handspring

Handspring's Try-and-Catch APIs

Application Note AN-05 Revision 1.01 Document No. 80-0121-00

Table of Contents

1. Overview

This document describes the use of Handspring's `HsCardErrTry` and `HsCardErrCatch` APIs. The `Try` block allows safe access to the module. If the module is removed during access to a Springboard module, the system will recover safely by trapping the bus error fault and returning control to the `Catch` block for proper clean up.

Prototype

```
HsCardErrTry
{
// Do something that accesses the removable module
}

HsCardErrCatch
{
// Recover or clean up after a failure in the above Try block.
// The code in this Catch block does not execute if
// the above Try block completes without a module removal
} HsCardErrEnd

// You must structure your code exactly as above. You cannot have a
// HsCardErrTry { } without a HsCardErrCatch { } HsCardErrEnd,
// or vice versa.
```

2. Usage

The `HsCardErrTry/HsCardErrCatch` macros should be wrapped around any section of code within an interrupt handler, system extension, shared library, or other system code that needs to access memory or hardware on a removable module. If the module is removed while the critical section of code is executing, control is first passed to the `HsCardErrCatch()` section.

These macros can be nested. For example, you can call a subroutine from within your `HsCardErrTry` block that has its own `try/catch` block. Every routine that has an `HsCardErrTry` clause, however, must have an `HsCardErrCatch`.

2.1. Restrictions

`HsCardErrTry` and `HsCardErrCatch` are based on `setjmp/longjmp`. At the beginning of a `Try` block, `setjmp` saves the machine registers. A module removal triggers `longjmp`, which restores the registers and jumps to the beginning of the `Catch` block. Therefore, changes in the `Try` block to variables that are stored in registers are not retained, and will be lost when entering the `Catch` block. For variables that are referenced in the `Try` block and are referenced when the `Catch` block is activated, the variables must be declared as "volatile." If the variables are not referenced when the `Catch` block is activated, they do not need to be declared as "volatile".

The `HsCardErrTry` or `HsCardErrCatch` blocks must not contain `return` or `goto` statements. When the `HsCardErrTry` block fails, the `HsCardErrCatch` block must execute completely to the end of the `HsCardErrEnd` macro to properly restore the machine registers. If the code leaves the `HsCardErrTry` block prematurely, it will corrupt the `HsCardErrTry/HsCardErrCatch` exception list, resulting in unpredictable system behavior.

2.2. Sample Source Code

```
static void
AppEventLoop(void)
{
  short             err;
  EventType event;
  volatile   Boolean      needAbort = false;

  do
       {
         EvtGetEvent (&event, sysTicksPerSecond/4);

         HsCardErrTry
              {
                // Access card in some manner that may fail
                if (*((Byte*)0x28000000UL))
                     needAbort = false;
              }

         HsCardErrCatch
              {
                // Recover or cleanup after a failure in above Try block
                // The code in this block does NOT execute if the above
                //  try block completes without a card removal

                needAbort = true;
              }
         HsCardErrEnd

         if (needAbort) return;

         if (!SysHandleEvent (&event))

              if (!MenuHandleEvent (0, &event, &err))

                if (!AppHandleEvent (&event))

                     FrmDispatchEvent (&event);

       } while(event.eType != appStopEvent);
}
```

3. History

Date	Revision #	Description of changes
13 Dec 00	1.01	Reformat.
3 Apr 00	1.00	Initial release.