NAME

xil_cis_attempt_recovery – attempt recovery after an error occurs in a compressed image sequence

SYNOPSIS

#include <xil/xil.h>

void xil_cis_attempt_recovery (XilCis *cis*,
unsigned int *nframes*,
unsigned int *nbytes***);**

DESCRIPTION

This function is used to attempt recovery from a non-autorecoverable error that occurs during the playback of a compressed image sequence (CIS). *cis* is the input CIS in which an error occurred. *nframes* is the maximum number of frames ahead that will be scanned through to recover from a non-autorecoverable error. *nbytes* is the maximum number of bytes ahead that will be scanned through to recover from an error. If both of these values are zero, then the attempt at recovery will search forward as many bytes or frames as necessary. If *nframes* is non-zero and *nbytes* is zero, then the error recovery mechanism will attempt to search *nframes* ahead with its best approximation of exactly how many bytes that should be. If *nbytes* is non-zero and *nframes* is zero, the search will go through *nbytes,* regardless of the number of frames.

xil_cis_attempt_recovery () only needs to be called for non-autorecoverable errors. Consult **xil_cis_get_autorecover**(3) for details.

Both autorecoverable and non-autorecoverable errors are reported to the user through the error handling mechanism. The user decides whether to attempt recovery of a non-autorecoverable error.

If the error is recoverable, after reporting the error, the attribute *AUTO_RECOVER* (see **xil_cis_get_autorecover**(3)) is checked to determine whether to attempt recovery. If the attribute is set to TRUE, recovery is attempted.

Non-autorecoverable errors are handled similarly, except the *AUTO_RECOVER* attribute has no effect on how these errors are handled. When a non-autorecoverable error is detected, the CIS is marked invalid before the user is notified of the error. The CIS is marked CIS_READ_INVALID for decompression and CIS_WRITE_INVALID for compression (see **xil_cis_get_read_invalid**(3) and **xil_cis_get_write_invalid**(3)). Thus, if an error occurs in one of the decompression routines, then compression routines or **xil_cis_put_bits**(3) can still write into the CIS.

After a non-autorecoverable error has occurred, the user can validate the CIS in one of three ways: by calling **xil_cis_reset**(3), seeking to a valid frame, or asking XIL to attempt recovery using **xil_cis_attempt_recovery** (). If the user attempts to seek to a valid frame and the CIS cannot successfully complete the request, a seek error is generated.

To find out where the CIS is located after the call to **xil_cis_attempt_recovery** (), use **xil_cis_get_read_frame**(3) to get the bext approximation of the current *read_frame,* and **xil_cis_has_data**(3) to get the exact number of bytes left in the CIS. By checking and comparing the values returned by **xil_cis_has_data** () before and after calls to **xil_cis_attempt_recovery** (), you can determine the exact number of bytes that were searched through. It is also possible to determine the approximate number of frames that were skipped by checking and comparing the values returned by **xil_cis_get_read_frame**(3) before and after calls to **xil_cis_attempt_recovery** (). If **xil_cis_attempt_recovery** () succeeds, the CIS is returned to a VALID state. You can determine whether this function was successful by testing the state of the CIS. If you set the number of bytes or number of frames to check through to a low value, multiple calls to this function may be necessary.

CIS error recovery has been implemented so that **xil_cis_attempt_recovery** () can be called from the error handling function itself. If this function is called during the error handling function, the current decompress call will fail regardless of whether recovery was successful, the CIS will be marked VALID, and the next decompress call will succeed (unless another error is encountered).

ERRORS

For a complete list of XIL error messages by number, consult Appendix B of the *XIL Programmer's Guide.*

EXAMPLES

In this example, when an error occurs, the error handler is called and the user gives the CIS permission to search indefinitely in an attempt to recover. If the attempt is unsuccessful, then **xil_cis_has_data**(3) fails, and the decompression loop is halted as if the video concluded.

```
/*
 * Example Error Recovery
 */

Xil_boolean
 my_error_handler( XilError error )
{

   XilCis cis;
   XilObject obj;

   /* If an XIL-CIS error occurred */
   if ( ( xil_error_get_category(error) == XIL_ERROR_CIS_DATA ) &&
       ( (obj = xil_error_get_object(error)) != NULL) &&
       ( xil_object_get_type(obj) == XIL_CIS ) ) {

     cis = (XilCis)obj;

     /* Has an error occured that we can attempt to
      *   recover from?  If so, attempt recovery.
```

```
            */
           if (xil_cis_get_read_invalid(cis)) {

             xil_cis_attempt_recovery(cis, 0, 0);

             /* If the CIS is now OK, we've handled it correctly. */
             if (!xil_cis_get_read_invalid(cis))
                return TRUE;
           }
     }
     return xil_call_next_error_handler(error);

}

main() {

   XilCis cis;
   XilSystemState state;
   XilImage image;
   XilImage displayimage;
   XilLookup lookup;

   if ( ( state = xil_open() ) == NULL ) {
      printf(" Couldn't initialize XIL\n");
      exit(1);
   }

   /* install error handler */
   xil_install_error_handler( state, my_error_handler );

   while(xil_cis_has_data(cis)) {
      xil_decompress(cis, image);
      xil_nearest_color(image, displayimage, lookup);
   }
}
```

NOTES

Occasionally, it is possible that error recovery may revalidate the CIS, but be off-sync from the number of frames that would have been in the CIS if the data had been correct. This can cause another error later, when the CIS reaches the end of the data inserted through the **xil_cis_put_bits**(3) call and then finds that the number of frames that it decoded from the data chunk is different than what the user said was in it.

SEE ALSO

xil_cis_get_autorecover(3), **xil_cis_get_read_invalid**(3), **xil_cis_get_write_invalid**(3), **xil_cis_get_read_frame**(3), **xil_cis_put_bits**(3), **xil_call_next_error_handler**(3), **xil_cis_reset**(3).