PHILIPS

PHILIPS INTERACTIVE MEDIA of America

Technical Note #73.2

Reading the Player Control Keys

Charles Golvin

Revised September 1, 1992
Supersedes TN# 73.1, dated May 7, 1992

A proposed addendum to the Green Book describes the implementation of the Player Control Keys (PCKs). This note gives concrete examples of how to actually implement these keys on both the Philips 910 player and, for debugging purposes, on the Philips 18x player. This note has been revised to describe the role of the **ss_enable** function in the use of the PCKs. This revision supersedes Technical Note #73.1.

Warning: *Although the implementation of the PCKs illustrated in this note does work on the current Philips players, it is not based on a Change Control Board (CCD) approved Green Book extension. Developers who choose to add this feature to a disc should be aware that they may be forced to change to another strategy when the CCB does accept a proposal for implementation of player control keys. In the author's opinion, it is likely that only small differences will exist between the implementation described here and the proposal that will finally be endorsed by the CCB.*

Reading The Player Control Keys

A proposed addendum to the *CD-I Full Functional Specification* (Green Book) describes the proper implementation of the "player control keys." This specification has been followed in developing the Philips 910 player, and several applications have already been developed that take advantage of this feature. This note provides some code examples that may be used to read the player control keys. More importantly, this note explains how such code can be developed and tested using the Philips 18x player. For the text of the Functional Specification for the player control keys, please consult Appendix A of Technical Note #62.

Players that follow the specification for the player control keys have an entry in the CSD for the player control keys device. The entry in the Philips 910 player looks like the following:

```
11:/pck:KG="8":
```

As given in the specification, **11** is the device number. As with all CSD entries, the string that is between the first set of colons provides the name of the device that must be used to open a path to that device for reading of the player control keys—in this case, the device is called /pck.

The final string in the CSD entry specifies the key group supported by this device; the entry "8" is for the player control keys. The functions that are supported by this key group are: Play, Stop, Pause, Next Track, Previous Track, Search Backward, and Search Forward. The remote controller sold with the 910 player does not possess a button for Search Backward or Search Forward.

The front panel controls must be enabled in order to use the player control keys. Use the function **ss_enable** to enable the front panel controls and **ss_disable** to disable them. The function **ss_enable** takes a path to the CD device. You can open a path to the CD directory explicitly, or, once you have opened a path to a real-time file, you can use that path as the path to the CD device.

The basic mechanism for reading the player control keys comes from the keyboard input functions in the UCM file manager. To begin, the application must query the CSD for the correct device name and open a path to that device. Once a path is obtained, the application arms a signal for that device using the UCM function code **KB_SSig**. When the user presses a player control key, the signal number that was requested is sent to the process that armed the signal. If an event is pending in the queue when KB_SSig is called, the signal is sent immediately.

Once the signal is received, the application uses the UCM function code KB_Read to read the value of the key that was pressed. The function KB_Rdy may be used to "count" the number of key events that are currently in the queue. For more detailed information on the unique behavior of KB_SSig with respect to KB_Read and KB_Rdy, consult chapter VII, pages 206 through 208 of the Green Book. Once the key value is read, the handling of the event is the prerogative of the application. Events are generated for both "key down" and "key up" states. (Although the Green Book allows for "auto-repeat" events, I have never seen such an occurrence on either the 18x, 605, or 910 player.)

The authors of the software for the 18x player were concerned that if they placed an entry for the player control keys in the CSD of this development player, an application developer might mistakenly expect that this (non base-case) functionality would be present in all players. Therefore, the CSD data module that would be observed at startup and cause an entry in the CSD for the player control key device is absent from the 18x player. However, with this lone exception, all the necessary software for player control key implementation is in the player.

The actual key-handling code that is in the application need not change from the 18x player to the 910 player. The only addition is conditional compilation code that is used to (1) hard code the name of the player control key device on the 18x player; and (2) conditionally define the actual function codes associated with certain keys. The reason for (2) is that the key codes changed between the development of the 18x player and the finalization of the player control key specification.

If the 18x player were to contain a CSD entry for the player control key device, it would be slightly different from the 910 player, not only because of the device name, but because the 18x remote controller has keys that are not present on the 910 remote. It would be similar to the following expression:

```
11:/cdikeys:KG="3,7,8":
```

As mentioned above, key group "8" contains the player control keys. Key group "7" contains the numeric keys (0-9) and keygroup "3" is for "special keys." Examples of the latter might be the "MUTE" and "SCREEN" keys on the 18x remote controller unit.

Below is some sample code that has been used to read the player control keys on both the 18x and 605 players. This is provided only as a guide and is not necessarily the correct or best method. For example, it would be wise to remove the key-reading code from the signal handler function. The code actually compiles into a usable demonstration program. This program simply identifies the key that was pressed (if recognized; otherwise, "unknown" is printed), and the type of key event (up, down, auto-repeat) that was read.

```
/*
 * The preprocessor symbol FOR_910 is defined when the program is
 *  compiled to run on the Philips 910 player.  When this symbol is
 *  not defined, the result will run on the Philips 18x player.
 */
        /* Miscellaneous stuff not specific to PCK things */

#include <stdio.h>
#include <modes.h>
#define SYSERR                      -1
#define PCK_SIGNAL                  1024

#if defined(FOR_910)                /* This is included for the function */
#include <csd.h>                    /* csd_devname(), which is used to read the */
#endif                              /* PCK device name from the CSD */

#define DT_PCK          11          /* This should be in csd.h */

#define UNSTACK         0x0002      /* This is an input parameter to KB_Read() */

#if defined(FOR_910)  /* These are the proper key events as given in the spec */

#define PLAY                        0x80
#define STOP                        0x81
#define PAUSE                       0x82
#define NEXT_TRACK                  0x83
#define PREVIOUS_TRACK              0x84
#define SCAN_FORWARD                0x85
#define SCAN_BACKWARD               0x86

#else                               /* These are the key events on the 18x player */

#define MUTE_ON                     0x80
#define STOP                        0x81
#define PAUSE                       0x82
#define PLAY                        0x83
#define SCAN_FORWARD                0x84
#define SCAN_BACKWARD               0x85
#define NEXT_TRACK                  0x86
#define HIDE_SCREEN                 0x88
#define PREVIOUS_TRACK              0x89
#define MUTE_OFF                    0x8a
#define PLAY_PAUSE                  0x8f

#endif

#if !defined(FOR_910)  /* Hardcode the PCK device name on the 18x player */
static char *pck180nam = "/cdikeys";
#endif

extern int errno;

int pckpath, done;                  /* These are global for simplicity */
```

```
/*********************************************************
 * Signal handling function - only input is the signal number
 * the only signals expected are the PCK signal and ^E,^C to
 * quit the program
 *********************************************************/
void sig_catch( thesig )
short thesig;
{
 char typ;                          /* This will hold the type of event returned */
 short cod;                         /* This will hold the actual key returned */

 switch( thesig )
 {
  case PCK_SIGNAL:

                               /* We got the signal so read the actual key that was hit. UNSTACK
                               \* tells the system to remove the read event from the queue*/
   if ( kb_read( pckpath, UNSTACK, &cod, &typ) == SYSERR )
   { fprintf( stderr, "error reading pck\n" ); done=1; break; }

   printf( "Got a key: " );
   switch ( cod )                   /* What key did we get? */
   {
       case STOP:                        printf( "STOP (%x)", cod ); break;
       case PAUSE:                       printf( "PAUSE (%x)", cod ); break;
       case PLAY:                        printf( "PLAY (%x)", cod ); break;
       case NEXT_TRACK:                  printf( "NEXT_TRACK (%x)", cod ); break;
       case PREVIOUS_TRACK:              printf( "PREVIOUS_TRACK (%x)", cod ); break;
       case SCAN_FORWARD:                printf( "SCAN_FORWARD (%x)", cod ); break;
       case SCAN_BACKWARD:               printf( "SCAN_BACKWARD (%x)", cod) ; break;

#if !defined(FOR_910)               /* These are only valid on the 18x */
       case MUTE_ON:                     printf( "MUTE_ON (%x)", cod ); break;
       case HIDE_SCREEN:                 printf( "HIDE_SCREEN (%x)", cod ); break;
       case MUTE_OFF:                    printf( "MUTE_OFF (%x)", cod ); break;
       case PLAY_PAUSE:                  printf( "PLAY_PAUSE (%x)", cod ); break;
#endif
       default:                          printf( "unknown: %x", cod ); break;
   }
                                    /* What type of event was it? */
   printf( ", type is " );
   if ( typ & 4)                         printf( "key just released" );
   else if ( typ & 2 )                   printf( "key auto repeat" );
   else if ( typ & 1 )                   printf( "key just down" );
   else                                  printf( "unknown" );
   printf( "\n" );
   break;

                                         /* These signals kill the program */
  case 2: case 3:                        done = 1; break;
  default:                               printf( "Got unknown signal %d\n", thesig );
break;
 }
 return;
}
```

```
main( ac, av )
int ac;
char **av;
{
        char *pckname;                  /* Will hold the name of the PCK device */

        intercept(sig_catch);
        errno = 0;
        done = 0;

#if defined(FOR_910)                    /* On 910, read the CSD */

        if ( (pckname = csd_devname( DT_PCK, 1 )) == 0 )
        { fprintf( stderr, "Can't fetch pck name\n" );exit( errno ); }

#else

        pckname = pck180nam;            /* On 18x, use hardcoded name */

#endif

        if ( (pckpath = open( pckname, S_IREAD )) == SYSERR )
        { fprintf( stderr, "Can't open %s\n",pckname ); exit( errno ); }

#if defined(FOR_910)
        free( pckname );                /* Release malloc'd memory */
#endif
        while ( !done )
        {
                if ( kb_ssig(pckpath,PCKSIG) == SYSERR )
                { fprintf( stderr, "Can't arm PCK signal\n" ); break; }
                sleep(0);
        }
        close( pckpath );
        exit( errno );
}
```