Why is there a RestoreLastError function that does the same thing as SetLastError?

devblogs.microsoft.com/oldnewthing/20110429-00

April 29, 2011

Raymond Chen

Matt Pietrek noticed that `SetLastError` and `RestoreLastError` do exactly the same thing and wondered why there's a separate function for it.

It's to assist in debugging and diagnostics.

Say you're debugging a problem and when you call `GetLastError` you get `ERROR_ACCESS_DENIED`. It would really help a lot if you could figure out who set the error code to `ERROR_ACCESS_DENIED`. If you set a breakpoint on `SetLastError`, you find that people call `SetLastError` for two different reasons:

1. To report an error.
2. To restore the error code to what it was before they did something that might change the last error code.

That second one needs a little explanation. You might have a logging function that goes like this:

```
// Remember, code in italics is wrong
void LogSomething(blah blah)
{
 DWORD dwError = GetLastError();
 ... do logging stuff ...
 SetLastError(dwError);
}
// or if you prefer RAII
class PreserveLastError
{
public:
    PreserveLastError() : m_dwLastError(GetLastError()) {}
    ~PreserveLastError() { SetLastError(m_dwLastError); }
private:
    DWORD m_dwLastError;
};
void LogSomething(blah blah)
{
 PreserveLastError preserve;
 ... do logging stuff ...
}
```

It's important that functions which perform logging, assertion checking, and other diagnostic operations are nonintrusive. You don't want a bug to go away when you turn on logging because the logging code somehow perturbed the system. Therefore, your logging function saves the value of `GetLastError()` and sets that back as the error code when it's done, so that any errors that took place during logging do not escape and inadvertently affect the rest of the program.

Now let's go back to the code that's trying to figure out who set the error code to `ERROR_ACCESS_DENIED`. You set up your debugging diagnostic tool and tell it to record everybody who calls `SetLastError()` and pay particular attention to everybody who sets the error to `ERROR_ACCESS_DENIED`. You then run your scenario, your program encounters the failure you're trying to debug, and you ask the diagnostic tool, "Tell me who set the error code to `ERROR_ACCESS_DENIED`." The diagnostic tool says, "Ah, I have that in my history. The function that set the error code to `ERROR_ACCESS_DENIED` is… `LogSomething`!"

Of course, `LogSomething` wasn't really the originator of the `ERROR_ACCESS_DENIED`; it was just restoring things to how it found them. The real `ERROR_ACCESS_DENIED` came from somebody else, and the log function was just being careful not to disturb it.

```
...
  if (!FunctionX()) {
    LogSomething("Function X failed");
  } else {
    LogSomething("Function X succeeded");
   FunctionY(); // also does some logging
  }
  FunctionZ(); // also does some logging
  Assert(EverythingOkay()); // assertion fires
  // GetLastError() returns ERROR_ACCESS_DENIED
...
```

All those calls to logging functions in between called `GetLastError()` and got `ERROR_ACCESS_DENIED` back, then when the logging was complete, they called `SetLast-Error(ERROR_ACCESS_DENIED)` to put things back. Your diagnostic error-tracing tool gleefully points the finger at your logging function: "Look! Look! This guy set the error code to `ERROR_ACCESS_DENIED`!"

Enter `RestoreLastError`. This function does the same thing as `SetLastError`, but its use is a message to diagnostic tools that "Sure, you may see me set an error code, but *it wasn't my idea*. I'm just trying to put things back the way I found them. *Keep looking backwards in your history*."

(The message also works forward in time: If you want to catch `ERROR_ACCESS_DENIED` in the act, you might set a breakpoint on `SetLastError`, and then get frustrated that the breakpoint keeps getting hit by your logging function. Switching the logging function to `RestoreLastError` keeps the breakpoint on `SetLastError` from firing spuriously.)

The corrected version of the `LogSomething` function is therefore something like this:

```
void LogSomething(blah blah)
{
 DWORD dwError = GetLastError();
 ... do logging stuff ...
 RestoreLastError(dwError);
}
// or if you prefer RAII
class PreserveLastError
{
public:
    PreserveLastError() : m_dwLastError(GetLastError()) {}
    ~PreserveLastError() { RestoreLastError(m_dwLastError); }
private:
    DWORD m_dwLastError;
};
```

Raymond Chen

Follow