Using C to Get a Pointer to the CLI Window

by Rob Wyesham

Programs sometime need to get a pointer to their CLI window. A method for obtaining this pointer was illustrated in the *Amiga Mail* article entitled "CLIColors.asm - Using ConsolePackets in 68000 Assembler" (page II-65, July/August 1989 *Amiga Mail*). However, many C programmers would rather avoid using assembly language. This article shows how to use C to get a pointer the CLI window.

The basic method is the same whether you use C or Assembler. You send an `ACTION_DISK_INFO` packet to the console handler process which includes a pointer to an InfoData struture. When the packet is returned by AmigaDOS, the id_VolumeNode field of the InfoData structure will be filled in with a pointer to the CLI window that the console handler is using.

The program below, WindowPtr.c, follows the code of CLIColors.asm closely. Like the latter program, the strategy behind WindowPtr.c is to:

- ❏ Make sure the program was launched from a CLI.
- ❏ Send an `ACTION_DISK_INFO` packet.
- ❏ Extract the window pointer from the information returned in the packet.

The program starts by examining certain structures to determine if it was launched from a CLI. If it passes that test, the program checks its Process structure to make sure that it has a valid pointer to a console task. Next, the program allocates memory for FindWindow, a custom structure composed of a struct StandardPacket and a struct InfoData.

The program then initializes the StandardPacket fields and sends an ACTION_DISK_INFO packet to the DOS. When the packet returns, the program checks for an error. An error could possibly be caused by the CLI not having an associated window. A CLI that uses the AUX: handler rather than the CON: handler could cause such an error.

If the packet returns without an error, the id_VolumeNode field of the InfoData structure points to the CLI window. This is an ordinary C pointer, not a BPTR. The program shows that the pointer obtained does indeed point to the current CLI window by printing out a few of the window's parameters.

WindowPtr.c is listed below. It was compiled under Lattice 5.05 using the command: LC -b1 -cfist -L -v -w WindowPtr.c. Link with lc.lib and amiga.lib.

```
/*
* File name: windowptr.c
* Purpose: To illustrate how to get the pointer
*            to the current CLI window.
* Requires: Must be started from a CLI
*/

#include <libraries/dosextens.h>        /* Defines the StandardPacket
                                         * structure, the packet types,
                                         * and the Process structure
                                         */

#include <libraries/dos.h>        /* Defines the InfoData structure */
                                  /* and RETURN_XXX                 */
#include <exec/types.h>
#include <exec/memory.h>          /* Defines MEMF_XXX */
#include <stdio.h>                /* Defines NULL and stdout */
#include <stdlib.h>               /* Declares exit() */
#include <proto/exec.h>
#include <proto/intuition.h>


/* Set up the FindWindow structure. */
struct FindWindow
{
   struct StandardPacket FW_Pack;       /* A StandardPacket is all long-words */
   struct InfoData   FW_Info;           /* so InfoData is long-word-aligned   */
};                                      /* if the FindWindow struct is.       */


#define THISTASK NULL

extern VOID cleanExit( struct FindWindow *, int );
```

```
VOID main( int argc, char *argv[] )
{
    struct Process *myProcess = NULL;
    struct FindWindow *thisWindow = NULL;
    struct Window *addrThisWindow = NULL;
    SHORT minW, minH;
    USHORT maxW, maxH;
    SHORT xPos, yPos, W, H;

    /* Get address of the current task */
    myProcess = (struct Process *)FindTask( THISTASK );

    /* Make sure this program wasn't started from Workbench */
    if ( argc == 0 ) cleanExit( thisWindow, RETURN_WARN );

    /* Make sure that there is a pr_ConsoleTask */
    if ( myProcess->pr_ConsoleTask == NULL )
        cleanExit( thisWindow, RETURN_WARN );

    /* Allocate memory for the FindWindow structure */
    thisWindow = (struct FindWindow *)
             AllocMem( sizeof( struct FindWindow ), MEMF_PUBLIC|MEMF_CLEAR );
    if ( thisWindow == NULL ) cleanExit( thisWindow, RETURN_WARN );

    /* Initialize the packet */
    thisWindow->FW_Pack.sp_Msg.mn_Node.ln_Name =
                                 (char *)&thisWindow->FW_Pack.sp_Pkt;
    thisWindow->FW_Pack.sp_Pkt.dp_Link = (struct Message *)thisWindow;
    thisWindow->FW_Pack.sp_Pkt.dp_Port = &myProcess->pr_MsgPort;
    thisWindow->FW_Pack.sp_Pkt.dp_Type = ((LONG)&thisWindow->FW_Info) >> 2;

    /* Now send the packet, and wait for it to return */
    PutMsg( (struct MsgPort *)myProcess->pr_ConsoleTask,
                                    (struct Message *)thisWindow );

    (void)WaitPort( (struct MsgPort *) &myProcess->pr_MsgPort );

    /* Our program has been replied to, so get the reply message */
    (void)GetMsg( &myProcess->pr_MsgPort );

    /* Save the address of the CLI window */
    addrThisWindow = (struct Window *) (&(thisWindow->FW_Info))->id_VolumeNode;


    /* If there was an error, terminate execution */
    if ( thisWindow->FW_Pack.sp_Pkt.dp_Res1 == DOSFALSE )
              cleanExit( thisWindow, RETURN_WARN );

    if ( addrThisWindow == NULL ) cleanExit( thisWindow, RETURN_WARN );
```

```
/* Get the position, dimensions, and minimum and maximum
     * dimensions of the currently active CLI window */
    xPos = addrThisWindow->LeftEdge;
    yPos = addrThisWindow->TopEdge;
    W = addrThisWindow->Width;
    H = addrThisWindow->Height;
    minW = addrThisWindow->MinWidth;
    minH = addrThisWindow->MinHeight;
    maxW = addrThisWindow->MaxWidth;
    maxH = addrThisWindow->MaxHeight;


    /* Print the position, dimensions, and minimum and maximum
     * dimensions of the currently active CLI window */
    printf( "Position: x = %d, y = %d   ", xPos, yPos );
    printf( "Dimensions: w = %d, h = %d\n", W, H );
    printf( "%d <= width <= %d   ", minW, maxW );
    printf( "%d <= height <= %d\n", minH, maxH );

    cleanExit( thisWindow, RETURN_OK );
}


VOID cleanExit( pktptr, returnValue )
struct FindWindow *pktptr;
int returnValue;
{
    if ( pktptr ) FreeMem( pktptr, (LONG)sizeof( struct FindWindow ) );

    exit( returnValue );
}
```