— CHAPTER 10 —

XCONTROL

The Extensible Control Panel

Overview

XCONTROL is a desk accessory which provides a shell for Control Panel Extensions (CPX's). Typical uses for CPX's include:

- System Configuration (volume, key click, etc.)
- Hardware Configuration (serial port speed, disk access rate, etc.)
- TSR Configuration

Most CPX's require only 512 bytes of system memory for header storage when not being executed as they are loaded only when selected by the user.

Applications, games, and other programs not used for configuration purposes should not be created as CPX's.

CPX Executable Format

A CPX executable is identical to a standard **GEMDOS** executable with the exception of an additional 512 byte header which precedes the standard 28 byte **GEMDOS** header. When **XCONTROL** is initialized at boot time, the header of each CPX contained in the user's designated CPX directory is loaded and stored. The header data contains the following information:

```
typedef struct _cpxhead
{
       UWORD magic;                   /* Magic = 100 dec */

       struct {
            unsigned reserved : 13;   /* Reserved */
            unsigned resident : 1;        /* Resident CPX if set */
            unsigned bootinit : 1;        /* Boot initialize if set*/
            unsigned setonly  : 1;        /* Set only CPX if set */
       } flags;

       LONG      cpx_id;              /* CPX ID Value */
       UWORD     cpx_version;         /* CPX Version */
       char      i_text[14];          /* Icon Text */
       UWORD     sm_icon[48];         /* Icon Bitmap 32x24 */
       UWORD     i_color;             /* Icon Color */
       char      title[18];           /* Title (16 char max) */
       UWORD     t_color;             /* Title text color */
       char      buffer[64];          /* User-storage */
       char      reserved[306];       /* Reserved */
} CPXHEAD;
```

Following the 512-byte CPX header the 28-byte **GEMDOS** header and executable follow. CPX's do not have a '**main()**' function. Execution begins at the first instruction of the TEXT segment. The first source file you should link should resemble the following:

```
          .xref    _cpx_init
          .text
cpxstart:
          jmp      _cpx_init

          .end
```

Every CPX must have a **cpx_init()** function.

If you plan to store defaults back into the CPX using **CPX_Save()** (described later) you should add to the first source file a statement allocating as much storage as you will need at the beginning of the DATA segment. For example, the following is a complete stub for a CPX requiring 10 **LONG**s of data for permanent storage.

```
          .xref    _cpx_init
          .globl   _save_vars

          .text
cpxstart:
          jmp      _cpx_init

      .data

_save_vars:
          .dc.l        0, 0, 0, 0, 0, 0, 0, 0, 0, 0

          .end
```

XCONTROL Structures

CPXINFO

A pointer to a CPX's **CPXINFO** structure must be returned by the **cpx_init()** function ('Set Only' CPX's return **NULL**). The **CPXINFO** structure is filled in with pointers to user functions as follows:

```
typedef struct
{
       WORD (*cpx_call)( GRECT * );
       VOID (*cpx_draw)( GRECT * );
       VOID (*cpx_wmove)( GRECT * );
       VOID (*cpx_timer)( WORD * );
       VOID (*cpx_key)( WORD, WORD, WORD * );
       VOID (*cpx_button)( MRETS *, WORD * );
       VOID (*cpx_m1)( MRETS *, WORD * );
       VOID (*cpx_m2)( MRETS *, WORD * );
       WORD (*cpx_hook)( WORD, WORD *, MRETS *, WORD *, WORD * );
       WORD (*cpx_close)( WORD );
} CPXINFO;
```

Form CPX's use only **cpx_call()** and (optionally) **cpx_close()**. Event CPX's use the remaining members. Members not being used should be set to **NULL**.

XCPB

A pointer to the "XControl Parameter Block" is passed to the **cpx_call()** function. This pointer should be copied to a static variable on entry so that other functions may utilize its members. **XCPB** is defined as follows:

```
typedef struct
{
       WORD      handle;
       WORD      booting;
       WORD      reserved;
       WORD      SkipRshFix;
       VOID      *reserve1;
       VOID      *reserve2;
       VOID      (*rsh_fix)( WORD, WORD, WORD, WORD, OBJECT *, TEDINFO *, char *,
                 ICONBLK *, BITBLK *, LONG *, LONG *, LONG *,   VOID * );
       VOID      (*rsh_obfix)( OBJECT *, WORD );
       WORD      (*Popup)( char *items[], WORD, WORD, WORD,
                 GRECT *, GRECT * );
       VOID      (*Sl_size)( OBJECT *, WORD, WORD, WORD, WORD,
             WORD, WORD );
       VOID      (*Sl_x)( OBJECT *, WORD, WORD, WORD, WORD, WORD,
                 void (*)();
       VOID      (*Sl_y)( OBJECT *, WORD, WORD, WORD, WORD, WORD,
                 void (*)() );
       VOID      (*Sl_arrow)( OBJECT *, WORD, WORD, WORD, WORD,
             WORD, WORD, WORD *, WORD, void (*)() );
       VOID      (*Sl_dragx)( OBJECT *, WORD, WORD, WORD, WORD,
             WORD *, void (*)() );
       VOID      (*Sl_dragy)( OBJECT *, WORD, WORD, WORD, WORD,
             WORD *, void (*)() );
       WORD      (*Xform_do)( OBJECT *, WORD, WORD * );
       GRECT *   (*GetFirstRect)( GRECT * );
       GRECT *   (*GetNextRect)( VOID );
       VOID      (*Set_Evnt_Mask)( WORD, MOBLK *, MOBLK *, LONG );
       WORD      (*XGen_Alert)( WORD );
       WORD      (*CPX_Save)( VOID *, LONG );
       VOID *    (*Get_Buffer)( VOID );
       WORD      (*getcookie)( LONG, LONG * );
       WORD      Country_Code;
       VOID      (*MFSave)( WORD, MFORM * );
} XCPB;
```

Almost all of **XCPB**'s members are pointers to utility functions covered in the **XCONTROL** Function Reference at the end of this chapter. The remaining utilized members have the following meaning:

XCPB Member	Meaning
handle	This value contains the physical workstation handle returned by **graf_handle()** to the Control Panel for use in calling **v_opnvwk()**.
booting	When **XCONTROL** is initializing as the result of a power-on, reset, or resolution change, it loads each CPX and calls its **cpx_init()** function with *booting* set to **TRUE**. At all other times, **XCONTROL** sets *booting* to **FALSE**.

SkipRshFix	When a CPX is first called after being loaded, its *SkipRshFix* flag is set to **FALSE**. The application should then use **xcpb->rsh_fix()** to fix its internal resource tree. **xcpb->rsh_fix()** sets the CPX's *SkipRshFlag* to **TRUE** so that the CPX can skip this step on subsequent calls.
Country_Code	This value indicates the country which this version of the Control Panel was compiled for as follows:

Country_Code	**Country**
0	USA
1	Germany
2	France
3	United Kingdom
4	Spain
5	Italy
6	Sweden
7	Swiss (French)
8	Swiss (German)
9	Turkey
10	Finland
11	Norway
12	Denmark
13	Saudi Arabia
14	Holland

CPX Flavors

Boot Initialization

Any CPX which has its *_cpxhead.bootinit* flag set will have its **cpx_init()** function called when **XCONTROL** initializes upon bootup. This provides a way for CPX's to set system configuration from data the user has saved in previous sessions.

cpx_init() is always called each time the user selects your CPX from the **XCONTROL** CPX list prior to calling **cpx_call()**. If the CPX is being initialized at boot time, the *xcpb->booting* flag will be **TRUE**.

Resident CPX's

CPX's which have their *_cpxhead.resident* flag set will be retained in memory after being initialized at bootup. In general, this option should not be used unless absolutely necessary.

Resident CPX's should be aware that variables stored in their DATA and BSS segments will not be reinitialized each time the CPX is called.

Set-Only CPX's

Set-Only CPX's are designed to initialize system configuration options each time **XCONTROL** initializes (during boot-ups and resolution changes) by calling the **cpx_init()** function. These CPX's will not appear in the **XCONTROL** list of CPX's.

Form CPX's

Every CPX must be either a 'Form' or 'Event' CPX. Most CPX's will be Form CPX's.

In a Form CPX, **XCONTROL** handles most user-interaction and messaging by relaying messages through a callback function. **XCONTROL** is responsible for redraws (although the CPX does have a hook to do non-**AES** object redraws) and form handling. A simple 'C' outline for a Form CPX follows:

```
/* Example Form CPX Skeleton */

#include "skel.h"
#include "skel.rsh"
#include <cpxdata.h>

CPXINFO *cpx_init();
BOOLEAN cpx_call();

XCPB *xcpb;
CPXINFO cpxinfo;

CPXINFO
*cpx_init( Xcpb )
XCPB *Xcpb;
{
       xcpb = Xcpb;

       appl_init();

       if(xcpb->booting)
       {

         /* CPX's that do boot-time initialization do it here */

         /* Returning TRUE here tells XCONTROL to retain the header
          * for later access by the user. If CPX is Set-Only,
          * return FALSE.
          */

         return ( (CPXINFO *) TRUE )
       }
       else
       {
         /* If you haven't already done so, fix resource tree.
          *
          * DEFINE's and variables are from an RSH file generated
          * by the Atari Resource Construction Set.
          */

         if(!SkipRshFix)
```

```
                (*xcpb->rsh_fix)( NUM_OBS, NUM_FRSTR, NUM_FRIMG,    NUM_TREE,
        rs_object,                  rs_tedinfo, rs_strings, rs_iconblk, rs_bitblk,
        rs_frstr, rs_frimg,                 rs_trindex, rs_imdope );

          cpxinfo.cpx_call = cpx_call;
          cpxinfo.cpx_draw = NULL;
          cpxinfo.cpx_wmove = NULL;
          cpxinfo.cpx_timer = NULL;
          cpxinfo.cpx_key = NULL;
          cpxinfo.cpx_button = NULL;
          cpxinfo.cpx_m1 = NULL;
          cpxinfo.cpx_m2 = NULL;
          cpxinfo.cpx_hook = NULL;
          cpxinfo.cpx_close = NULL;

          /* Tell XCONTROL to send generic and keyboard
           * messages.
           */

          return ( &cpxinfo );
        }
}

BOOLEAN
cpx_call( rect )
GRECT *rect;
{
        /* Put MAINFORM tree in *tree for object macros */

        OBJECT *tree = (OBJECT *)rs_trindex[ MAINFORM ];
        WORD button, quit = FALSE;
        WORD msg[8];

        ObX( ROOT ) = rect->g_x;
        ObY( ROOT ) = rect->g_y;

        objc_draw( tree, ROOT, MAX_DEPTH, PTRS( rect ) );

        do
        {
          button = (*xcpb->Xform_do)( tree, 0, msg );

          /* Be sure and mask off double-clicks if you're
           * not interested in them.
           */

          if( ( button & 0x8000 ) && ( button != 0xFFFF ) ) {
              button &= 0x7FFF;

          button &= 0x7FFF;

          switch( button )
          {
              /* Check for EXIT or TOUCHEXIT resource objects */

              case OK:
                  break;
              case CANCEL:
                  break;
              case -1:
                    switch( msg[0] )
                    {
                        case WM_REDRAW:
                            break;
                        case AC_CLOSE:
                            quit = TRUE;
                            break;
                        case WM_CLOSED:
                            quit = TRUE;
                            break;
                        case CT_KEY:
                            break;
                    }
                    break;
           }
        } while( !quit );

        return( FALSE );
}
```

Event CPX's

CPX's which are not possible as Form CPX's may be designed as Event CPX's.

Event CPX's accomplish most of their work in several callback functions identified to the Control Panel by the **CPXINFO** structure and called when the appropriate message is received. An outline for a typical Event CPX follows:

```
/* Example Event CPX Skeleton */

#include “skel.h”
#include “skel.rsh”
#include <cpxdata.h>

CPXINFO *cpx_init();
BOOLEAN cpx_call();
void cpx_draw(), cpx_wmove(), cpx_key();

XCPB *xcpb;
CPXINFO cpxinfo;

CPXINFO
*cpx_init( Xcpb )
XCPB *Xcpb;
{
        xcpb = Xcpb;

        appl_init();

        if(xcpb->booting)
        {

          /* CPX’s that do boot-time initialization do it here */

          /* Returning TRUE here tells XCONTROL to retain the header
           * for later access by the user. If CPX is Set-Only,
           * return FALSE.
           */
          return ( (CPXINFO *) TRUE )
        }
        else
        {
          /* If you haven't already done so, fix resource tree.
           *
           * DEFINE's and variables are from RSH file generated
           * by the Atari Resource Construction Set.
           */

          if(!SkipRshFix)
               (*xcpb->rsh_fix)( NUM_OBS, NUM_FRSTR, NUM_FRIMG,    NUM_TREE,
        rs_object,                 rs_tedinfo, rs_strings, rs_iconblk, rs_bitblk,
        rs_frstr, rs_frimg,                  rs_trindex, rs_imdope );

          cpxinfo.cpx_call = cpx_call;
          cpxinfo.cpx_draw = cpx_draw;
          cpxinfo.cpx_wmove = cpx_wmove;
          cpxinfo.cpx_timer = NULL;
          cpxinfo.cpx_key = cpx_key;
          cpxinfo.cpx_button = NULL;
          cpxinfo.cpx_m1 = NULL;
          cpxinfo.cpx_m2 = NULL;
          cpxinfo.cpx_hook = NULL;
          cpxinfo.cpx_close = NULL;

          /* Tell XCONTROL to send generic and keyboard
           * messages.
           */

          (*xcpb->Set_Evnt_Mask)( MU_MESAG | MU_KEYBD, NULL, NULL, -1L );

          return ( &cpxinfo );
        }
}

BOOLEAN
cpx_call( rect )
GRECT *rect;
{
        /* Put MAINFORM tree in *tree for object macros */

        OBJECT *tree = (OBJECT *)rs_trindex[ MAINFORM ];

        ObX( ROOT ) = rect->g_x;
        ObY( ROOT ) = rect->g_y;

        objc_draw( tree, ROOT, MAX_DEPTH, PTRS( rect ) );

        return ( TRUE );
}

VOID
cpx_draw( rect )
GRECT *rect;
{
        OBJECT *tree = (OBJECT *)rs_trindex[ MAINFORM ];
        GRECT *xrect, rect;

        xrect = (*xcpb->GetFirstRect)( rect );
        while( xrect )
        {
         rect = *xrect;
         objc_draw( tree, ROOT, MAX_DEPTH, ELTS( rect ) );
         xrect = (*xcpb->GetNextRect)();
        }
}

VOID
cpx_wmove( work )
GRECT *work;
{
        OBJECT *tree = (OBJECT *)rs_trindex[ MAINFORM ];

        ObX( tree ) = work->g_x;
        ObY( tree ) = work->g_y;
}

VOID
cpx_key( kstate, key, quit )
WORD kstate, key;
WORD *quit;
{
        /* Substitute case values for values you’re interested
         * in.
         */

        switch( key )
        {
         case KEY_1:
         case KEY_2:
        }
}
```

CPX File Formats

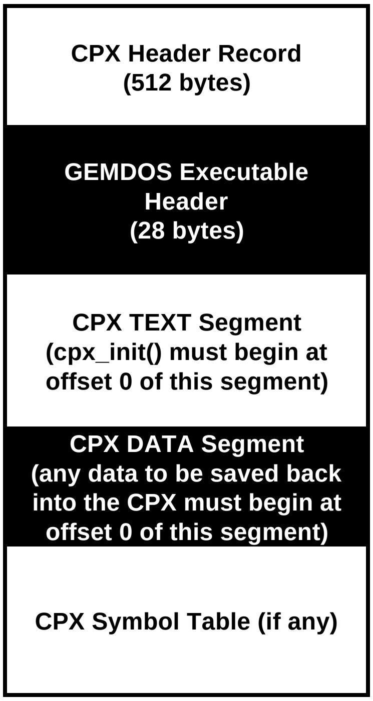
File Naming

Several standard naming conventions for CPX executables and development files follow:

File Name	Meaning
*.CPX	Standard CPX ready for execution by the Control Panel.
*.CP	CPX missing the 512 byte header.
*_R.CPX	A resident CPX.
*_S.CPX	A “Set-only” CPX.
*.HDR	A 512 byte CPX header file.
*.CPZ	An inactive CPX.
*.RSH	An “embeddable” resource file. CPX’s can’t execute a **rsrc_load()** so all resource files must be in this format.

The CPX File Format

A CPX file can be represented graphically as follows:


CPX Header Record
(512 bytes)
GEMDOS Executable Header
(28 bytes)
CPX TEXT Segment
(cpx_init() must begin at offset 0 of this segment)
CPX DATA Segment
(any data to be saved back into the CPX must begin at offset 0 of this segment)
CPX Symbol Table (if any)

XCONTROL Function Calling Procedure

Calling Conventions

XCONTROL uses "right–left" stack-based parameter passing for all of its functions and expects that user defined callback functions are similarly "right–left" stack-based. Compilers which do not default to this method should use either the 'cdecl' or '_stdargs' keyword depending on your compiler.

Function entry stubs must also consider the longword return code placed on the stack by the 68x00 'JSR' function. 'C' compilers always expect this. For example, the pointer to the **XCPB** passed to the **cpx_init()** function can be stored through the following machine language statement:

```
_cpx_init:
                move.l      4(sp),xcpb
```

Stack Space

CPX programmers should note that all CPX operations use the default Control Panel stack space (2048 bytes) and should therefore restrict heavy usage of automatic variables and other large consumers of stack space.