, 0x027C,
    0xBFBF, 0x02C6,
    0xEDFF, 0x1AC2,
    0xA57D, 0x1AFE,
    0xBF19, 0x02FE,
    0x8F12, 0x00FC,
    0x04FC, 0x0000,
    0x0809, 0x0000,
    0x3FFE, 0x0000,
  },
  {
    0x1FC0, 0x0000,
    0x0200, 0x0000,
    0x077C, 0x0000,
    0x8786, 0x027C,
    0xBFBF, 0x02C6,
    0xEDFF, 0x1AC2,
    0xA57D, 0x1AFE,
    0xBF19, 0x02FE,
    0x0F12, 0x00FC,
    0x04FC, 0x0000,
    0x0809, 0x0000,
    0x3FFE, 0x0000,
  },
  {
    0x3FE0, 0x0000,
    0x0200, 0x0000,
    0x877C, 0x0000,
    0x8786, 0x027C,
    0xBFBF, 0x02C6,
    0xEDFF, 0x1AC2,
    0xA57D, 0x1AFE,
    0xBF19, 0x02FE,
    0x8F12, 0x00FC,
    0x04FC, 0x0000,
    0x0809, 0x0000,
    0x3FFE, 0x0000,
  },
  {
    0x7FF0, 0x0000,
    0x0200, 0x0000,
    0x077C, 0x0000,
    0x8786, 0x027C,
    0xBFBF, 0x02C6,
    0xEDFF, 0x1AC2,
    0xA57D, 0x1AFE,
    0xBF19, 0x02FE,
    0x0F12, 0x00FC,
    0x04FC, 0x0000,
    0x0809, 0x0000,
    0x3FFE, 0x0000,
  }
};



/* 2. Declare VSprite structures: */
struct VSprite head, tail, vsprite[ MAXVSPRITES ];


/* 3. Declare the VSprites' colour tables:     */
WORD colour_table[ MAXVSPRITES ][ 3 ];


/* 4. Declare a GelsInfo structure: */
struct GelsInfo ginfo;


/* This boolean variable will tell us if the VSprites are */
/* in the list or not:                                    */
BOOL vsprite_on = FALSE;


/* This program will not open any console window if run from */
/* Workbench, but we must therefore not print anything.      */
/* Functions like printf() must therefore not be used.       */
void _main()
{
  /* The GelsInfo structure needs the following arrays: */
  WORD nextline[ 8 ];
  WORD *lastcolor[ 8 ];

  /* Direction of the sprite: */
  WORD x_direction[ MAXVSPRITES ];

  /* Boolean variable used for the while loop: */
  BOOL close_me = FALSE;

  /* Declare a pointer to an IntuiMessage structure: */
  struct IntuiMessage *my_message;

  UBYTE loop;      /* Used as counter in the for loop: */
  UBYTE image = 0; /* Which image is used, 1-6.        */
  UBYTE x = 0;     /* X and Y position.                */
  UBYTE y = 0;



  /* Open the Intuition Library: */
  IntuitionBase = (struct IntuitionBase *)
    OpenLibrary( "intuition.library", 0 );

  if( IntuitionBase == NULL )
    clean_up(); /* Could NOT open the Intuition Library! */



  /* Since we are using sprites we need to open the Graphics Library: */
  /* Open the Graphics Library: */
  GfxBase = (struct GfxBase *)
    OpenLibrary( "graphics.library", 0);

  if( GfxBase == NULL )
    clean_up(); /* Could NOT open the Graphics Library! */



  /* We will now try to open the screen: */
  my_screen = (struct Screen *) OpenScreen( &my_new_screen );

  /* Have we opened the screen succesfully? */
  if(my_screen == NULL)
    clean_up();
```

```
my_new_window.Screen = my_screen;


/* We will now try to open the window: */
my_window = (struct Window *) OpenWindow( &my_new_window );

/* Have we opened the window succesfully? */
if(my_window == NULL)
  clean_up(); /* Could NOT open the Window! */



/* 6. Initialize the GelsInfo structure: */

/* All sprites may be used to draw the VSprites: */
/* ( 11111111 = 0xFF )                            */
ginfo.sprRsrvd = 0xFF;
/* If we do not exclude the first two sprites, the mouse */
/* pointer's colours may be affected.                    */


/* Give the GelsInfo structure some memory: */
ginfo.nextLine = nextline;
ginfo.lastColor = lastcolor;


/* Give the Rastport a pointer to the GelsInfo structure: */
my_window->RPort->GelsInfo = &ginfo;


/* Give the GelsInfo structure to the system: */
InitGels( &head, &tail, &ginfo );



/* 7. Initialize the VSprite structures: */
for( loop = 0; loop < MAXVSPRITES; loop++ )
{
  /* Set the VSprite's colours: */
  colour_table[ loop ][ 0 ] = 0x0000; /* Black */
  colour_table[ loop ][ 1 ] = 0x0080; /* Dark green */
  colour_table[ loop ][ 2 ] = 0x00D0; /* Green */

  /* Set the speed and horizontal direction of the VSprite: */
  x_direction[ loop ] = SPEED;

  vsprite[ loop ].Flags = VSPRITE; /* It is a VSprite.    */
  vsprite[ loop ].X = 10 + 20 * x; /* X position.         */
  vsprite[ loop ].Y = 30 + 20 * y; /* Y position.         */
  vsprite[ loop ].Height = 12;     /* 16 lines tall.      */
  vsprite[ loop ].Width = 2;       /* 2 words wide.       */
  vsprite[ loop ].Depth = 2;       /* 2 bitpl, 4 colours. */

  /* Pointer to the sprite data: */
  vsprite[ loop ].ImageData = ship_data[ image ];

  /* Pointer to the colour table: */
  vsprite[ loop ].SprColors = colour_table[ loop ];
```

```
  /* 8. Add the VSprites to the VSprite list: */
  AddVSprite( &vsprite[ loop ], my_window->RPort );


  /* Position of the VSprites: */
  y++;
  if( y > 7 )
  {
    y = 0;
    x++;
  }
}


/* The VSprites are in the list. */
vsprite_on = TRUE;


/* Stay in the while loop until the user has selected the Close window */
/* gadget: */
while( close_me == FALSE )
{
  /* Stay in the while loop as long as we can collect messages: */
  while(my_message = (struct IntuiMessage *) GetMsg(my_window->UserPort))
  {
    if( my_message->Class == CLOSEWINDOW)
      close_me=TRUE;

    ReplyMsg( my_message );
  }


  /* Affect all VSprites: */
  for( loop = 0; loop < MAXVSPRITES; loop++ )
  {
    /* Change the x position of the VSprite: */
    vsprite[ loop ].X += x_direction[ loop ];


    /* Check that the sprite does not move outside the screen: */
    if(vsprite[ loop ].X > 300)
      x_direction[ loop ] = -SPEED;

    if(vsprite[ loop ].X < 0)
      x_direction[ loop ] = SPEED;


    /* Change the image of the VSprite: */
    vsprite[ loop ].ImageData = ship_data[ image ];
  }


  /* Image counter: */
  image++;
  if( image > 5 )
    image = 0;


  /* 9. Sort the Gels list: */
  SortGList( my_window->RPort );
```

```
    /* 10. Draw the Gels list: */
    DrawGList( my_window->RPort, &(my_screen->ViewPort) );

    /* 11. Set the Copper and redraw the display: */
    MakeScreen( my_screen );
    RethinkDisplay();
  }



  /* Free all allocated memory: (Close the window, libraries etc) */
  clean_up();

  /* THE END */
}



/* This function frees all allocated memory. */
void clean_up()
{
  UBYTE loop;

  if( vsprite_on )
    for( loop = 0; loop < MAXVSPRITES; loop++ )
      RemVSprite( &vsprite[ loop ] );

  if( my_window )
    CloseWindow( my_window );

  if(my_screen )
    CloseScreen( my_screen );

  if( GfxBase )
    CloseLibrary( GfxBase );

  if( IntuitionBase )
    CloseLibrary( IntuitionBase );

  exit();
}
```

A.14 HINTS AND TIPS

Example1

This example tell you if you have an American (NTSC) or European (PAL) system.

```
/* Example 1                                              */
/* This example tell you if you have an American (NTSC) or */
/* European (PAL) system.                                 */


/* Declares commonly used data types, such as UWORD etc: */
#include <exec/types.h>

/* This header file declares the GfxBase structure: */
#include <graphics/gfxbase.h>


/* Pointer to the GfxBase structure. NOTE! This pointer must */
/* allways be called "GfxBase"!                               */
struct GfxBase *GfxBase;


main()
{
  int loop;

  /* Open the Graphics Library: (any version) */
  GfxBase = (struct GfxBase *)
    OpenLibrary( "graphics.library", 0 );

  if( !GfxBase )
    exit(); /* ERROR! Could not open the Graphics Library! */


  if( GfxBase->DisplayFlags & NTSC )
    printf( "You have an American (NTSC) Amiga.\n" );

  if( GfxBase->DisplayFlags & PAL )
    printf( "You have an European (PAL) Amiga.\n" );


  /* Close the Graphics Library: */
  CloseLibrary( GfxBase );


  /* Wait for a while: */
  for( loop = 0; loop < 500000; loop++ )
    ;
}
```

Example2

This program will print "Hello!" in the CLI window if started from CLI, or the text will be printed in a special window that is automatically opened if run from workbench.

```
/* Example 2                                              */
/* This program will print "Hello!" in the CLI window if      */
/* started from CLI, or the text will be printed in a special */
/* window that is automatically opened if run from workbench. */

void main();

void main()
{
  int loop;

  printf( "Hello!\n" );

  /* Wait for a while: */
  for( loop = 0; loop < 500000; loop++ )
    ;
}
```

Example3

This program will not open any console window if run from workbench. The disadvantage is of course that you can not use any "console functions" such as printf().

```
/* Example 3 */
/* This program will not open any console window if run from */
/* workbench. The disadvantage is of course that you can not */
/* use any "console functions" such as printf().             */

void _main();

void _main() /* Note the special character in front of main()! */
{
  int loop;

  /* Wait for a while: */
  for( loop = 0; loop < 500000; loop++ )
    ;
}
```

Example4

This program tells you if it was run from workbench or from a CLI window.

```
/* Example 4                                             */
/* This program tells you if it was run from workbench or */
/* from a CLI window.                                    */

void main();

void main( argc, argv )
int argc;
char *argv[];
{
  int loop;


  if( argc )
    printf( "This program was started from a CLI window!\n" );
  else
    printf( "This program was started from Workbench!\n" );


  /* Wait for a while: */
  for( loop = 0; loop < 500000; loop++ )
    ;
}
```

B FUNCTIONS

B.1 INTRODUCTION

Here is the complete list of all functions described in the Manual.

B.2 INTUITION LIBRARY

The Intuition Library must have been opened before you may call these functions, and you will probably have to include the file "intution.h". For example:

```
#include <intuition/intuition.h>

struct IntuitionBase *IntuitionBase;

main()
{
  /* Open the Intuition Library: */
  IntuitionBase = (struct IntuitionBase *)
    OpenLibrary( "intuition.library", 0 );

  if( IntuitionBase == NULL )
    exit(); /* Could NOT open the Intuition Library! */


  ... ...


  /* Close the Intuition Library: */
  CloseLibrary( IntuitionBase );
}
```

AddGadget()

  This function adds a gadget to the gadget list.

  Synopsis: result = AddGadget(window, gadget, position);

```
  result:   (long) The actual position of the gadget when it
            has been added.

  window:   (struct Window *) Pointer to the window, to which
            the gadget should be added.

  gadget:   (struct Gadget *) Pointer to the gadget which will
            be added.
```

```
position: (long) Position in the gadget list. (Starts from
          zero). Eg:
            0 -> Before all other gadgets.
            1 -> After the first gadget, but before the
                 second.
            If a too big value is entered (or -1), the gadget
            will be placed last in the list.
```

Important, after your program has added the necessary gadgets, you need to call the function RefreshGadgets() in order to see your changes. You may add (or take away) several gadgets, but when you are finished you must call that function.

AddVSprite()

This function will add a VSprite to the VSprite list.

Synopsis: AddVSprite(vsprite, rp);

```
vsprite:  (struct VSprite *) Pointer to an initialized
          VSprite structure.

rp:       (struct RastPort *) Pointer to the RastPort.
```

AllocRemember()

This function allocates both memory (same as AllocMem), but will also allocate space for a Remember structure which are initialized with the size of the allocated memory, and a pointer to that memory. Each time the program allocates memory with this function, the Remember structures are linked together.

Since the Remember structures contains all necessary information about the memory, and are linked together, all memory can be deallocated with one single function call (FreeRemember()).

Synopsis: memory = AllocRemember(remember, size, type);

```
memory:   (char *) Pointer to the new allocated memory, or
          NULL if no memory could be allocated. Remember!
          Never use memory which you have not successfully
          allocated.

remember: (struct Remember **) Address of a pointer to a
          Remember structure. Before you call the
          AllocRemember() function for the first time you
          should set this pointer to NULL. (Note that it is
          a pointer to a pointer!)
```

```
  size:     (long) The size (in bytes) of the memory you want.
            (AllocMem() always allocates memory in multiples of
            eight bytes. So if you only ask for 9 bytes, Exec
            would actually give you 16 Bytes (2*8).)

  type:     (long) You need to choose one of the three
            following types of memory (see chapter 0
            INTRODUCTION for more information about Chip and
            Fast memory):

            MEMF_CHIP   Chip memory. This memory can be
                        accessed by both the main processor, as
                        well as the Chips. Graphics/Sound data
                        MUST therefore be placed in Chip memory.
                        If it does not matter what type of
                        memory you get (Fast or Chip), you
                        should try to allocate Fast memory
                        before you allocate Chip memory. (Chip
                        memory is more valuable than Fast
                        memory.)

            MEMF_FAST   Fast memory. This memory can only be
                        accessed by the main processor.
                        (Graphics and Sound data can NOT be
                        stored in Fast memory, use Chip memory.)
                        This memory is normally a little bit
                        faster than Chip memory, since only the
                        main processor is working with it, and
                        it is not disturbed by the Chips.

            MEMF_PUBLIC If it does not matter what type of
                        memory you get (you do not intend to
                        use the memory for Graphics/Sound data),
                        you should use Fast memory. However,
                        all Amigas do not have Fast memory,
                        since you need to by a memory expansion
                        in order to get it. If want to tell
                        Exec that you would like to use Fast
                        memory if there is any, else use Chip
                        memory, you should ask for MEMF_PUBLIC.

            If you want the allocated memory to be cleared
            (initialized to zeros), you should set the flag
            MEMF_CLEAR.
```

AutoRequest()

This function opens a Simple requester. Intuition will automatically activate it and take care of the response from the user. It will return TRUE if the left gadget was selected, and FALSE if the right gadget was selected.

```
  Synopsis:  result = AutoRequest( my_window, info_txt, pos_txt,
                                   neg_txt, pos_IDCMP, neg_IDCMP,
                              width, height );

  my_window: (struct Window *) Pointer to a window if there
             exist one, else NULL.

  info_txt:  (struct IntuiText *) Pointer to an IntuiText
             structure containing the "body text".

  pos_txt:   (struct IntuiText *) Pointer to an IntuiText
             structure containing the "positive text". Eg:
             "TRUE", "YES", "RETRY" etc. (Optional)

  neg_txt:   (struct IntuiText *) Pointer to an IntuiText
             structure containing the "negative text". Eg:
             "FALSE", "NO", "CANCEL" etc.

  pos_IDCMP: (long) IDCMP flags which satisfy the "positive"
             gadget. (The flag RELVERIFY is already set.)

  pos_IDCMP: (long) IDCMP flags which satisfy the "negative"
             gadget. (The flag RELVERIFY is already set.)

  width:     (long) How many pixels wide the requester should
             be.

  height:    (long) How many lines high the requester should
             be.

  result:    (long) Boolean value. The function returns TRUE if
             the positive gadget was satisfied, and FALSE if
             the negative gadget was satisfied.
```

BeginRefresh()

This function will speed up your redrawing of the window. You should call this function before you start to refresh the window, and only the parts that needs to be redrawn are redrawn.

```
  Synopsis:  BeginRefresh( my_window );

  my_window: (struct Window *) Pointer to a Window structure
             which has previously been initialized by an
             OpenWindow() call.
```

ClearDMRequest()

This function disables a Double-menu requester. The user can not open the requester any more.

```
  Synopsis:     result = ClearDMRequest( my_window );

  my_window:    (struct Window *) Pointer to the Window
```

```
                structure which the requester is connected to.
                The DMRequest pointer in the Window structure
                is set to NULL.

  result:       (long) If the function could disable the
                DM-requester it returns TRUE, else (something
                went wrong, the requester is in use etc) it
                returns FALSE.
```

ClearMenuStrip()

This function removes a menu strip from a window. Remember to always remove the menu strip before you close the window, or changes the menu strip.

```
  Synopsis:   ClearMenuStrip( my_window );

  my_window:  (struct Window *) Pointer to the window which
              menu strip should be removed.
```

ClearPointer()

This will remove the "custom" pointer, and replace it with Intuition's default pointer.

```
  Synopsis:  ClearPointer( my_window );

  my_window: (struct Window *) Pointer to a Window structure
             which has previously been initialized by an
             OpenWindow() call.
```

CloseScreen()

This function will close a Custom Screen which you have previously opened.

```
  Synopsis:      CloseScreen( my_screen );

  my_screen:     (struct Screen *) Pointer to an already opened
                 screen.
```

All windows (See chapter 2 WINDOWS for more information) on your Screen MUST have been closed before you may close the screen. If you close a window after the screen has been closed, the system will crash. (Not recommended.)

If there does not exist any more screens when you close yours, Intuition will automatically reopen the Workbench Screen.

CloseWindow()

This function will close a window you have previously opened. Remember that you need to close all windows connected to a screen before you may close the screen, and all opened windows must have been closed before your program quits.

```
Synopsis:  CloseWindow( my_window );

my_window: (struct Window *) Pointer to a Window structure
           which has previously been initialized by an
           OpenWindow() call.
```

CloseWorkBench()

This function will try to close the Workbench Screen if possible. If any other programs is using the Workbench Screen, the function can not close it. Closing the Workbench will free some memory, and can therefore be used if your program needs more memory.

(Remember to reopen the Workbench Screen when your program terminates.)

```
Synopsis:      result = CloseWorkBench();

result:        (long) A boolean value which tell us if the
               Workbench screen has been (or already was)
               closed (TRUE), or not (FALSE).
```

CurrentTime()

This function gives the current time.

```
Synopsis: CurrentTime( seconds, micros );

seconds:  (long *) Pointer to an ULONG variable which will be
          initialized with the current seconds stamp.

micros:   (long *) Pointer to an ULONG variable which will be
          initialized with the current micros stamp.
```

DisplayAlert()

This function activates an Alert message.

```
Synopsis: result = DisplayAlert( nr, message, height );

nr:       (long)  Value which describes if it is a
          RECOVERY_ALERT or a DEADEND_ALERT.
```

message: (char *) Pointer to an array of characters (char). It contains the strings we want to display, and some extra information (position etc). The string itself is divided into substrings, which all contain information about its position etc.

- 2 bytes (16-bit) which are used for the x position of the text.
- 1 byte (8-bit) which is used for the y position of the text.
- The text string which ends with a NULL ('\0') sign.
- A Continuation byte. If it is TRUE there is another substring after this one, else this was the last substring.

height: (long) The height of the Alert box.

result: (long) The function DisplayAlert() returns a boolean value. If it is a RECOVERY_ALERT and the user pressed the left mouse button it returns TRUE else, if the user pressed the right mouse button, it returns FALSE. If it is a DEADEND_ALERT the function will immediate return FALSE.

DisplayBeep()

This function flashes the screen's colours. Can be used whenever you want to catch the user's attention.

Synopsis: `DisplayBeep( screen );`

screen: (struct Screen *) Pointer to the screen, which colours you want to flash. If you have not opened a screen yourself (you are using the Workbench Screen), you can find a pointer to that screen in the Window structure: (my_window is a pointer to an opened window)

```
DisplayBeep( my_window->WScreen );
```

DoubleClick()

This function checks if the user double-clicked on one of the mouse buttons. You give the function the current as well as the previous time when the button was pressed, and it will check the preferences and return TRUE if the two button events happened within the time limit.

Synopsis: `double = DoubleClick( sec1, mic1, sec2, mic2 );`

double: (long) If the two button events happened within the current time limit, the function will return TRUE, else it will return FALSE.

```
  sec1:     (long) Time (seconds) when the button was pressed
            for the first time.

  mic1:     (long) Time (micros) when the button was pressed
            for the first time.

  sec2:     (long) Current time (seconds).

  mic2:     (long) Current Time (micros).
```

DrawBorder()

```
  This function draws the specified Borders into a RastPort
  (Screen/Window).

  Synopsis:  DrawBorder( rast_port, border, x, y );

  rast_port: (struct RastPort *) Pointer to a RastPort.

             If the lines should be drawn in a window, and
             my_window is a pointer to that window, you write:
             my_window->RPort.

             If the lines should be drawn in a Screen, and
             my_screen is a pointer to that screen, you write:
             my_screen->RastPort.

  border:    (struct Border *) Pointer to a Border structure
             which has been initialized with your requirements.

  x:         (long) Number of pixels added to the x coordinates.

  y:         (long) Number of lines added to the y coordinates.
```

DrawGList()

```
  This function will draw the VSprites into the specified
  Rastport.

  Synopsis: DrawGList( rp, vp );

  rp:       (struct RastPort *) Pointer to the RastPort.

  vp:       (struct ViewPort *) Pointer to the ViewPort.
```

DrawImage()

```
  This function draws the specified images into a RastPort
  (Screen/Window).
  Synopsis:  DrawImage( rast_port, image, x, y );

  rast_port: (struct RastPort *) Pointer to a RastPort.

             If the images should be drawn in a window, and
             my_window is a pointer to that window, you write:
             my_window->RPort.

             If the images should be drawn in a Screen, and
             my_screen is a pointer to that screen, you write:
             my_screen->RastPort.

  image:     (struct Image *) Pointer to an Image structure
             which has been initialized with your requirements.

  x:         (long) Number of pixels added to the x position of
             the image.

  y:         (long) Number of lines added to the y position of
             the image.
```

EndRefresh()

This function will tell Intuition that you have finished with your redrawings. IMPORTANT! If you receive a REFRESHWINDOW message, you must call the functions BeginRefresh() and EndRefresh(), even if you do not want to redraw anything.

```
  Synopsis:  EndRefresh( my_window );

  my_window: (struct Window *) Pointer to a Window structure
             which has previously been initialized by an
             OpenWindow() call.
```

EndRequest()

This function deactivates a requester which has been activated.

```
  Synopsis:     EndRequest( my_requester, my_window );

  my_requester: (struct Requester *) Pointer to the Requester
                structure which will be removed.

  my_window:    (struct Window *) Pointer to the Window
                structure which the requester is connected to.
```

FreeRemember()

This function deallocates all memory which has been allocated by the AllocRemember() function. Note, you can deallocate all

Remember structures only, and deallocate the memory yourself, if you want to.

```
Synopsis:   FreeRemember( remember, everything );

remember:   (struct Remember **) Address of a pointer to the
            first Remember structure (initialized by the
            AllocRemember() function). (Note that it is a
            pointer to a pointer!)

everything: (long) A boolean value. If everything is equal
            to TRUE, all memory (both the allocated memory
            and the Remember structures) are deallocated.
            However, if everything is equal to FALSE, only
            the Remember structures are deallocated, and you
            have to deallocate the memory yourself.
```

GetDefPrefs()

This function makes a copy of the default Preferences structure.

```
Synopsis: pref = GetPrefs( buffer, size );

pref:     (struct Preferences *) Pointer to the default
          preferences. If the function could not make a copy
          of the preferences, the function returns NULL.

buffer:   (struct Preferences *) Pointer to the memory buffer
          which should be used to store a copy of the default
          preferences in.

size:     (long) The number of bytes you want to copy to the
          buffer. Important, the buffer must be at least as
          big as the number of bytes you want to copy.
```

GetMsg()

This function tries to get a message from a message port.

```
Synopsis:        my_message = GetMsg( my_message_port );

my_message:      (struct Message *) Pointer to a Message
                 structure, in this case a pointer to an
                 IntuiMessage structure, or NULL if no
                 message was collected.

my_message_port: (struct MsgPort *) Pointer to an MsgPort. If
                 you have opened a window, you can find your
                 window's message port in the Window
                 structure. ( my_window->UserPort )
```

```
GetPrefs()

  This function makes a copy of the Preferences structure.

  Synopsis: pref = GetPrefs( buffer, size );

  pref:     (struct Preferences *) Pointer to your preferences.
            Same as your memory pointer (buffer), but is
            returned so you can check if you got a copy or not.
            If you could not get a copy of the preferences, the
            function returns NULL.

  buffer:   (struct Preferences *) Pointer to the memory buffer
            which should be used to store a copy of the
            preferences in.

  size:     (long) The number of bytes you want to copy to the
            buffer. Important, the buffer must be at least as
            big as the number of bytes you want to copy.



InitGels()

  This function "gives" an already prepared GelsInfo structure
  to the system.

  Synopsis: InitGels( head, tail, ginfo );

  head:     (struct VSprite *) Pointer to the first "dummy"
            VSprite structure.

  tail:     (struct VSprite *) Pointer to the second "dummy"
            VSprite structure.

  ginfo:    (struct GelsInfo *) Pointer to an initialized GelsInfo
            structure.



ItemAddress()

  This function returns a pointer to the Menu or Item
  structure which is specified by the menu number.

  Synopsis:    ItemAddress( my_menu, menu_number );

  my_menu:     (struct Menu *) Pointer to the first Menu
               structure in the menu strip.

  menu_number: (USHORT) This menu number specifies a subitem/
               item/menu.


Lock()
```

This function "locks" a file so no other processes may alter the contents (SHARED_LOCK). You can even prevent other processes to read the file (EXCLUSIVE_LOCK).

Synopsis: lock = Lock(name, mode);

lock: (BPTR) Actually a pointer to a FileLock structure.

name: (char *) Pointer to a text string which contains the file/directory name.

```
mode:     (long) Accessmode:
            SHARED_LOCK:     Other tasks may read the file.
            ACCESS_READ:                   - " -
            EXCLUSIVE_LOCK:  No other tasks may use this f.
            ACCESS_WRITE:                  - " -
```

ModifyIDCMP()

This function changes the Window structure's IDCMPFlags field.

Synopsis: ModifyIDCMP(my_window, IDCMPFlags);

my_window: (struct Window *) Pointer to an already opened window.

IDCMPFlags: (long) None or more IDCMP flags.

If you call this function with no IDCMP flags set, the window's IDCMP Ports will be closed. On the other hand, if you call this function, with one or more IDCMP flags set, a Port will be, if necessary, opened for you.

ModifyProp()

This function modifies a proportional gadget's values and knob. For example, if your program is reading files from the disk, VertBody was maybe equal to 0xFFFF (MAXBODY) in the beginning, but as more files are collected from the disk, you maybe want to change the size of the knob etc. You then simply call this function and it will change the values as well as redraw the gadget.

```
Synopsis:    ModifyProp( gadget, window, requester, flags,
             horiz_pot, vert_pot, horiz_body, vert_body );
```

gadget: (struct Gadget *) Pointer to the proportional gadget which should be changed and redrawn.

window: (struct Window *) Pointer to the window which the proportional gadget is connected to.

requester: (struct Requester *) If the gadget is connected to a requester, set this pointer to point to that requester, else NULL. Important, if this gadget is connected to a requester, it must be displayed when you execute this command!

flags: (long) Here is the list of all flags you may use:

FREEHORIZ	Set this bit if you want the user to be able to move the knob horizontally.
FREEVERT	Set this bit if you want the user to be able to move the knob vertically.
AUTOKNOB	Set this bit if you want that the size of the knob to be controlled by Intuition. (HorizBody and VertBody affects the size of the Autoknob.) - If you want to use Intuition's Autoknob you should give GadgetRender a pointer to an Image structure. (You do not need to initialize the Image structure since Intuition takes care of it.) - If you on the other hand would like to use your own knob image, you give GadgetRender a pointer to your Image structure, which you have initialized yourself.
PROPBORDERLESS	Set this bit if you do not want any border around the container.

(See chapter 4.7 for more information.)

horiz_pot: (long) This variable contains the actual (horizontally) proportional value. If the knob should be moved 25% to the right, HorizPot should be set to 25% of MAXPOT (0xFFFF). (0xFFFF * 0.25 = 0x3FFF)

vert_pot: (long) Same as HorizPot except that this is the vertically proportional value.

horiz_body: (long) Describes how much HorizPot should change every time the user clicks inside the container. If the volume of a melody can be between 0-63

(64 steps), HorizPot should change 1/64 each time. The HorizBody should therefore be set to: 1/64 * MAXBODY (0xFFFF) == 3FF

HorizBody describes also how much the user can see/use of the entire data. For example, if you have a list of 32 file names, and the user only can see 8 names at one time (25%), the knob (AUTOKNOB) should fill 25% of the container. HorizBody should in this case be set to: MAXBODY * 8 / 32 (25% of 0xFFFF) == 3FFFF

vert_body: Same as HorizBody except that it affects VertPot, and the vertical size of the knob (AUTOKNOB).

MoveScreen()

This function will move the screen. For the moment you may only move it vertically.

Synopsis: MoveScreen(my_screen, delta_x, delta_y);

my_screen: (struct Screen *) Pointer to the screen which you want to move.

delta_x: (long) Number of pixels which the screen should move horizontally. For the moment you may not move a screen horizontally, set it therefore to 0.

delta_y: (long) Number of lines which the screen should move vertically.

MoveWindow()

This function will move a window. It has the same effect as if the user would have moved the window by using the Drag Gadget.

Synopsis: MoveWindow(my_window, delta_x, delta_y);

my_window: (struct Window *) Pointer to a Window structure which has previously been initialized by an OpenWindow() call.

delta_x: (long) Deltamovement horizontally.

delta_y: (long) Deltamovement vertically.

MrgCop()

This function reorganizes the Copper list. This is why each VSprite can have its own individual colour values.

Synopsis: MrgCop(view);

```
view:     (struct View *) Pointer to the View structure which
          copper list should be changed.
```

OffGadget()

This function disables a gadget (sets the GADGDISABLED bit in the gadget structure's Flags field):

Synopsis: OffGadget(gadget, window, requester);

```
gadget:    (struct Gadget *) Pointer to the gadget which will
           be disabled.

window:    (struct Window *) Pointer to the window that the
           gadget is attached to.

requester: (struct Requester *) If the gadget is connected to
           a requester, set this pointer to point to that
           requester, else NULL. Important, if this gadget is
           connected to a requester, it must be displayed
           when you execute this command!
```

OnGadget()

This function enables a gadget (removes the GADGDISABLED bit in the gadget structure's Flags field):

Synopsis: OnGadget(gadget, window, requester);

```
gadget:     (struct Gadget *) Pointer to the gadget which
            will be enabled.

window:     (struct Window *) Pointer to the window that the
            gadget is attached to.

requester:  (struct Requester *) If the gadget is connected
            to a requester, set this pointer to point to that
            requester, else NULL. Important, if this gadget
            is connected to a requester, it must be displayed
            when you execute this command!
```

Remember, as long as the gadget is disabled the user can not select it, and it will not broadcast any messages. A disabled gadget is drawn as usual except that it "ghosted".

```
OffMenu()

  This function can disable a subitem, an item or even a whole
  menu. The image or text of the disabled items etc will be
  "ghosted", and the user can not select them.

  Synopsis:    OffMenu( my_window, menu_number );

  my_window:   (struct Window *) Pointer to the window which
               the menu strip is connected to.

  menu_number: (USHORT) This menu number specifies what should
               be disabled. Use the macros SHIFTMENU, SHIFTITEM
               and SHIFTSUB to calculate the correct menu
               number. If you just specify a menu, all items
               to that menu will be disabled. If you specify
               a menu and an item, that item will be disabled,
               and so all subitems connected to it if there are
               any.



OnMenu()

  This function can enable a subitem, an item or even a whole
  menu. The image or text of the enabled items etc, will become
  normal (not "ghosted") and the user can now select them.

  Synopsis:    OnMenu( my_window, menu_number );

  my_window:   (struct Window *) Pointer to the window which
               the menu strip is connected to.

  menu_number: (USHORT) This menu number specifies what should
               be enabled. Use the macros SHIFTMENU, SHIFTITEM
               and SHIFTSUB to calculate the correct menu
               number. If you just specify a menu, all items to
               that menu will be enabled. If you specify a menu
               and an item, that item will be enabled, so all
               subitem connected to it if there are any.



OpenScreen()

  This function will open a Custom Screen with your
  requirements.

  Synopsis:      my_screen = OpenScreen( my_new_screen );

  my_screen:     (struct Screen *) Pointer to a Screen
                 structure. It will point to your newly opened
                 screen or be equal to NULL if the screen could
                 not be opened.

  my_new_screen: (struct NewScreen *) Pointer to a NewScreen
```

structure which contains your preferences.

OpenWindow()

This function will open a window with the characteristics defined in the NewWindow structure. It returns a pointer to a Window structure.

If you are going to use the Workbench screen, and it has been closed, it will automatically reopen. If you on the other hand is going to connect the window to a Custom screen, you need to open it yourself before calling the OpenWindow() function.

Synopsis:	my_window = OpenWindow(my_new_window);
my_window:	(struct Window *) Pointer to a Window structure or NULL if the window could not be opened.
my_new_window:	(struct NewWindow *) Pointer to a NewWindow structure which has been initialized with your requirements.

OpenWorkBench()

This function will try to open the Workbench Screen if there exist enough memory.

Synopsis:	result = OpenWorkBench();
result:	(long) A boolean value which tell us if the Workbench Screen has been (or already was) opened (TRUE), or not (FALSE).

PrintIText()

This function prints text into a RastPort (Screen/Window).

Synopsis:	PrintIText(rast_port, intui_text, x, y);
rast_port:	(struct RastPort *) Pointer to a RastPort. If the text should be printed in a window, and my_window is a pointer to that window, you write: my_window->RPort. If the text should be printed in a Screen, and my_screen is a pointer to that screen, you write: my_screen->RastPort.
intui_text:	(struct IntuiText *) Pointer to a IntuiText

```
             structure which has been initialized with your
             requirements.

  x:         (long) Number of pixels added to the x position
             of the characters.

  y:         (long) Number of lines added to the y position
             of the characters.
```

RefreshGadgets()

This function redraws all the gadgets in the list, starting by the specified gadget. If you for example has added or deleted a gadget you need to call this function to see the changes. On the other hand, if you have changed the imagery of a gadget, or the gadget's image has been trashed by something, you can also use this function to refresh the display.

```
  Synopsis:  RefreshGadgets( gadget, window, requester);

  gadget:    (struct Gadget *) Pointer to the gadget where the
             redrawing should start. This gadget, and all the
             following gadgets in the list will be redrawn.

  window:    (struct Window *) Pointer to the window which the
             gadgets are connected to.

  requester: (struct Requester *) If the gadget is connected to
             a requester, set this pointer to point to that
             requester, else NULL. Important, if this gadget is
             connected to a requester, it must be displayed
             when you execute this command! (See chapter 5
             REQUESTERS for more information about requesters.)
```

RemoveGadget()

This function removes a gadget from the list:

```
  Synopsis: result = RemoveGadget( window, gadget );

  result:   (long) The position of the removed gadget or -1 if
            something went wrong.

  window:   (struct Window *) Pointer to the window that the
            gadget is connected to.

  gadget:   (struct Gadget *) Pointer to the gadget which will
            be removed.
```

Important, after your program has removed the necessary gadgets, you need to call the function RefreshGadgets() in order to see your changes. You may take away (or add) several gadgets, but when you are finished you must call that function.

ReplyMsg()

This function tells Intuition that you have finished reading the message. Remember, once you have replied you may not examine or change the IntuiMessage structure any more.

```
Synopsis:   ReplyMsg( my_message );

my_message: (struct Message *) Pointer to a Message
            structure, in this case a pointer to an
            IntuiMessage structure.
```

ReportMouse()

You can call this function if you want the window to start/ stop reporting the mouse position. (See chapter 8 IDCMP for more information about REPORTMOUSE.)

```
Synopsis:  ReportMouse( my_window, boolean );

my_window: (struct Window *) Pointer to a Window structure
           which has previously been initialized by an
           OpenWindow() call.

boolean:   (long) Set to TRUE if you want the window to start
           reporting mouse position, else set to FALSE, and
           the window will stop reporting.
```

Request()

This function activates a requester connected to a window.

```
Synopsis:     result = Request( my_requester, my_window );

my_requester: (struct Requester *) Pointer to the Requester
              structure.

my_window:    (struct Window *) Pointer to the Window
              structure which the requester should be
              connected to.

result:       (long) Boolean value returned. If Intuition
              could successfully open the requester the
              function returns TRUE, else (something went
              wrong, not enough memory etc) the function
              returns FALSE.
```

ScreenToBack()

This will move the screen behind all other screens.

```
Synopsis:      ScreenToBack( my_screen );

my_screen:     (struct Screen *) Pointer to the screen which
               you want to move.
```

ScreenToFront()

This will move the screen in front of all other screens.

```
Synopsis:      ScreenToFront( my_screen );

my_screen:     (struct Screen *) Pointer to the screen which
               you want to move.
```

SetDMRequest()

This function allows the user to activate a Double-menu requester by clicking twice on the mouse menu button.

```
Synopsis:  result = SetDMRequest( window, requester );

window:    (struct Window *) Pointer to the Window structure
           which the requester should be connected to.

requester: (struct Requester *) Pointer to the Requester
           structure.

result:    (long) Boolean value returned. If Intuition could
           successfully open the requester the function
           returns TRUE, else (something went wrong, not
           enough memory or a DM requester is already
           connected to the window, etc) the function returns
           FALSE.
```

SetMenuStrip()

This function connects a menu strip to a window. Remember that the window must have been opened before you may connect a menu strip to that window.

```
Synopsis:   SetMenuStrip( my_window, my_menu );

my_window:  (struct Window *) Pointer to the window which the
            menu strip should be connected to.
```

```
  my_menu:    (struct Menu *) Pointer to the first Menu
              structure in the menu strip.
```

SetPointer()

This function allows you to change the window's pointer.

```
  Synopsis:  SetPointer( my_window, data, height, width, x, y );

  my_window: (struct Window *) Pointer to a Window structure
             which has previously been initialized by an
             OpenWindow() call.

  data:      (short *) Pointer to the Sprite data.

  width:     (long) The width of the pointer. Less or equal
             to 16.

  height:    (long) The height of the pointer. Can be any
             height.

  x:        (long) The pointer's "Hot Spot" x position.

  y:        (long) The pointer's "Hot Spot" y position.
```

SetPrefs()

This function saves a modified preferences structure. Do NOT change the preferences unless the user really WANTS to!

```
  Synopsis: SetPrefs( pref, size, doit );

  pref:     (struct Preferences *) Pointer to your modified
            Preferences structure.

  size:     (long) The number of bytes you want to change.

  doit:     (long) Boolean value which if FALSE, changes the
            preferences, but will not send a NEWPREFS message.
            If doit is equal to TRUE, the settings will be
            changed, and a NEWPREFS message will be sent.
            As long as the user is changing the values, doit
            should be FALSE, but when the user has finished,
            set it to TRUE, and all programs will get a NEWPREFS
            message.
```

SetWindowTitles()

This function allows you to change the window title after the window has been opened.

```
  Synopsis:  SetWindowTitles( my_window, window_t, screen_t );

  my_window: (struct Window *) Pointer to a Window structure
             which has previously been initialized by an
             OpenWindow() call.

  window_t:  (char *) Pointer to a NULL-terminated string which
             will become the window's title, or
                0 : clear title bar, or
               -1 : keep the old title.

  screen_t:  (char *) Pointer to a NULL-terminated string which
             will become the window's screen title, or
                0 : clear title bar, or
               -1 : keep the old title.
```

ShowTitle()

This function will make the screen's Title appear above or behind any Backdrop Windows (See chapter 2 WINDOWS for more information about Backdrop Windows). (The screen's title appear always behind normal windows.)

```
  Synopsis:      ShowTitle( my_screen, show_it );

  my_screen:     (struct Screen *) Pointer to the screen.

  show_it:       (long) A boolean value which can be:
                 TRUE:  The title will be in front of any
                        Backdrop Windows, but behind any
                        other windows.
                 FALSE: The Title will be behind any windows
```

SizeWindow()

This function will change the size of the window as desired. It has the same effect as if the user would have resized the window by using the Size Gadget.

```
  Synopsis:  SizeWindow( my_window, delta_x, delta_y );

  my_window: (struct Window *) Pointer to a Window structure
             which has previously been initialized by an
             OpenWindow() call.

  delta_x:   (long) Number of pixels the horizontally size of
             the window will change.

  delta_y:   (long) Number of pixels the vertically size of the
             window will change.
```

SortGList()

This function will reorganize the VSprite list so that the further down on the display the sprites are positioned the later they will appear in the list.

```
Synopsis: SortGList( rp );

rp:       (struct RastPort *) Pointer to the RastPort.
```

WBenchToBack()

This will move the Workbench Screen behind all other screens.

```
Synopsis:      result = WBenchToBack();

result:        (long) A boolean value which is TRUE if the
               Workbench screen was open, or FALSE it it was
               not.
```

WBenchToFront()

This will move the Workbench Screen in front of all other screens.

```
Synopsis:      result = WBenchToFront();

result:        (long) A boolean value which is TRUE if the
               Workbench screen was open, or FALSE it it was
               not.
```

WindowLimits()

This function will change the maximum/minimum size limits of the window. Any values which are set to 0 will remain unchanged.

```
Synopsis:  WindowLimits( my_window, min_w, min_h, max_w, max_h );

my_window: (struct Window *) Pointer to a Window structure
           which has previously been initialized by an
           OpenWindow() call.

min_w:     (long) Minimum width of the window.

min_h:     (long) Minimum height of the window.

max_w:     (long) Maximum width of the window.

max_h:     (long) Maximum height of the window.
```

WindowToFront()

  This function will put the window in front of all other
  windows.

```
  Synopsis:  WindowToFront( my_window );

  my_window: (struct Window *) Pointer to a Window structure
             which has previously been initialized by an
             OpenWindow() call.
```

WindowToBack()

  This function will push the window behind all other windows.

```
  Synopsis:  WindowToBack( my_window );

  my_window: (struct Window *) Pointer to a Window structure
             which has previously been initialized by an
             OpenWindow() call.
```

B.3 GRAPHICS LIBRARY

The Graphics Library must have been opened before you may call these functions. For example:

```
struct GfxBase *GfxBase;

main()
{
  /* Open the Graphics Library: */
  GfxBase = (struct GfxBase *)
    OpenLibrary( "graphics.library", 0 );

  if( GfxBase == NULL )
    exit(); /* Could NOT open the Graphics Library! */


  ... ...


  /* Close the Graphics Library: */
  CloseLibrary( GfxBase );
}
```

AllocRaster()

  This function reserves display memory (one BitPlane).

  Synopsis: pointer = AllocRaster(width, height);

  pointer   (PLANEPTR) Pointer to the allocated memory or NULL
            if enough memory could not be reserved.

  width: (long) The width of the BitMap.

  height: (long) The height of the BitMap.

AreaDraw()

  This function will add a new vertex to the vector list.

  Synopsis: AreaDraw(rast_port, x, y);

  rast_port: (struct RastPort *) Pointer to the RastPort that
             should be affected.

  x: (long) New X position.

  y: (long) New Y position.

AreaEnd()

  This function will close, draw and fill the polygon.

  Synopsis: AreaEnd(rast_port);

  rast_port: (struct RastPort *) Pointer to the RastPort that
             should be affected.

AreaMove()

  This function will start a new polygon.

  Synopsis: AreaMove(rast_port, x, y);

  rast_port: (struct RastPort *) Pointer to the RastPort that
             should be affected.

  x: (long) Start X position.

  y: (long) Start Y position.

BltBitMap()

  This function copies parts of BitMaps directly without
  worrying about overlapping layers.

Synopsis:	BltBitMap(sb, sx, sy, db, dx, dy, w, h, fl, m, t);
sb:	(struct BitMap *) Pointer to the "source" BitMap.
sx:	(long) X offset, source.
sy:	(long) Y offset, source.
db:	(struct BitMap *) Pointer to the "destination" BitMap.
dx:	(long) X offset, destination.
dy:	(long) Y offset, destination.
w:	(long) The width of the memory area that should be copied.
h:	(long) The height of the memory area that should be copied.
fl:	(long) The four leftmost bits tells the blitter what kind of logically operations should be done.
m:	(long) You can here define a BitMap mask, and tell the blitter which BitPlanes should be used, and which should not. The first bit represents the first BitPlane, the second bit the second BitPlane and so on. If the bit is on (1) the corresponding BitPlane will be used, else (0) the BitPlane will not be used. To turn off BitPlane zero and two, set the mask value to 0xFA (11111010). To use all BitPlanes set the mask value to 0xFF (11111111).
t:	(char *) If the copy overlaps and this pointer points to some chip-memory, the memory will be used to store the temporary area in. However, normally you do not need to bother about this value.

BltClear()

This function clears large rectangular memory areas. This function work together with the blitter and is therefore very fast.

Synopsis:	BltClear(pointer, bytes, flags);
pointer	(char *) Pointer to the memory.
bytes:	(long) The lower 16 bits tells the blitter how many bytes per row, and the upper 16 bits how many rows. This value is automatically calculated for you with help of the macro RASSIZE(). Just give RASSIZE() the correct width and height and it will return the

```
            correct value. [RASSIZE() is defined in file
            "gfx.h".]

  flags:    (long) Set bit 0 to force the function to wait
            until the Blitter has finished with your request.
```

BNDROFF()

This macro (declared in file "gfxmacro.h") will turn off the outline mode.

```
  Synopsis:  BNDROFF( rast_port );

  rast_port: Pointer to the RastPort which outlinefunction
             should be turned off.
```

ClipBlit()

This function copies parts of BitMaps with help of Rastports and will therefore care about overlapping layers, and should be used if you have windows on your display.

```
  Synopsis: ClipBlit( srp, sx, sy, drp, dx, dy, w, h, flag );

  srp:      (struct RastPort *) Pointer to the "source"
            RastPort.

  sx:       (long) X offset, source.

  sy:       (long) Y offset, source.

  drp:      (struct RastPort *) Pointer to the "destination"
            RastPort.

  dx:       (long) X offset, destination.

  dy:       (long) Y offset, destination.

  w:        (long) The width of the memory area that should be
            copied.

  h:        (long) The height of the memory area that should be
            copied.

  flag:     (long) This value tells the blitter what kind of
            logically operations should be done. See below for
            more information.
```

Draw()

This function draws single lines from the current position

```
  to the new specified position.

  Synopsis:  Draw( rast_port, x, y );

  rast_port: (struct RastPort *) Pointer to the RastPort that
             should be affected.

  x:         (long) The new X position.

  y:         (long) The new Y position.



Flood()

  This function will flood fill complicated objects.

  Synopsis:  Flood( rast_port, mode, x, y );

  rast_port: (struct RastPort *) Pointer to the RastPort that
             should be affected.

  mode:      (long) Which mode should be used. If you want to
             use the Colour mode set the mode variable to 1, to
             get the Outline mode set the mode variable to 0.

  x:         (long) X position where the flood fill should
             start.

  y:         (long) Y position where the flood fill should
             start.



FreeColorMap()

  This function deallocates the memory that was allocated by
  the GetColorMap() function. Remember to deallocate all memory
  that you allocate. For every GetColorMap() function there
  should be one FreeColorMap() function.

  FreeColorMap( colormap );

  colormap: (struct ColorMap *) Pointer to a ColorMap structure
            that GetColorMap() returned and you now want to
            deallocate.



FreeCprList()

  This function will return all memory that was automatically
  allocated by the MrgCop() function.

  Synopsis: FreeCprList( cprlist );

  cprlist:  (struct cprlist *) Pointer to the View's cprlist
```

(LOFCprList) structure. If the View was interlaced you must also call the FreeCprList function with a pointer to the SHFCprList.

FreeRaster()

This function will deallocate display memory (BitPlane). Remember to deallocate all BitPlanes!

Synopsis: FreeRaster(bitplane, width, height);

bitplane:	(PLANEPTR) Pointer to a Bitplane.
width:	(long) The Bitplane's width.
height:	(long) The Bitplane's height.

FreeVPortCopLists()

This function will return all memory that was automatically allocated by the MakeVPort() function. Remember to call FreeVPortCopLists() for every ViewPort you have created!

Synopsis: FreeVPortCopLists(viewport);

view:	(struct ViewPort *) Pointer to the ViewPort.

GetColorMap()

This function allocates and initializes a ColorMap structure.

Synopsis: colormap = GetColorMap(colours);

colormap:	(struct ColorMap *) GetColorMap returns a pointer to the ColorMap structure it has allocated and initialized, or NULL if not enough memory.
colours:	(long) A value specifying how many colours you want that the ColorMap structure should store. (1, 2, 4, 8, 16, 32)

InitBitMap()

This function initializes a BitMap structure.

Synopsis: InitBitMap(bitmap, depth, width, height);

bitmap: (struct BitMap *) Pointer to the BitMap.

```
  depth:  (long) How many BitPlanes used.

  width:  (long) The width of the raster.

  height: (long) The height of the raster.



InitRastPort()

  This function initializes a RastPort.

  Synopsis:  InitRastPort( rast_port );

  rast_port: (RastPort *) Pointer to the RastPort that should
             be Initialized.



InitView()

  This function will initialize a View structure.

  Synopsis: InitView( view );

  view:     (struct View *) Pointer to the View that should be
            initialized.



InitVPort()

  This function will initialize a ViewPort structure.

  Synopsis:  InitVPort( view_port );

  view_port: (struct ViewPort *) Pointer to the ViewPort that
             should be initialized.



MakeVPort()

  This function prepares the Amiga's hardware (especially the
  Copper) to display a ViewPort. NOTE! You have to prepare
  EVERY ViewPort you are going to use!

  Synopsis: MakeVPort( view, viewport );

  view:     (struct View *) Pointer to the ViewPort's View.

  viewport: (struct ViewPort *) Pointer to the ViewPort.



Move()
```

This function moves the cursor.

```
Synopsis:  Move( rast_port, x, y );

rast_port: (struct RastPort *) Pointer to the RastPort that
           should be affected.

x:         (long) The new X position.

y:         (long) The new Y position.
```

MrgCop()

This function puts together all displayinstructions and prepares the view to be showed.

```
Synopsis: MrgCop( view );

view:     (struct View *) Pointer to the View.
```

LoadView()

This function will start showing a View. Remember that when you close your View you must switch back to the old view. (See examples for more details.)

```
Synopsis: LoadView( view );

view:     (struct View *) Pointer to the View.
```

PolyDraw()

This function will draw multiple lines.

```
Synopsis:    PolyDraw( rast_port, number, coordinates );

rast_port:   (struct RastPort *) Pointer to the RastPort that
             should be affected.

number:      (long) The number of coordinates (x,y) defined
             in the array.

coordinates: (short *) Pointer to an array of coordinates.
```

ReadPixel()

This function reads the colour value of a pixel.

```
Synopsis:  colour = ReadPixel( rast_port, x, y );
```

```
  colour:    (long) ReadPixel returns the colour value of the
             specified pixel (colour 0 - 255 ) or -1 if the
             coordinates were outside the Raster.

  rast_port: (struct RastPort *) Pointer to the RastPort which
             contain the pixel you want to examine.

  x:         (long) X position of the pixel.

  y:         (long) Y position of the pixel.
```

RectFill()

This function will draw filled rectangles.

```
  Synopsis:  RectFill( rast_port, minx, miny, maxx, maxy );

  rast_port: (struct RastPort *) Pointer to the RastPort that
             should be affected.

  minx:      (long) Left position of the rectangle.

  miny:      (long) Top          - " -

  maxx:      (long) Right        - " -

  maxy:      (long) Bottom       - " -
```

SetAfPt()

This function will set the area pattern:

```
  Synopsis:     SetAfPt( rast_port, area_pattern, pow2 );

  rast_port:    (struct RastPort *) Pointer to the RastPort
                that should be affected.

  area_pattern: (UWORD) Pointer to an array of UWORDS that
                generate the pattern. Each bit in the array
                represents one dot.

  pow2:         (BYTE) The pattern must be two to the power of
                pow2 lines tall. If the pattern is one line tall
                 pow2 should be set to 0, if the pattern is two
                lines tall pow2 should be set to 1, if the
                pattern is four lines tall pow2 should be set to
                2, and so on. (If you use multicoloured patterns
                the pow2 should be negative. A sixteen lines
                tall multicoloured pattern should therefore have
                the pow2 value set to -4 [2^4 = 16].)
```

```
SetAPen()

  This function will change the FgPen's colour.

  Synopsis:   SetAPen( rast_port, new_colour );

  rast_port:  (struct RastPort *) Pointer to the RastPort that
              should be affected.

  new_colour: (long) A new colour value.



SetBPen()

  This function will change the BgPen's colour.

  Synopsis:   SetBPen( rast_port, new_colour );

  rast_port:  (struct RastPort *) Pointer to the RastPort that
              should be affected.

  new_colour: (long) A new colour value.



SetDrMd()

  This function will change the drawing mode.

  Synopsis:  SetDrMd( rast_port, new_mode );

  rast_port: (struct RastPort *) Pointer to the RastPort that
             should be affected.

  new_mode:  (long) The new drawing mode. Set one of the
             following: JAM1, JAM2, COMPLEMENT, INVERSVID|JAM1
             or INVERSVID|JAM2.

             JAM1           The FgPen will be used, the
                            background unchanged. (One colour
                            jammed into a Raster.)

             JAM2           The FgPen will be used as foreground
                            pen while the background (when you
                            are writing text for example) will
                            be filled with the BgPen's colour.
                            (Two colours are jammed into a
                            Raster.)

             COMPLEMENT     Each pixel affected will be drawn
                            with the binary complement colour.
                            Where you write 1's the
                            corresponding bit in the Raster
                            will be reversed.
```

INVERSVID|JAM1	This mode is only use together with text. Only the background of the text will be drawn with the FgPen.
INVERSVID|JAM2	This mode is only use together with text. The background of the text will be drawn with the FgPen, and the characters itself with the BgPen.

SetDrPt()

This function will set the line pattern.

Synopsis:	SetDrPt(rast_port, line_pattern);
rast_port:	(struct RastPort *) Pointer to the RastPort that should be affected.
line_pattern:	(UWORD) The pattern. Each bit represents one dot. To generate solid lines you set the pattern value to 0xFFFF [hex] (1111111111111111 [bin]).

SetOPen()

This macro will change the AOlPen's colour. Note! This is not a function. It is actually a macro that is defined in the header file "gfxmacros.h". If you want to use this function you have to remember to include this file.

Synopsis:	SetOPen(rast_port, new_colour);
rast_port:	(struct RastPort *) Pointer to the RastPort that should be affected.
new_colour:	(long) A new colour value.

SetRast()

This function sets a whole Raster to a specific colour.

Synopsis:	SetRast(rast_port, colour);
rast_port:	(struct RastPort *) Pointer to the RastPort that should be affected.
colour:	(long) The colour reg. you want to fill the whole raster with.

SetRGB4()

This function allows you to change your screen's colours. Each colour may be picked out of a 4096 colour palette. (16 levels of red, 16 levels of green and 16 levels of blue; 16*16*16 = 4096.)

IMPORTANT! Before you may use this function you must have opened the Graphics Library. (All other functions are in the Intuition Library.) (See chapter 0 AMIGA for more information.)

Synopsis:	SetRGB4(viewport, register, red, green, blue);
viewport:	(struct ViewPort *) Pointer to a ViewPort which colour registers we are going to change. We can find the screen's ViewPort in the Screen structure. (If my_screen is a pointer to a Screen structure, this will get us a pointer to that screen's ViewPort: &my_screen->ViewPort)
register:	(long) The colour register you want to change. The screen's Depth decides how many colour registers the screen have:

Depth	Colour Registers
1	0 - 1
2	0 - 3
3	0 - 7
4	0 - 15
5	0 - 31
6	0 - 63

red:	Amount of red. (0 - 15)
green:	Amount of green. (0 - 15)
blue:	Amount of blue. (0 - 15)

Eg: SetRGB4(&my_screen->ViewPort, 2, 15, 15, 0); will change colour register 2 to be light yellow. (Red and green together will be yellow.)

ScrollRaster()

This function will scroll a rectangular area of a raster.

Synopsis:	ScrollRaster(rp, dx, dy, minx, miny, maxx, maxy);
rp:	(struct RastPort *) Pointer to the RastPort that should be affected.

```
  dx:         (long) Delta X movement. (A positive number moves
              the area to the right, a negative number to the
              left.)

  dy:         (long) Delta Y movement. (A positive number moves
              the area down, a negative number up.)

  minx:       (long) Left edge of the rectangle.

  miny:       (long) Top edge of the rectangle.

  maxx:       (long) Right edge of the rectangle.

  maxy:       (long) Bottom edge of the rectangle.


Text()

  This function prints text into a Raster.

  Synopsis:  Text( rast_port, string, nr_of_chr );

  rast_port: (struct RastPort *) Pointer to the RastPort that
             should be affected.

  string:    (char *) Pointer to a text string that will be
             printed.

  nr_of_chr: (long) The number of characters that should be
             printed.


WritePixel()

  This function will draw a single pixel.

  Synopsis:  WritePixel( rast_port, x, y );

  rast_port: (struct RastPort *) Pointer to the RastPort that
             should be affected.

  x:          (long) X position of the pixel.

  y:          (long) Y position of the pixel.
```

B.4 EXEC LIBRARY

AllocMem()

This function allocates memory. You specifies what type and how much you want, and it returns a pointer to the allocated memory, or NULL if there did not exist enough memory.

Synopsis: memory = AllocMem(size, type);

memory:	(void *) Pointer to the new allocated memory, or NULL if no memory could be allocated. Remember! Never use memory which you have not successfully allocated.
size:	(long) The size (in bytes) of the memory you want. (AllocMem() always allocates memory in multiples of eight bytes. So if you only ask for 9 bytes, Exec would actually give you 16 Bytes (2*8).)
type:	(long) You need to choose one of the three following types of memory (see chapter 0 INTRODUCTION for more information about Chip and Fast memory):

MEMF_CHIP	Chip memory. This memory can be accessed by both the main processor, as well as the Chips. Graphics/Sound data MUST therefore be placed in Chip memory. If it does not matter what type of memory you get (Fast or Chip), you should try to allocate Fast memory before you allocate Chip memory. (Chip memory is more valuable than Fast memory.)
MEMF_FAST	Fast memory. This memory can only be accessed by the main processor. (Graphics and Sound data can NOT be stored in Fast memory, use Chip memory.) This memory is normally a little bit faster than Chip memory, since only the main processor is working with it, and it is not disturbed by the Chips.
MEMF_PUBLIC	If it does not matter what type of memory you get (you do not intend to use the memory for Graphics/Sound data), you should use Fast memory. However, all Amigas do not have Fast memory, since you need to by a memory expansion in order to get it. If want to tell Exec that you would like to use Fast memory if there is any, else use Chip memory, you should ask for MEMF_PUBLIC.

```
            If you want the allocated memory to be cleared
            (initialized to zeros), you should set the flag
            MEMF_CLEAR.



FreeMem()

  This function deallocated previously allocated memory.
  Remember to deallocate all memory you have taken, and never
  deallocate memory which you have not taken.

  Synopsis: FreeMem( memory, size );

  memory    (void *) Pointer to some memory which has
            previously been allocated. Remember! never use
            memory which has been deallocated.

  size      (long) The size (in bytes) of the memory you want
            to deallocate.
```

B.5 AMIGA DOS LIBRARY

```
Close()

  This function closes an already opened file. Remember to
  close ALL files you have opened!

  Synopsis:    Close( file_handle );

  file_handle: (BPTR) Actually a pointer to a FileHandle
               structure which has been initialized by a
               previous Open() call.



CreateDir()

  This function creates a new directory, AND "locks" is
  automatically. (Remember to unlock the directory later on.)

  Synopsis: lock = CreateDir( name );

  lock:     (BPTR) Actually a pointer to a FileLock structure.
            If lock is equal to NULL, AmigaDOS have not been
            able to create the new directory.

  name:     (char *) Pointer to a string containing the name
            of the new directory.
```

CurrentDir()

This function makes a specified directory "current directory". You need to lock the new directory (new_lock) before you can make it the current directory. The function returns the old current directories lock so you can unlock it if necessary.

```
Synopsis: old_lock = CurrentDir( new_lock );

old_lock: (BPTR) Actually a pointer to a FileLock structure.
          It is the old current directory lock.

new_lock: (BPTR) Actually a pointer to a FileLock structure.
          The new current directory lock.
```

DeleteFile()

This function deletes a file or directory. Remember that a directory must be empty before it can be deleted.

```
Synopsis: ok = DeleteFile( name );

ok:       (long) Actually a Boolean. It is TRUE if AmigaDOS
          could delete the file/directory, else FALSE which
          means something went wrong. (Eg. disk write-
          protected, directory not empty etc.)

name:     (char *) Pointer to a string containing the name
          of the file/directory you want to delete.
```

Info()

This function returns information about a specified disk. You specify which disk by either lock that disk, or a file/ directory on that disk.

```
Synopsis:  ok = Info( lock, info_data );

ok:        (long) Actually a Boolean. It is TRUE if AmigaDOS
           could get information about the disk, else FALSE
           which means something went wrong.

lock:      (BPTR) Actually a pointer to a FileLock structure.

info_data: (struct InfoData *) Pointer to an InfoData
           structure which will be initialized by the Info()
           function. The problem with this structure is that
           it must be on a four byte boundary, so you need
           to use the function AllocMem() to get the right
           type of memory for the structure. (See Example.)
```

IoErr()

This function can be used to get more information about an error message. Whenever you have used an AmigaDOS function which did not work properly (you have received an error message), you call this function and it will return an explanation.

Synopsis: error = IoErr();

```
error:    (long) This field contains a flag returned by
          IoErr() which can be: (I do not think I need
          to explain what they mean.)

          ERROR_NO_FREE_STORE
          ERROR_TASK_TABLE_FULL
          ERROR_LINE_TOO_LONG
          ERROR_FILE_NOT_OBJECT
          ERROR_INVALID_RESIDENT_LIBRARY
          ERROR_NO_DEFAULT_DIR
          ERROR_OBJECT_IN_USE
          ERROR_OBJECT_EXISTS
          ERROR_DIR_NOT_FOUND
          ERROR_OBJECT_NOT_FOUND
          ERROR_BAD_STREAM_NAME
          ERROR_OBJECT_TOO_LARGE
          ERROR_ACTION_NOT_KNOWN
          ERROR_INVALID_COMPONENT_NAME
          ERROR_INVALID_LOCK
          ERROR_OBJECT_WRONG_TYPE
          ERROR_DISK_NOT_VALIDATED
          ERROR_DISK_WRITE_PROTECTED
          ERROR_RENAME_ACROSS_DEVICES
          ERROR_DIRECTORY_NOT_EMPTY
          ERROR_TOO_MANY_LEVELS
          ERROR_DEVICE_NOT_MOUNTED
          ERROR_SEEK_ERROR
          ERROR_COMMENT_TOO_BIG
          ERROR_DISK_FULL
          ERROR_DELETE_PROTECTED
          ERROR_WRITE_PROTECTED
          ERROR_READ_PROTECTED
          ERROR_NOT_A_DOS_DISK
          ERROR_NO_DISK
          ERROR_NO_MORE_ENTRIES
```

Open()

This function opens a file. Remember, before you can read/write files you have to open them.

Synopsis: file_handle = Open(file_name, mode);

file_handle: (BPTR) Actually a pointer to a FileHandle

```
                structure. If the system could not open the file
                with our requirements Open() returns NULL.

  file_name:    (char *) Pointer to a text string which contains
                the file name including any necessary devices/
                directories.

  mode:         (long) When you open a file you need to tell the
                system what you are going to do with it. This
                field should therefore contain one of the
                following flags:

                MODE_OLDFILE:   Opens an existing file for
                                reading and writing.

                MODE_NEWFILE:   Opens a new file for writing.
                                (If the file already exist it
                                is deleted.)

                MODE_READWRITE: Opens an old file with an
                                exclusive lock. (The file is
                                automatically locked with an
                                EXCLUSIVE_LOCK.)

                MODE_READONLY:  Same as MODE_OLDFILE.



Read()

  This function reads a specified number of bytes from a file.

  Synopsis:     bytes_read = Read( file_handle, buffer, size );

  bytes_read:   (long) Number of bytes actually read. Even if
                you tell AmigaDOS that you want to read x
                number of bytes, it is not certain that you
                actually can do it. The file is maybe corrupted,
                not as big as you thought etc.

  file_handle:  (BPTR) Actually a pointer to a FileHandle
                structure which has been initialized by a
                previous Open() call.

  buffer:       (char *) Pointer to the data buffer you want to
                read the data into.

  size:         (long) Number of bytes you want to read.



Rename()

  This function renames a file or directory. You can even
  move a file between directories by renaming it. (For example,
  Rename( "df0:Documents/Sale.doc", "df0:Letters/Sale.doc" );
  will move the file Sale.doc from the directory "Documents"
  to directory "Letters". Note! You can not rename a file from
  one volume to another.)

  Synopsis: ok = Rename( old_name, new_name );

  ok:       (long) Actually a Boolean. It is TRUE if AmigaDOS
            could rename the file/directory, else FALSE which
            means something went wrong. (Eg. disk write
            -protected.)

  old_name: (char *) Pointer to a string containing the old
            file/directory name.

  new_name: (char *) Pointer to a string containing the new
            file/directory name.



Seek()

  This function moves the "file cursor" inside a file:

  Synopsis:    old_pos = Seek( file_handle, new_pos, mode );

  old_pos:     (long) Previous position in the file, or -1 if
               an error occurred.

  file_handle: (BPTR) Actually a pointer to a FileHandle
               structure which has been initialized by a
               previous Open() call.

  new_pos:     (long) New position relative to the "mode".

  mode:        (long) The new_pos can be relative to:
                 OFFSET_BEGINNING: Beginning of the file.
                 OFFSET_CURRENT:   Current position.
                 OFFSET_END:       The end of the file.



SetComment

  This function attach a comment to a file or directory.

  Synopsis: ok = SetComment( name, comment );

  ok:       (long) Actually a Boolean. It is TRUE if AmigaDOS
            could attach the new comment, else FALSE which
            means something went wrong. (Eg. disk write-
            protected.)

  name:     (char *) Pointer to a string containing the name
            of the file/directory you want to attach the
            comment to.

  comment:  (char *) Pointer to a string containing the
            comment. (A comment may be up to 80 characters
          long.)
```

SetProtection()

```
  This function alters the protection bits of a file.
  You can set following flags:

  FIBF_DELETE  : the file/directory can not be deleted.
  FIBF_EXECUTE : the file can not be executed.
  FIBF_WRITE   : you can not write to the file.
  FIBF_READ    : you can not read the file.
  FIBF_ARCHIVE : Archive bit.
  FIBF_PURE    : Pure bit.
  FIBF_SCRIPT  : Script bit.

  (Note! All of the flags are for the moment not working!)

  Synopsis: ok = SetProtection( name, mask );

  ok:       (long) Actually a Boolean. It is TRUE if AmigaDOS
            could alter the protection bits, else FALSE which
            means something went wrong. (Eg. disk write-
            protected.)

  name:     (char *) Pointer to a string containing the name
            of the file/directory you want to change the
            protection bits.

  mask:     (long) The protection bits. (For example, if you
            want to make the file/directory not deletable,
            and that it can not be executed you should set the
            protection bits: FIBF_DELETE | FIBF_EXECUTE.)
```

UnLock()

```
  This function unlocks a previously locked file: (Remember to
  unlock ALL files you have locked!)

  Synopsis: Unlock( lock );

  lock:     (BPTR) Actually a pointer to FileLock structure
            which has been initialized by a previous Lock()
            call.
```

Write()

```
  This function writes a specified number of bytes to a file.

  Synopsis:  bytes_wr = Write( file_handle, buffer, size );

  bytes_wr:    (long) Number of bytes actually written. Even if
```

```
              you tell AmigaDOS that you want to write x number
              of bytes, it is not certain that you actually
              can do it. Maybe the disk was full, write-
              protected etc.

file_handle: (BPTR) Actually a pointer to a FileHandle
              structure which has been initialized by a
              previous Open() call.

buffer:       (char *) Pointer to the data buffer which you
              want to write.

size:         (long) Number of bytes you want to write.
```

C SYSTEM DEFAULT CONSOLE KEY MAPPING

Raw Key Code	Unshifted Default Value	Shifted Default Value
00	`	~
01	1	!
02	2	@
03	3	#
04	4	$
05	5	%
06	6	^
07	7	&
08	8	*
09	9	(
0A	0	)
0B	-	_
0C	=	+
0D	\	|
0E	(undefined)	(undefined)
0F	0	0 (numeric pad)
10	q	Q
11	w	W
12	e	E
13	r	R
14	t	T
15	y	Y
16	u	U
17	i	I
18	o	O
19	p	P
1A	[	{
1B	]	}
1C	(undefined)	(undefined)
1D	1	1 (numeric pad)
1E	2	2 (numeric pad)
1F	3	3 (numeric pad)
20	a	A
21	s	S
22	d	D
23	f	F
24	g	G
25	h	H
26	j	J
27	k	K
28	l	L
29	;	:
2A	'	"
2B	(reserved)	(reserved)
2C	(undefined)	(undefined)
2D	4	4 (numeric pad)
2E	5	5 (numeric pad)
2F	6	6 (numeric pad)

30	(reserved)	(reserved)
31	z	Z
32	x	X
33	c	C
34	v	V
35	b	B
36	n	N
37	m	M
38	,	<
39	.	>
3A	/	?
3B	(undefined)	(undefined)
3C	.	. (numeric pad)
3D	7	7 (numeric pad)
3E	8	8 (numeric pad)
3F	9	9 (numeric pad)
40	SPACE (20)	SPACE (20)
41	BACK SPACE (08)	BACK SPACE (08)
42	TAB (09)	TAB (09)
43	ENTER (0D) (numeric pad)	ENTER (0D) (numeric pad)
44	RETURN (0D)	RETURN (0D)
45	ESC (1B)	ESC (1B)
46	DEL (7F)	DEL (7F)
47	(undefined)	(undefined)
48	(undefined)	(undefined)
49	(undefined)	(undefined)
4A	-	- (numeric pad)
4B	(undefined)	(undefined)
4C	UP ARROW <CSI>A	UP ARROW <CSI>T
4D	DOWN ARROW <CSI>B	DOWN ARROW <CSI>S
4E	RIGHT ARROW <CSI>C	RIGHT ARROW <CSI> A
4F	LEFT ARROW <CSI>D	LEFT ARROW <CSI> @
50	F1 <CSI>0~	F1 <CSI>10~
51	F2 <CSI>1~	F2 <CSI>11~
52	F3 <CSI>2~	F3 <CSI>12~
53	F4 <CSI>3~	F4 <CSI>13~
54	F5 <CSI>4~	F5 <CSI>14~
55	F6 <CSI>5~	F6 <CSI>15~
56	F7 <CSI>6~	F7 <CSI>16~
57	F8 <CSI>7~	F8 <CSI>17~
58	F9 <CSI>8~	F9 <CSI>18~
59	F10 <CSI>9~	F10 <CSI>19~
5A	(undefined)	(undefined)
5B	(undefined)	(undefined)
5C	(undefined)	(undefined)
5D	(undefined)	(undefined)
5E	(undefined)	(undefined)
5F	HELP <CSI>?~	HELP <CSI>?~
60	LEFT SHIFT	LEFT SHIFT
61	RIGHT SHIFT	RIGHT SHIFT
62	CAPS LOCK	CAPS LOCK

63	CTRL	CTRL
64	LEFT ALT	LEFT ALT
65	RIGHT ALT	RIGHT ALT
66	LEFT AMIGA	CLOSE AMIGA
67	RIGHT AMIGA	OPEN AMIGA
68	LEFT MOUSE BUTTON	LEFT MOUSE BUTTON
69	RIGHT MOUSE BUTTON	RIGHT MOUSE BUTTON
6A	MIDDLE MOUSE BUTTON	MIDDLE MOUSE BUTTON
6B	(undefined)	(undefined)
6C	(undefined)	(undefined)
6D	(undefined)	(undefined)
6E	(undefined)	(undefined)
6F	(undefined)	(undefined)
70 - 7F	(undefined)	(undefined)
80-F8	UP TRANSITION (80 for 00, 81 for 01 ... F8 for 7F)	UP TRANSITION
F9	LAST KEYKODE BAD	
FA	KEYBOARD BUFFER OVERFLOW	
FB	(undefined)	(undefined)
FC	KEYBOARD SELFTEST FAILED	
FD	POWER-UP KEY STREAM START	
FE	POWER-UP KEY STREAM END	
FF	MOUSE EVENT (mouse moved only, no button changed)	

D ASCII CODES

Decimal	Hexadecimal	ASCII	DESCRIPTION
0	00	NUL	CTRL/1
1	01	SOH	CTRL/A
2	02	STX	CTRL/B
3	03	ETX	CTRL/C
4	04	EOT	CTRL/D
5	05	ENQ	CTRL/E
6	06	ACK	CTRL/F
7	07	BEL	CTRL/G
8	08	BS	CTRL/H, BACKSPACE
9	09	HT	CTRL/I, TAB
10	0A	LF	CTRL/J, ENTER
11	0B	VT	CTRL/K
12	0C	FF	CTRL/L
13	0D	CR	CTRL/M, RETURN
14	0E	SO	CTRL/N
15	0F	SI	CTRL/O
16	10	DLE	CTRL/P
17	11	DC1	CTRL/Q
18	12	DC2	CTRL/R
19	13	DC3	CTRL/S
20	14	DC4	CTRL/T
21	15	NAK	CTRL/U
22	16	SYN	CTRL/V
23	17	ETB	CTRL/W
24	18	CAN	CTRL/X
25	19	EM	CTRL/Y
26	1A	SUB	CTRL/Z
27	1B	ESC	ESC, ESCAPE
28	1C	FS	CTRL<
29	1D	GS	CTRL/
30	1E	RS	CTRL/=
31	1F	US	CTRL/-
32	20	SP	SPACE
33	21	!	!
34	22	"	"
35	23	#	#
36	24	$	$
37	25	%	%
38	26	&	&
39	27	'	'
40	28	(	(
41	29	)	)
42	2A	*	*
43	2B	+	+
44	2C	,	,
45	2D	-	-
46	2E	.	.
47	2F	/	/
48	30	0	0

49	31	1	1
50	32	2	2
51	33	3	3
52	34	4	4
53	35	5	5
54	36	6	6
55	37	7	7
56	38	8	8
57	39	9	9
58	3A	:	:
59	3B	;	;
60	3C	<	<
61	3D	=	=
62	3E	>	>
63	3F	?	?
64	40	@	@
65	41	A	A
66	42	B	B
67	43	C	C
68	44	D	D
69	45	E	E
70	46	F	F
71	47	G	G
72	48	H	H
73	49	I	I
74	4A	J	J
75	4B	K	K
76	4C	L	L
77	4D	M	M
78	4E	N	N
79	4F	O	O
80	50	P	P
81	51	Q	Q
82	52	R	R
83	53	S	S
84	54	T	T
85	55	U	U
86	56	V	V
87	57	W	W
88	58	X	X
89	59	Y	Y
90	5A	Z	Z
91	5B	[	[
92	5C	\	\
93	5D	]	]
94	5E	^	^
95	5F	_	_
96	60	`	`
97	61	a	a
98	62	b	b
99	63	c	c
100	64	d	d
101	65	e	e
102	66	f	f
103	67	g	g
104	68	h	h
105	69	i	i

106	6A	j	j
107	6B	k	k
108	6C	l	l
109	6D	m	m
110	6E	n	n
111	6F	o	o
112	70	p	p
113	71	q	q
114	72	r	r
115	73	s	s
116	74	t	t
117	75	u	u
118	76	v	v
119	77	w	w
120	78	x	x
121	79	y	y
122	7A	z	z
123	7B	{	{
124	7C	|	|
125	7D	}	}
126	7E	~	~
127	7F	DEL	DELETE

E DATA TYPES

E.1 LATTICE C DATA TYPES

TYPE	BITS	Minimum value	Maximum value
char	8	-128	127
unsigned char	8	0	255
short	16	-32 768	32 767
unsigned short	16	0	65 535
int	32	-2 147 483 648	2 147 483 647
unsigned int	32	0	4 294 987 295
long	32	-2 147 483 648	2 147 483 647
unsigned long	32	0	4 294 987 295
float	32	±10E-37	±10E+38
double	64	±10E-307	±10E+308

E.2 AMIGA DATA TYPES

Amiga Data Types	Lattice C Data Types	Description
LONG	long	Signed 32-bit
ULONG	unsigned long	Unsigned 32-bit
LONGBITS	unsigned long	32 bits manipulation
WORD	short	Signed 16-bit
UWORD	unsigned short	Unsigned 16-bit
WORDBITS	unsigned short	16 bits manipulation
BYTE	char	Signed 8-bit
UBYTE	unsigned char	Unsigned 8-bit
BYTEBITS	unsigned char	8 bits manipulation
VOID	void	Nothing
STRPTR	*unsigned char	String pointer
CPTR	ULONG	Absolute memory pointer
TEXT	unsigned char	Text
BOOL	short	Boolean (The file has also defined the two words TRUE = 1 and FALSE = 0)

Here is a list of some data types which should not be used any more:

APTR	*STRPTR	Absolute memory pointer (Misdefined, use CPTR!)
SHORT	short	Signed 16-bit (WORD)
USHORT	unsigned short	Unsigned 16-bit (UWORD)

F GURU-MEDITATION

F.1 INTRODUCTION

If you have been programming the Amiga for a while you have most certainly seen your nice Amiga blow up in front of your eyes. However, the Amiga is a nice computer, and if it crashes it will try to do that as neatly as possible. You are usually (!) allowed to save any important files, and when the Amiga goes down it will give you a last message, trying to tell you what went wrong.

Many programmers have not realized how important that last message is. For them, it is just a collection of strange numbers. But those numbers can actually tell you what exactly went wrong, and once you know what went wrong, it is usually no problem to find the bug.

F.2 AMIGA CRASHING

When the Amiga is crashing it happens that the Exec have realized that something will go wrong and halted that task. Exec will then open a System requester with a warning message:

```
  --------------------------------
  |  Software error - task held  |
  |   Finish all disk activity   |
  | Select CANCEL to reset/debug |
  |                              |
  | ---------         ---------- |
  | | Retry |         | Cancel | |
  | ---------         ---------- |
  --------------------------------
```

You can then save any important files to a disk before you answer the requester (once you have pressed CANCEL the Guru will visit you!). Important, since the Amiga is in trouble it can crash any second, and if you were saving anything onto a disk at that moment, the disk may become corrupted, and all data lost. The best solution is to have an empty emergency disk that you only use when the Amiga is upset. Sometimes the Exec have not been able to halt the task, and the Guru will immediately visit you.

F.3 GURU ALERTS

Once the Amiga really crashes, the powerlight will flash for some seconds, and an Alert will be activated. For example:

```
  ----------------------------------------------------------
  | Software failure! Press left mouse button to continue. |
  |         Guru Meditation   #84010007.00C13870           |
  ----------------------------------------------------------
```

Now you only need to decode the message and you have found the error. (Advanced programmer can attach a modem to the Amiga and use a program called ROMWack in order to debug the computer. But we will not discuss it here.)

F.4 GURU MEDITATION NUMBERS

There exist two different types of Guru Meditation Numbers:
- CPU Errors (680x0 Processor Traps)
- System Software Errors

F.5 CPU ERRORS

CPU Errors look like this:

```
Guru Meditation  #0000000x.yyyyyyyy
```

x is one of the following values:

2	Bus Error	Hardware error
3	Address Error	Word access on odd byte boundary
4	Illegal Instruction	
5	Divide by zero	
6	CHK Instruction	
7	TRAPV Instruction	
8	Privilege Violation	
9	Trace	
A	Opcode 1010 Emulation	Instruction word with a value between A000-AFFF.
B	Opcode 1111 Emulation	Instruction word with a value between F000-FFFF.

yyyyyyyy is the address of the task which went wrong. (It is normally your own program that caused the problem.)

If you get a Guru Meditation number like #00000005.00C13870 it means: A program, at the address C13870, tried to devide a value by zero. Since you now know what the error was you only need to look at the places where your program can be forced to devide a value by zero, and it should not take to long time to find the bug.

F.6 SYSTEM SOFTWARE ERRORS

System Software errors look like this:

```
Guru Meditation  #aabbcccc.dddddddd
```

The first field of the number tells us if the error is a Recoverable Error or if it is a Dead End Alert. If the error is a Dead End Alert the number will start with 8 otherwise it is 0. (The total screen will also be black, while on Recoverable Errors the screen is merely pushed down a bit.)

The first field of the number tells us also which Device, Library or Resource went wrong:

01	Exec Library	LIBRARIES
02	Graphics Library	
03	Layers Library	
04	Intuition Library	
05	Math Library	
06	CList Library	
07	AmigaDOS Library	
08	RAM Handler Library	
09	Icons Library	
10	Audio Device	DEVICES
11	Console Device	
12	GamePort Device	
13	Keyboard Device	
14	Trackdisk Device	
15	Timer Device	
20	CIA Resource	RESOURCES
21	Disk Resource	
22	Misc Resource	
30	BootStrap	OTHERS
31	Workbench	
32	Disk Copy	

A number like 04 means: Recoverable Error in the Intuition Library. While a number like 84 means: Fatal Error in the Intuition Library.

The second field (bb) of the Guru Meditation number gives us the general cause of the problem:

01	No Memory
02	Unable to Create Library
03	Unable to Open Library
04	Unable to Open Device
05	Unable to Open Resource
06	Input/Output (I/O) Error

07 No Signal

So a number like 8201cccc means a fatal error in the Graphics Library, the problem was caused by not enough memory.

The last field (cccc) before the dot gives some more specific information. Here is a list of some common Guru Meditation Numbers: (This information is taken form the headerfile "exec/alerts.h" [V1.3]:)

Exec Library:

01000000	
81000001	68000 exception vector checksum
81000002	Execbase checksum
81000003	Library checksum failure
81000004	No memory to make library
81000005	Corrupted memory list
81000006	No memory for interrupt servers
81000007	InitStruct() of an APTR source
81000008	A semaphore is in illegal state
81000009	Freeing memory already freed
8100000A	Illegal 68k exception taken

Graphics Library:

02000000	
82010000	Graphics out of memory
82010006	Long frame, no memory
82010007	Short frame, no memory
02010009	Text, no memory for TmpRas
8201000A	BltBitMap, no memory
8201000B	Regions, memory not available
82010030	MakeVPort, no memory
82011234	Emergency memory not available *

Layers Library:

03000000	
83010000	Layers out of memory

Intuition Library:

04000000	
84000001	Unknown gadet type
04000001	Recovery form of AN_GadgetType
84010002	Create port, no memory
04010003	Item plane alloc, no memory
04010004	Sub alloc, no memory
84010005	Plane alloc, no memory
84000006	Item box top < RelZero
84010007	Open screen, no memory
84010008	Open screen, raster alloc, no memory
84000009	Open sys screen, unknown type
8401000A	Add SW gadgets, no memory
8401000B	Open window, no memory
8400000C	Bad State Return entering Intuition

```
  8400000D  Bad Message received by IDCMP
  8400000E  Weird echo causing incomprehension
  8400000F  Couldn't open the Console Device


Amiga DOS Library:
  07000000
  07010001  No memory at startup
  07000002  EndTask didn't
  07000003  Qpkt failure
  07000004  Unexpected packet received
  07000005  Freevec failed
  07000006  Disk block sequence error
  07000007  Bitmap corrupt
  07000008  Key already free
  07000009  Invalid checksum
  0700000A  Disk Error
  0700000B  Key out of range
  0700000C  Bad overlay


RAM Library:
  08000000
  08000001  No overlays in library seglists


Trackdisk Device:
  14000000
  14000001  Calibrate: seek error
  14000002  Delay: error on timer wait


Timer Device:
  15000000
  15000001  Bad request
  15000002  Power supply does not supply ticks


Disk Resourcek.resource:
  21000000
  21000001  Get unit: already has disk
  21000002  Interrupt: no active unit


BootStrap:
  30000000
  30000001  Boot code returned an error
```

The number after the dot (dddddd dd) can be three things:

1. Address of the task which went wrong.
2. If the error occured because of some sort of memory allocation/deallocation, it is the address of that memory block.
3. If Exec is realy confused the number is 48454C50, which stands for HELP. (48=H, 45=E, 4C=L, 50=P)

G OPERATORS

PRECEDENCE AND ASSOCIATIVITY OF OPERATORS

Operators	Type	Associativity
() [] . ->	groups membership	left to right
- ~ ! * &	unary	right to left
++ -- sizeof casts	unary	right to left
* / %	multiplicative	left to right
+ -	additive	left to right
<< >>	shift	left to right
< > <= >=	relational	left to right
== !=	equality	left to right
&	bitwise AND	left to right
^	bitwise EXCL OR	left to right
|	bitwise INCL OR	left to right
&&	logical AND	left to right
||	logical OR	left to right
?:	conditional	right to left
= *= /= %= += -= <<= >>= &= ^= |=	assignment	right to left
,	comma	left to right

H ADDITIONAL EXAMPLE SOURCES

H.1 INTRODUCTION

The following additional source code examples can be found in the original ACM package, incl. referenced data files.

http://aminet.net/package/dev/c/ACM

H.2 COLOURWINDOW

ColourWindow is the first and only true colour requester in the Public Domain. It adjust itself to any depth (2, 4, 8, 16 or 32 colours), and can be used with high- as well as low resolution screens. Everything is done by the rules, and this program will return everything it has taken, and use a minimum of processing time. It is yet another program releaced from the Amiga C Club!

H.3 EASYSOUND

Now at last you can easily write C programs that plays digitized sound. You simply use four functions that will take care of all the work of allocating memory, loading the files, opening the ports and reserving the sound channels. Despite the simplicity you can still decide what volume and rate, which channel, and how many times the sound should be played. The functions contain full error checking, and will close and return everything that have been taken.

H.4 FILEWINDOW

The reason why I created FILE WINDOW was that I never more wanted to see that old stupid "Please type in filename:" prompt. It is a disgrace for any program to use it, and VERY annoying for the user.

REMEMBER, one of the first things the user will see of your program is the file requester. If you want your program to make a good impression, and look solid, I would recommend you to use a good file requester.

FILE WINDOW is written to be as easy as possible to use, and is fully amigaized.

H.5 INPUT

H.5.1 JOYSTICK

Joystick() is a handy, easy and fast but naughty function that hits the hardware of the Amiga. It looks at either port 1 or port 2, and returns a bitfield containing the position of the stick and the present state of the button.

H.5.2 KEYBOARD

Keyboard() is a handy, easy and fast but naughty function that hits the hardware of the Amiga. It checks the keyboard, and returns the Raw Key Code. (See chapter (C) SYSTEM DEFAULT CONSOLE KEY MAPPING for the full list of Raw Key Codes.)

H.5.3 MOUSE

Mouse() is a handy, easy and fast but naughty function that hits the hardware of the Amiga. It looks at either port 1 or port 2, and returns the (x and y) delta movement of the mouse, as well as a bitfield containing the present state of the three buttons. (A normal Amiga mouse has only two buttons (left and right), but it is possible to connect a mouse with three buttons, so why shouldn't we support it?)

H.6 HACKS

H.6.1 COPPER

This program demonstrates how to play with the Copper.

H.6.2 LED

This fantastic useful program does what all true hackers have dreamt of. Enjoy your Amiga's fantastic ability to flash one LED!

H.7 TOOLS

H.7.1 P2C

This program prints out the Sprite Data of the pointer. It can also print out the colours, and/or a SimpleSprite structure.