Handling Errors in a Shore Application[1]

The Shore Project Group
Computer Sciences Department
UW-Madison
Madison, WI

Version 1.1.1

October 27, 1997

[1]This research is sponsored by the Advanced Research Project Agency, ARPA order number 018 (formerly 8230), monitored by the U.S. Army Research Laboratory under contract DAAB07-92-C-Q508.

Contents

1 Introduction

This document describes error handling facilities for Shore applications. Shore application writers are encouraged to read *Getting Started with Shore* and look at the example applications before reading this document. The SDL manual and the example programs show the standard ways of handling errors in Shore. This document is intended for application writers interested in more details.

This document has three sections. The first section describes the `shrc` (Shore return code) type, which is used by most Shore methods. The second section describes transaction macros, and the third section describes how to install an error handler function.

2 The `shrc` Class

Most methods in Shore return an error indicator of type `shrc`. The shrc type contains not only an integer error code, but also a stack trace showing where the error occurred and the series of function calls that triggered the error.

Shrc is a C++ class, and it contains various member functions that application programmers may find useful. The following is a portion of the class definition of shrc (`shrc` is the same type as `w_rc_t`, which is defined in `w_rc.h`):

```
class shrc
{
 public:
    bool is_error() const;
    int  err_num() const;
    operator const void*() const;

    friend ostream &operator<<(ostream& o, const shrc &rc);
};
```

If a Shore method completes successfully, it returns the constant `RCOK`. If a Shore method returns something other than `RCOK`, then an application can use then `shrc::err_num` method to get the integer error code corresponding to the error. The possible error codes are listed in the manual page *errors(OC)*. Each error code has a `#define`ed value, which is included in your sources if you include `ShoreApp.h`.

Shrc includes `operator const void*()` to allow applications to say things like

```
    shrc rc = ...;
    if (rc) ...
```

Shrc also includes an overloaded version of the C++ << operator.

3 Macros for Transaction Management

The most common way to handle errors in database systems is to abort the transaction that caused the error. Application programs can abort their transactions by calling Shore::abort_transaction. While this method causes any changes to persistent data to be be undone, it does not affect the state of the application program. Shore provides macros to begin, commit, and abort transactions that can help to address part of this problem. The macros employ ANSI C `setjmp` and `longjmp` facilities.

```
    SH_BEGIN_TRANSACTION(rc)        // rc is an "out" parameter
    SH_ABORT_TRANSACTION(rc)        // rc is an "in"  parameter
    rc = SH_COMMIT_TRANSACTION
    SH_DO(rc)                       // rc is an "in"  parameter
```

These macros are summarized in the manual page *transaction(OC)*.

The argument to `SH_BEGIN_TRANSACTION` is of type `shrc`. This argument must be an *lvalue*, as the macro will assign to it. `SH_BEGIN_TRANSACTION` calls `setjmp`, and can therefore return from either of two different contexts: a direct call to `SH_BEGIN_TRANSACTION`, or a call to `SH_ABORT_TRANSACTION` (described below). A direct call to `SH_BEGIN_TRANSACTION` calls `Shore::begin_transaction`. Upon return, `rc` will be set to the return value of `Shore::begin_transaction`. When `SH_BEGIN_TRANSACTION` returns because of a call to `SH_ABORT_TRANSACTION`, then the value of `rc` is whatever was passed to `SH_ABORT_TRANSACTION`.

`SH_COMMIT_TRANSACTION` is equivalent to `Shore::commit_transaction`.

`SH_ABORT_TRANSACTION` takes as parameter an expression of type `shrc`. (Unlike `SH_BEGIN_TRANSACTION`, this argument is an *rvalue*, not an *lvalue*). It calls `Shore::abort_transaction` and then performs a `longjmp`, which returns control back to the line where `SH_BEGIN_TRANSACTION` was called. The rc passed to `SH_ABORT_TRANSACTION` will be the return value of `SH_BEGIN_TRANSACTION`.

`SH_DO` takes an expression of type `shrc`. If this expression evaluates to `RCOK` then the macro returns. Otherwise, `SH_ABORT_TRANSACTION` is called with the value of the given expression as its argument.

The following code fragment illustrates the use of the transaction macros.

```
shrc rc;

// Begin a transaction.  A subsequent call to SH_ABORT_TRANSACTION
// will return us here.
SH_BEGIN_TRANSACTION(rc);

if(rc){

    // Some error occurred.  The rc indicates why new transaction could
    // not be started or why the transaction was aborted.
    cerr << rc << endl;
}

else {


    // We successfully started a transaction.  The main body of the
    // transaction goes here.

    SH_DO(operation 1);
    SH_DO(operation 2);
    SH_DO(...);


    // If we completed the main body of the transaction without
    // errors, then we try to commit the transaction.  Note that
    // if the commit fails, then SH_DO will call SH_ABORT_TRANSACTION
    // for us.  The shrc returned by SH_COMMIT_TRANSACTION will become
    // the the return value of SH_BEGIN_TRANSACTION (above).

    SH_DO(SH_COMMIT_TRANSACTION);
}
```

Because these macros make use of setjmp and longjmp, the function containing the call to `SH_BEGIN_TRANSACTION` must not have terminated before `SH_DO` or `SH_ABORT_TRANSACTION` is called. However, a call to `SH_COMMIT_TRANSACTION`, `SH_DO`, or `SH_ABORT_TRANSACTION` does not have to be in the same function as the call to `SH_BEGIN_TRANSACTION`. In particular, it may be useful to call `SH_ABORT_TRANSACTION` from within the Shore error handler function, which is described in the next section.

4 The Shore Error Handler Function

In certain cases, it is impossible or impractical to return a `shrc`. For example, `ref<T>::operator->` does not have an opportunity to return an error code, because of the way C++ defines `operator->`. Ideally, errors from overloaded operators would raise an exception that could be caught and handled by the application. Unfortunately, C++ exceptions are not stable enough to be used. Therefore, in situations where it is impractical to return a `shrc`, Shore calls a handler function, passing a `shrc` as its argument. The default handler function simply prints the `shrc` and exits by calling `_exit` (this form of `exit` does not call global destructors).

Applications can install their own error handler by calling `Shore::set_error_handler`. This function takes a pointer to a handler function as its argument. It installs the given function as the new handler and returns the old handler function. Passing a zero argument to `Shore::set_error_handler` reinstalls the default handler.

By installing a handler function, an application gives itself an opportunity to clean up its transient state before aborting a transaction. Once the transient state has been cleaned up, the handler function can call `SH_ABORT_TRANSACTION` to abort the transaction and `longjmp` back to the call to `SH_BEGIN_TRANSACTION`. Alternately, the handler function can exit the process with `_exit`. A handler function should not return, however. If it does, then Shore will terminate the process by calling `_exit`.

The error-handling functions are documented in *errors(OC)* and *rc(FC)*.