OCINF - The Occam-C Interface Generation Tool

(A Tutorial)

C. S. Lewis

University of KENT, Canterbury.

ABSTRACT

Ocinf is a tool which simplifies the task calling C functions from occam programs.

Ocinf takes care of the low level interfacing for you by defining external occam procedures headers for each interfaced C routine.

Some of the goals ocinf will enable occam programs to achieve:

(1) Issue Unix shell commands.

(2) Call user-written C routines.

(3) Call system library routines (Xlib).

This tutorial will explain how ocinf can be used to achieve these goals.

16 April, 1996

OCINF - The Occam-C Interface Generation Tool

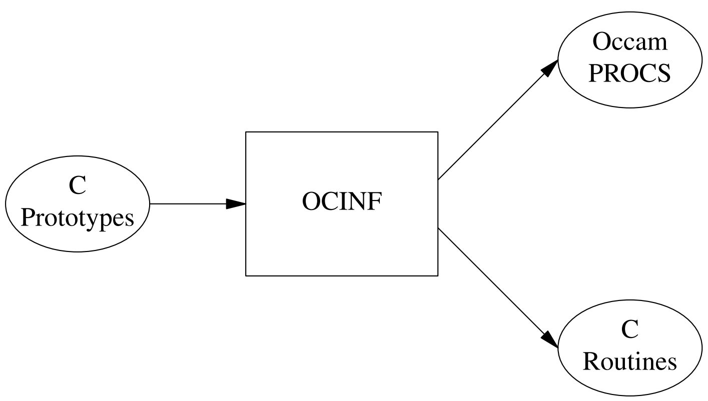
(A Tutorial)

C. S. Lewis

University of KENT, Canterbury.

1. INTRODUCTION

The ocinf tool creates occam interfaces to C routines, the following diagram shows the stages in interface generation.



1.1. Ocinf Input

The input to the interface tool is a standard C header file, this header file may contain:

(1) Function prototypes.

(2) Type declarations.

(3) C Pre-processor macros.

1.2. Ocinf Output

The Ocinf tool generates two output files:

(1) Occam procedure prototypes for the interfaced C routines.

(2) C routines which handle occam-C parameter passing.

1.3. Using the Interfaces

Once the required interfaces have been generated, occam programs can then call the required interfaced C routines by including the generated occam prototypes *(interface file 1)* and then calling the named routines.

Interfaced routines may have parameter data types that require conversion between occam and C formats. Some support routines are provided for this *([]BYTE <-> char[]),* other support routines will be released in the near future.

1.4. Compiling and Linking

The top level occam program is compiled with kroc to produce an object file, which is then linked with the required generated C files *(interface file 2)* and possibly linked with the required system libraries (-lxlib).

The included examples show this process.

1.5. Conventions

Ocinf uses the following conventions for file and routine names.

1.5.1. Generated File Names

The file names generated by ocinf from the input C header file 'example1.h' are 'example1_h_if.oif' and 'example1_h_if.c'.

1.5.2. Generated Routine Names

The occam interface routines have **C.** prefixed to the C prototype name. And '_' in the original C routine names are replaced by '.' in the occam routine name.

For example, the C routine name **rtn_1** will appear in the occam interface as **C.rtn.1**

occam routines which begin with **C.** are handled specially by the kroc system, users are asked to avoid writing occam procedures beginning with this prefix.

1.6. Type Mappings

The ocinf tool maps C parameter types into equivalent occam parameter types, the example below shows this mapping:

1.6.1. The C header file

```
typedef struct
  {
     int a;
     char b;
  } Record;

typedef Record *RR;

void example_mappings
(

    int i,
    int *ip,
```

```
    short int si,
    short int *sip,

    long long int li,
    long long int *lip,

    float f,
    float *fp,


    double d,
    double *dp,


    char c,
    char *cp,
    char **cpp,

    Record *rp,
    RR     *rpp
);
```

1.6.2. The generated occam interface file

```
-- Predefines for C typedef sizes
VAL INT C.Record       IS 8:
VAL INT C.RR           IS 4:


#PRAGMA EXTERNAL "PROC C.example.mappings (VAL INT i, INT ip,           *
                    *VAL INT16 si, INT16 sip, VAL INT64 li,         *
                    *INT64 lip, VAL REAL32 f, REAL32 fp,            *
                    *VAL REAL64 d, REAL64 dp, VAL []BYTE c,         *
                    *[]BYTE cp, INT32 cpp, INT32 rp, INT32 rpp)  = 0 "
```

1.7. Interface Creation

Ocinf should be applied to the input header file to generate the required interfaces. *See the following examples.*

2. EXAMPLES

We present three examples which show how ocinf is used to create interfaces, and how these interfaces are used by occam programs.

2.1. EXAMPLE 1

This example shows how commands can be issued to the Unix shell from occam. The C routine *system()* is used to issue shell commands.

2.1.1. Building Example1

The Makefile in this directory contains commands for building example1. Type the command "make example1", this will create the required interface files and compile and link the example1.occ program.

When you have created the executable example1, run it.

2.1.2. The Occam program

Below is the occam program which calls the C system routine.

```
-- include the Occam interface for {system}
#INCLUDE "example1_h_if.oif"
#INCLUDE "cconv.oif"

--
-- Example1:
--
--    Demonstrates calling the UNIX 'system' command from occam,
--    by issuing the shell command line "pwd ; ls".
--
--
PROC  example1 (CHAN OF BYTE stdin, stdout, stderr)

  -- System command to be executed.
  VAL []BYTE system.command IS "pwd ; ls":

  -- Declare space for C string
  [100]BYTE  c.system.command:


  PROC  out.str (VAL []BYTE s)
    VAL length IS SIZE s:
    SEQ
      SEQ i = 0 FOR length
        SEQ
          stdout ! s [i]
  :


  --
  --  ============
  --  MAIN PROGRAM
  --  ============
  --

  SEQ

    -- Produce Description of Example.
    --
```

```
    out.str("*c*n")
    out.str("*c*n")
    out.str("OCINF: example1")
    out.str("*c*n")
    out.str("Demonstrates calling the Unix SYSTEM command from occam,*c*n")
    out.str("by issuing the shell command line PWD ; LS.*c*n")
    out.str("*c*n*c*n")


    -- Initialise C string from occam string.
    --
    bytes.to.chars(system.command, c.system.command)


    --  Call UNIX system command
    --
    C.system(c.system.command)
:
```

There are two #INCLUDE lines, the first is the occam interface to the C routine system(), and the second is the occam interface to conversion routines.

Change "pwd ; ls" to "clear ; pwd ; ls" in example1.occ, then remake and run example1.

2.2. EXAMPLE 2

This example shows how user written C routines can be called from occam.

2.2.1. Building Example2

The Makefile in this directory contains commands for building example2. Type the command "make example2", this will create the required interface files and compile and link the example2.occ program.

When you have created the executable example2, run it.

2.2.2. The Occam program

Below is the occam program which calls the user written C routines.

```
-- include the Occam interface for {user written routines}
#INCLUDE "example2_h_if.oif"
#INCLUDE "cconv.oif"

--
-- Example2:
--
--    Demonstrates calling user written C routines from occam.
--
--
--
PROC  example2 (CHAN OF BYTE stdin, stdout, stderr)
```

```
INT     a,b,c:
INT     level:


PROC  out.str (VAL []BYTE s)
  VAL length IS SIZE s:
  SEQ
    SEQ i = 0 FOR length
      SEQ
        stdout ! s [i]
:

PROC  CHECK.INT (INT actual, VAL INT expected, VAL []BYTE message)
  SEQ
    IF
      (actual = expected)
        SEQ
          out.str("OK: ")
          out.str(message)
          out.str("*c*n")
      (TRUE)
        SEQ
          out.str("ERROR: ")
          out.str(message)
          out.str("*c*n")
:


--
--  ============
--  MAIN PROGRAM
--  ============
--

SEQ

  -- Produce Description of Example.
  --
  out.str("*c*n")
  out.str("*c*n")
  out.str("OCINF: example2")
  out.str("*c*n")
  out.str("Demonstrates calling user written C routines from occam.*c*n")
  out.str("*c*n*c*n")


  -- Initialise variables
  --
  a := 1
  b := 2
  c := 0
  level := -1
```

```
  -- Call and check called C routines
  --
  C.stack.level(level)
  CHECK.INT(level,0,"Initial stack level")

  C.stack.push.int(a)
  C.stack.push.int(b)
  C.stack.pop.int(c)
  CHECK.INT(c,2,"popped value")

  C.stack.level(level)
  CHECK.INT(level,1,"Final stack level")

:
```

2.2.3. The original C header file

```
/* Example2:  Interface to user written C routines */

/* specification */

void  stack_push_int(int);
int   stack_pop_int(void);
int   stack_level(void);
```

2.2.4. The generated occam interfaces

```
#PRAGMA EXTERNAL "PROC C.stack.push.int (VAL INT P1)  = 0 "
#PRAGMA EXTERNAL "PROC C.stack.pop.int (INT result)  = 0 "
#PRAGMA EXTERNAL "PROC C.stack.level (INT result)  = 0 "
```

2.3. EXAMPLE 3

This example shows how the Xlib library routines can be called from occam.

The program below opens a window, then displays some text and draws a dotted rectangle within the opened window. The program then enters an event loop which is terminated by the user pressing a mouse button or key while pointing inside the window.

If the window is resized, then the text and rectangle will be redrawn and placed within the new window dimensions.

2.3.1. Building Example3

The interface files are supplied, to build this example just type "make xlib_test" then run.

2.3.2. Notes

This occam program is a direct line-by-line translation of a demonstration C program. The original C program used macros and accessed structure element directly, this functionality has been achieved in occam by interfacing to specially written routines.

2.3.2.1. Structures and Macros

In the occam program, macro and structure functionality has been provided by writting C interface routines (xlibmacros_if.oif, and xlibmacros_if.c) these routines are named C.MACRO.etc and C.STRUCTURE.etc.

2.3.2.2. Allocating Structure Space

Xlib expects space for some strutures to be allocated by the calling program, in this case the occam program declares enough array space for the required structure. The structure sizes are included in the occam interface file.

To get the parameter passing mechanism correct, occam needs to pass the address of some structures as and INT32, this is achieved by declaring an INT32 and setting its value to the the address of the required structure using transputer ASM.

2.3.3. The Occam Program

```
--
-- KROC Occam/C Interface Tool X Windows test program
-- $Source: /proj/kroc/develop/examples/ocinf/xlib/RCS/xlib_test.occ,v $
--
-- $Id: xlib_test.occ,v 1.2 1995/09/18 13:30:36 djb1 Exp $
--
-- (C) Copyright 1995 C.S.Lewis <csl2@mint.ukc.ac.uk>
-- University of Kent at Canterbury
--
--

-- include the Occam interface for {Xlib, Xutil, Xos, Xatom, X} :
#INCLUDE "xlib_h_if.oif"
#INCLUDE "xlibmacros_if.oif"

#INCLUDE "cconv.oif"

PROC  main (CHAN OF BYTE stdin, stdout, stderr)

  -- Define interface data types
  DATA TYPE Display       IS INT32:
  DATA TYPE Window        IS INT32:
  DATA TYPE Pixmap        IS INT32:
  DATA TYPE XSizeHints    IS INT32:
  DATA TYPE XIconSize     IS INT32:
  DATA TYPE XEvent        IS INT32:
  DATA TYPE GC            IS INT32:
  DATA TYPE XFontStruct   IS INT32:
  DATA TYPE XWMHints      IS INT32:
  DATA TYPE XClassHint    IS INT32:
  DATA TYPE XTextProperty IS INT32:

  -- Declare Occam interface variables
  INT32      display:                          -- Display
  INT        screen.num:                       -- int

  BOOL       Alive:
  INT        report.type, report.xexpose.count:
```

```
INT       shmw, shmh:


VAL INT  BITMAPDEPTH IS 1:
VAL INT  TOO.SMALL   IS 0:
VAL INT  BIG.ENOUGH  IS 1:
VAL INT icon.bitmap.width  IS 16:
VAL INT icon.bitmap.height IS 16:
VAL INT border.width IS 4:
VAL []BYTE progname IS "XLIB.test":                -- char *
VAL []BYTE window.name IS "Basic Window Program":  -- char *
VAL []BYTE icon.name IS "basicwin":                -- char *
VAL []BYTE argc IS " ":                            -- char *
VAL []BYTE font.name IS "9x15":                    -- char *


INT32  win:                                        -- Window
INT width, height, x, y:                           -- int
INT display.width, display.height, depth:          -- unsigned int
INT icon.width, icon.height:                       -- unsigned int

INT32  icon.pixmap:                                -- Pixmap

INT32  size.list:                                  -- XIconSize *
INT    count:                                      -- int
INT32  count32:
INT32  gc:                                         -- GC
INT32  font.info:                                  -- XFontStruct *
INT    window.size:                                -- int
INT    result:                                     -- int
INT    temp.i:
INT32  temp.i32:
BOOL   temp.b:


INT32  size.hints:                          -- XSizeHints       !!! Local space !!!
[C.XSizeHints]INT Filler1:

INT32  report:                              -- XEvent           !!! Local space !!!
[C.XEvent]INT Filler2:

INT32  wm.hints:                            -- XWMHints         !!! Local space !!!
[C.XWMHints]INT Filler3:

INT32  class.hints:                         -- XClassHint       !!! Local space !!!
[C.XClassHint]INT Filler4:

INT32  windowName:                          -- XTextProperty    !!! Local space !!!
[C.XTextProperty]INT Filler5:

INT32  iconName:                            -- XTextProperty    !!! Local space !!!
[C.XTextProperty]INT Filler6:
```

```
-- INT32 pointers to the 'C' Strings.
INT32 ptr.TO.c.window.name, ptr.TO.c.progname, ptr.TO.c.window.name:
INT32 ptr.TO.c.argv, ptr.TO.c.icon.name, ptr.TO.c.font.name:



[32]BYTE icon.bitmap.bits:

-- Declare C interface variables
[100]BYTE  c.progname:              -- char *
[100]BYTE  c.argc:                  -- char *
[100]BYTE  c.progname:              -- char *
[100]BYTE  c.window.name:           -- char *
[100]BYTE  c.icon.name:             -- char *
[100]BYTE  c.font.name:             -- char *
[100]BYTE  c.display.name:          -- char *
INT32      root.window:             -- ???
INT32      black.pixel, white.pixel:   -- unsigned long int
INT        c.argv:                  -- int

--  Xlib size hints
VAL INT USPosition   IS (1 << 0): -- user specified x, y
VAL INT USSize       IS (1 << 1): -- user specified width, height
VAL INT PPosition    IS (1 << 2): -- program specified position
VAL INT PSize        IS (1 << 3): -- program specified size
VAL INT PMinSize     IS (1 << 4): -- program specified minimum size
VAL INT PMaxSize     IS (1 << 5): -- program specified maximum size
VAL INT PResizeInc   IS (1 << 6): -- program specified resize increments
VAL INT PAspect      IS (1 << 7): -- program specified min and max aspect ratios
VAL INT PBaseSize    IS (1 << 8): -- program specified base for incrementing
VAL INT PWinGravity  IS (1 << 9): -- program specified window gravity

--  Xlib XWMHints flags
--
VAL INT InputHint          IS (1 << 0):
VAL INT StateHint          IS (1 << 1):
VAL INT IconPixmapHint     IS (1 << 2):
VAL INT IconWindowHint     IS (1 << 3):
VAL INT IconPositionHint   IS (1 << 4):
VAL INT IconMaskHint       IS (1 << 5):
VAL INT WindowGroupHint    IS (1 << 6):
VAL INT AllHints           IS #7F:


VAL INT WithdrawnState   IS 0: -- for windows that are not mapped */
VAL INT NormalState      IS 1: -- most applications want to start this way */
VAL INT IconicState      IS 3: -- application wants to start as an icon */



--  Xlib Event Mask Values
--
VAL INT32 NoEventMask        IS 0:
VAL INT32 KeyPressMask       IS (1 << 0):
```

```
VAL INT32 KeyReleaseMask IS (1 << 1):
VAL INT32 ButtonPressMask IS (1 << 2):
VAL INT32 ButtonReleaseMask IS (1 << 3):
VAL INT32 EnterWindowMask IS (1 << 4):
VAL INT32 LeaveWindowMask IS (1 << 5):
VAL INT32 PointerMotionMask IS (1 << 6):
VAL INT32 PointerMotionHintMask IS (1 << 7):
VAL INT32 Button1MotionMask IS (1 << 8):
VAL INT32 Button2MotionMask IS (1 << 9):
VAL INT32 Button3MotionMask IS (1 << 10):
VAL INT32 Button4MotionMask IS (1 << 11):
VAL INT32 Button5MotionMask IS (1 << 12):
VAL INT32 ButtonMotionMask IS (1 << 13):
VAL INT32 KeymapStateMask IS (1 << 14):
VAL INT32 ExposureMask IS (1 << 15):
VAL INT32 VisibilityChangeMask IS (1 << 16):
VAL INT32 StructureNotifyMask IS (1 << 17):
VAL INT32 ResizeRedirectMask IS (1 << 18):
VAL INT32 SubstructureNotifyMask IS (1 << 19):
VAL INT32 SubstructureRedirectMask IS (1 << 20):
VAL INT32 FocusChangeMask IS (1 << 21):
VAL INT32 PropertyChangeMask IS (1 << 22):
VAL INT32 ColormapChangeMask IS (1 << 23):
VAL INT32 OwnerGrabButtonMask IS (1 << 24):

-- Xlib XEvent Values
--
VAL INT KeyPress IS 2:
VAL INT KeyRelease IS 3:
VAL INT ButtonPress IS 4:
VAL INT ButtonRelease IS 5:
VAL INT MotionNotify IS 6:
VAL INT EnterNotify IS 7:
VAL INT LeaveNotify IS 8:
VAL INT FocusIn IS 9:
VAL INT FocusOut IS 10:
VAL INT KeymapNotify IS 11:
VAL INT Expose IS 12:
VAL INT GraphicsExpose IS 13:
VAL INT NoExpose IS 14:
VAL INT VisibilityNotify IS 15:
VAL INT CreateNotify IS 16:
VAL INT DestroyNotify IS 17:
VAL INT UnmapNotify IS 18:
VAL INT MapNotify IS 19:
VAL INT MapRequest IS 20:
VAL INT ReparentNotify IS 21:
VAL INT ConfigureNotify IS 22:
VAL INT ConfigureRequest IS 23:
VAL INT GravityNotify IS 24:
VAL INT ResizeRequest IS 25:
VAL INT CirculateNotify IS 26:
VAL INT CirculateRequest IS 27:
VAL INT PropertyNotify IS 28:
```

```
VAL INT SelectionClear       IS 29:
VAL INT SelectionRequest     IS 30:
VAL INT SelectionNotify      IS 31:
VAL INT ColormapNotify       IS 32:
VAL INT ClientMessage             IS 33:
VAL INT MappingNotify             IS 34:


#INCLUDE "simpleio.inc"


PROC  out.str (VAL []BYTE s)
  VAL length IS SIZE s:
  SEQ
    SEQ i = 0 FOR length
      SEQ
        stdout ! s [i]
:

PROC  out.num (VAL INT s)
  SEQ
    out.number(s, 10, stdout)
:


PROC  TRACE (VAL []BYTE s)
  SEQ
    out.str("*c*n")
    out.str("TRACE: ")
    out.str(s)
    out.str("*c*n")
:

PROC  EVENT (VAL []BYTE s)
  SEQ
    out.str("*c*n")
    out.str("EVENT: ")
    out.str(s)
    out.str("*c*n")
:

PROC  ERROR (VAL []BYTE s)
  SEQ
    out.str("ERROR: ")
    out.str(s)
    out.str("*c*n")
:

PROC  OK (VAL []BYTE s)
  SEQ
    out.str("OK: ")
    out.str(s)
    out.str("*c*n")
:
```

```
PROC  CHECK.RESULT (INT r, VAL []BYTE s)
  SEQ
    IF
      (r = 0)
        ERROR(s)
      (TRUE)
        OK(s)
:

PROC  CHECK.RESULT32 (INT32 r, VAL []BYTE s)
  SEQ
    IF
      (r = 0)
        ERROR(s)
      (TRUE)
        OK(s)
:


-- Local Procedures...
--
PROC load.font()
  SEQ
    -- Load font and info structure
    TRACE("XLoadQueryFont")
    C.XLoadQueryFont(font.info, display, c.font.name)
    CHECK.RESULT32(font.info, "XLoadQueryFont")
:


PROC getGC ()

  -- Xlib Line type values...
  --
  VAL INT LineSolid       IS 0:
  VAL INT LineOnOffDash   IS 1:
  VAL INT LineDoubleDash  IS 2:
  VAL INT CapNotLast      IS 0:
  VAL INT CapButt         IS 1:
  VAL INT CapRound        IS 2:
  VAL INT CapProjecting   IS 3:
  VAL INT JoinMiter       IS 0:
  VAL INT JoinRound       IS 1:
  VAL INT JoinBevel       IS 2:

  -- Local variables
  --
  INT32 values:                              -- XGCValues       !!! Local space !!!
  [C.XGCValues]INT GCFiller1:

  VAL INT32 valuemask   IS  0:                        -- unsigned long
  VAL INT   line.width  IS  6:                 -- unsigned int
  VAL INT   line.style  IS  LineOnOffDash:                -- int
  VAL INT   cap.style   IS  CapRound:                     -- int
```

```
 VAL INT   join.style  IS  JoinRound:              -- int
 VAL INT   dash.offset IS  0:                -- int
 VAL INT   list.length IS  2:                -- int
 [2]BYTE   dash.list:                        -- static char []
 [10]BYTE  c.dash.list:

 SEQ


  dash.list := [#0C,#18]
  bytes.to.chars(dash.list, c.dash.list)

  -- Initialise pointers for occam allocated 'C' Xlib structure space...
  --
  ASM
   LD   ADDRESSOF  GCFiller1                          -- XGCValues
   ST   values


  -- Create default graphics Context
  TRACE("XCreateGC")
  C.XCreateGC(gc, display, win, valuemask, values)
  CHECK.RESULT32(gc , "XCreateGC")

  -- Specifiy Font
  C.STRUCTURE.XFontStruct.GET.fid(font.info, temp.i32)
  TRACE("XSetFont")
  C.XSetFont(result, display, gc, temp.i32)
  CHECK.RESULT(result , "XSetFont")

  -- Specifiy foreground
  TRACE("XSetForeground")
  C.XSetForeground(result, display, gc, black.pixel)
  CHECK.RESULT(result , "XSetForeground")

  -- Set Line Attributes
  TRACE("XSetLineAttributes")
  C.XSetLineAttributes(result, display, gc, line.width, line.style, cap.style, join.style)
  CHECK.RESULT(result , "XSetLineAttributes")

  -- Set Dashes--
  TRACE("XSetDashes")
  C.XSetDashes(result, display, gc, dash.offset, c.dash.list, 2)
  CHECK.RESULT(result, "XSetDashes")
:



PROC place.text()
 [100]BYTE c.string1, c.string2, c.string3, c.string4:
 INT       len1, len2, len3, len4:
 INT       width1, width2, width3:

 VAL []BYTE string1 IS "This is a Window":
```

```
 VAL []BYTE string2 IS "To Terminate Program, Press any key":
 VAL []BYTE string3 IS "or Button while inside window.":
 VAL []BYTE string4 IS "Screen Dimensions:":

 INT font.height, initial.y.offset, x.offset:

 INT  ascent, descent:

 SEQ

   bytes.to.chars(string1, c.string1)
   bytes.to.chars(string2, c.string2)
   bytes.to.chars(string3, c.string3)
   bytes.to.chars(string4, c.string4)


   -- Length for XTextWidth and XdDrawString
   len1 := SIZE string1
   len2 := SIZE string2
   len3 := SIZE string3

   -- Lengths for centreing
   C.XTextWidth(width1, font.info, c.string1, len1)
   C.XTextWidth(width2, font.info, c.string2, len2)
   C.XTextWidth(width3, font.info, c.string3, len3)


   C.STRUCTURE.XFontStruct.GET.ascent  (font.info, ascent)
   C.STRUCTURE.XFontStruct.GET.descent (font.info, descent)
   font.height := ascent + descent

   -- Output Text (centred)
   C.XDrawString(result, display, win, gc, (width - width1)/2, font.height, c.string1, len1)
   C.XDrawString(result, display, win, gc, (width - width2)/2, INT (height - (2*font.height)), c.string2, len2)
   C.XDrawString(result, display, win, gc, (width - width3)/2, INT (height - font.height), c.string3, len3)

:



PROC place.graphics()
 INT  l.x, l.y, l.width, l.height:
 SEQ
   l.height := height/2
   l.width  := (3*width)/4
   l.x      := (width/2) - (l.width/2)
   l.y      := (height/2) - (l.height/2)
   C.XDrawRectangle(result, display, win, gc, l.x, l.y, l.width, l.height)
   CHECK.RESULT(result, "XDrawRectangle")
:
```

```
PROC TooSmall()
  VAL []BYTE string1 IS "Too Small":
  INT y.offset, x.offset, ascent:
  [100]BYTE  c.string1:
  SEQ
    C.STRUCTURE.XFontStruct.GET.ascent  (font.info, ascent)
    y.offset := ascent + 2
    x.offset := 2

    bytes.to.chars(string1, c.string1)

    C.XDrawString(result, display, win, gc, x.offset, y.offset, c.string1, SIZE string1)
    CHECK.RESULT(result, "XDrawString")
:



--
--  ============
--  MAIN PROGRAM
--  ============
--

SEQ

  window.size := 0

  -- Initialise pointers for occam allocated 'C' Xlib structure space...
  --
  ASM
    LD   ADDRESSOF  Filler1
    ST   size.hints
    LD   ADDRESSOF  Filler2
    ST   report
    LD   ADDRESSOF  Filler3
    ST   wm.hints
    LD   ADDRESSOF  Filler4
    ST   class.hints
    LD   ADDRESSOF  Filler5
    ST   windowName
    LD   ADDRESSOF  Filler6
    ST   iconName

  -- Initialise C strings from occam strings.
  --
  bytes.to.chars(progname,     c.progname)
  bytes.to.chars(window.name,  c.window.name)
  bytes.to.chars(icon.name,    c.icon.name)
  bytes.to.chars(argc,         c.argc)
  bytes.to.chars(font.name,    c.font.name)
  c.argv := 0

  C.GETENV.DISPLAY(c.display.name)
```

```
-- Initialise INT32 (pointers) to 'C' strings
--
ASM
  LD  ADDRESSOF c.window.name
  ST  ptr.TO.c.window.name
  LD  ADDRESSOF c.progname
  ST  ptr.TO.c.progname
  LD  ADDRESSOF c.window.name
  ST  ptr.TO.c.window.name
  LD  ADDRESSOF c.icon.name
  ST  ptr.TO.c.icon.name
  LD  ADDRESSOF c.argv
  ST  ptr.TO.c.argv
  LD  ADDRESSOF c.font.name
  ST  ptr.TO.c.font.name


icon.bitmap.bits := [#FF, #FF, #AB, #AA, #55, #D5, #AB, #AA, #05, #D0, #0B, #A0, #05, #D0, #0B, #A0, #05, #D0, #0B, #A0, #05, #D0, #0B, #A0, #55, #D5, #AB, #AA, #55, #D5, #FF, #FF]



--  Connect to Server
--
TRACE("Connecting")
C.XOpenDisplay(display, c.display.name)
CHECK.RESULT32(display, "XOpenDisplay")

TRACE("Synchronizing")
C.XSynchronize(result, display, 1)


--  Get Screen Size
TRACE("Screen Info")
C.MACRO.DefaultScreen(screen.num, display)
CHECK.RESULT(screen.num, "DefaultScreen")
C.MACRO.DisplayWidth(display.width, display, screen.num)
C.MACRO.DisplayHeight(display.height, display, screen.num)
C.MACRO.RootWindow(root.window, display, screen.num)
C.MACRO.BlackPixel(black.pixel, display, screen.num)
C.MACRO.WhitePixel(white.pixel, display, screen.num)
OK("Screen Info")

--  Size Window
x := 0
y := 0
width  := display.width / 3
height := display.height / 4

--  Create Opaque Window
TRACE("XCreateSimpleWindow")
C.XCreateSimpleWindow(win, display, root.window, x, y, width, height, border.width, black.pixel, white.pixel)
OK("XCreateSimpleWindow")
```

```
-- Get Icon sizes available--
TRACE("XGetIconSizes")
C.XGetIconSizes(result, display, root.window, size.list, count)
CHECK.RESULT(result, "XGetIconSizes")


-- Create bit map for Icon
TRACE("Create bitmap Icon")
C.XCreateBitmapFromData(icon.pixmap, display, win, icon.bitmap.bits, icon.bitmap.width, icon.bitmap.height)
OK("Create bitmap Icon")


-- Set hints
temp.i := PPosition + (PSize + PMinSize)
C.STRUCTURE.XSizeHints.SET.flags      (size.hints, temp.i)
temp.i := 300
C.STRUCTURE.XSizeHints.SET.min.width  (size.hints, temp.i)
temp.i := 200
C.STRUCTURE.XSizeHints.SET.min.height (size.hints, temp.i)

-- Set Properties
TRACE("Setting Properties")


TRACE("StringToList")
C.XStringListToTextProperty(result, ptr.TO.c.window.name, 1, windowName)
CHECK.RESULT(result, "XStringListToTextProperty - window")

C.XStringListToTextProperty(result, ptr.TO.c.icon.name, 1, iconName)
CHECK.RESULT(result, "XStringListToTextProperty - icon")

temp.i := NormalState
C.STRUCTURE.XWMHints.SET.initial.state (wm.hints, temp.i)
temp.i := 1
C.STRUCTURE.XWMHints.SET.input (wm.hints, temp.i)
C.STRUCTURE.XWMHints.SET.icon.pixmap (wm.hints, icon.pixmap)
temp.i := StateHint + (IconPixmapHint + InputHint)
C.STRUCTURE.XWMHints.SET.flags (wm.hints, temp.i)

C.STRUCTURE.XClassHint.SET.res.name  (class.hints, c.progname)
C.STRUCTURE.XClassHint.SET.res.class (class.hints, c.icon.name)

TRACE("Set Properties")
C.XSetWMProperties(display, win, windowName, iconName, ptr.TO.c.argv, 0, size.hints, wm.hints, class.hints)


-- Select require event types--
TRACE("Select Events")
C.XSelectInput(result, display, win, (ExposureMask + (KeyPressMask + (ButtonPressMask + StructureNotifyMask))) )
CHECK.RESULT(result, "XSelectInput")

-- Init Font--
TRACE("Load Font")
load.font()
```

```
-- Create GC for Text and Drawing--
TRACE("Create GC")
getGC()

-- Display Window--
TRACE("Display Window")
C.XMapWindow(result, display, win)
CHECK.RESULT(result, "XMapWindow")


--
-- Event handler
--
TRACE("Event Loop")
Alive := TRUE
WHILE (Alive)
  SEQ
    C.XNextEvent(result, display, report)
    CHECK.RESULT(result, "XNextEvent")
    C.STRUCTURE.XEvent.GET.type(report, report.type)

    IF
      (report.type = Expose)
        SEQ
          EVENT("Expose")
          C.STRUCTURE.XEvent.GET.xexpose.count(report, report.xexpose.count)
          IF
            (report.xexpose.count = 0)
              SEQ
                IF
                  (window.size = TOO.SMALL)
                    TooSmall()
                  (TRUE)
                    SEQ
                      place.text()
                      place.graphics()
            (TRUE)
              SKIP


      (report.type = ConfigureNotify)
        SEQ
          EVENT("ConfigureNotify")
          C.STRUCTURE.XEvent.GET.xconfigure.width(report, width)
          C.STRUCTURE.XEvent.GET.xconfigure.height(report, height)
          C.STRUCTURE.XSizeHints.GET.min.width(size.hints, shmw)
          C.STRUCTURE.XSizeHints.GET.min.height(size.hints, shmh)
          out.str("*c*n     width  ")
          out.num(width)
          out.str("*c*n     height ")
          out.num(height)
          out.str("*c*n min width  ")
          out.num(shmw)
          out.str("*c*n min height ")
```

```
      out.num(shmh)
      out.str("*c*n")
      IF
        ((width < shmw) OR (height < shmh))
          window.size := TOO.SMALL
        (TRUE)
          window.size := BIG.ENOUGH


  ((report.type = ButtonPress) OR (report.type = KeyPress))
    SEQ
      EVENT("ButtonPress/KeyPress")
      C.STRUCTURE.XFontStruct.GET.fid(font.info, temp.i32)
      C.XUnloadFont(result, display, temp.i32)
      C.XFreeGC(result, display, gc)
      C.XCloseDisplay(result, display)
      Alive := FALSE


  (TRUE)
    SEQ
      Alive := TRUE
      out.str("Xlib Event: ")
      out.num(report.type)
      out.str("*c*n")
      ERROR("Event Loop:  Unknown Event, Ignoring")
:
```