Scanning Directories 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 **1**

2 The DirScan Class **1**

3 A Directory Scan Example **3**

1 Introduction

The *DirScan* class provides a convenient interface for scanning directories in Shore VAS clients. This document describes how to use the DirScan class. The Shore release includes a simple example program called *shls* ("Shore *ls*"), that illustrates the use of the DirScan class. Shls can be found in the `examples/shls` directory of the Shore release. The DirScan class is part of the SDL library, but it is not necessary to use SDL or the SDL compiler to use it.

2 The DirScan Class

An application scans a directory by creating a DirScan object. DirScan objects are transient C++ objects, not persistent Shore objects. A DirScan object can be in one of two states: *open* or *closed*. A scan must be opened inside a transaction, and can only be used inside the transaction in which it was opened. Once a scan is opened, it stays open until it is explicitly closed, or until the scan object is destroyed, or the transaction terminates. In particular, the scan object remains open even after the end of the scan has been reached or an error has occurred.

Directory scans request blocks of directory entries from the Shore server and store them in a buffer. Applications can set the size of this buffer, although the size must be at least `DEF_DIRSCAN_BUFSIZE` to ensure that the buffer is large enough to hold at least one directory entry. If a smaller buffer size is requested, it will be replaced by the default size.

A scan of a directory returns some number of `DirEntry` structures. The `DirEntry` type (defined in `OCTypes.h`) looks like this:

```
struct DirEntry {
    LOID loid;
    int namelen;
    char name[MAXNAMLEN + 1];
};
```

The members of the structure are

`loid`, the logical OID of the object named by the directory entry,

`namelen`, the length of the `name` field, and

`name`, the null-terminated name of the entry (the last component of the pathname of the named object).

The public interface to the DirScan class is

```
class DirScan {
 public:
    DirScan();
    DirScan(const char *path, int bufsize = DEF_DIRSCAN_BUFSIZE);

    shrc open(const char *path, int bufsize = DEF_DIRSCAN_BUFSIZE);

    shrc next(DirEntry *entry);

    bool is_open();
    shrc rc();
    int operator==(shrc &rc);
    int operator!=(shrc &rc);

    shrc close();
    ~DirScan();
};
```

`DirScan()` is the default constructor. The resulting DirScan object is in the *closed* state. It may be opened by the `open` member function.

`DirScan(const char *path, int bufsize).` This constructor invokes the `open` member function. The caller should whether the `open` operation was successful by calling `is_open()` or by testing the return code with `rc()`, `operator==()`, or `operator!=()`.

`shrc open(const char *path, int bufsize).` The `path` argument should be the pathname of a valid directory in the Shore filesystem namespace, and the DirScan object should be in the *closed* state. The caller must have (at least) read permission for the named directory and must have a transaction open. The `bufsize` argument indicates the size of the scan's buffer. This parameter defaults to `DEF_DIRSCAN_BUFSIZE`, which is large enough to hold at least one directory entry. Any smaller value for this argument is treated as `DEF_DIRSCAN_BUFSIZE`. Upon successful completion, `RCOK` is returned. Any other return value indicates an error condition.

`shrc next(DirEntry *entry)` copies the next director entry into the DirEntry structure pointed to by the argument. Returns RCOK on success. Returns `OC_EndOfScan` if no more entries exist (i.e., `next` is called after the last item in the scan has already been returned). Any other return code indicates an error condition.

`bool is_open()` indicates whether the scan is open or closed. Note that the scan remains open after retrieving the last entry.

`shrc rc()` returns the return code generated by the last operation (`open` or `next`).

`int operator==(shrc rc)`

`int operator!=(shrc rc)` These operators provide a convenient way to check the status of the scan. For example, applications can say `if (scan == RCOK) ...`.

`shrc DirScan::close()` closes the scan object. Closing a scan object releases any resources associated with the scan in both the server and client processes (such as the scan buffer). It also allows the scan object to be reused. The scan is closed implicitly when the DirScan object is destroyed (either via operator delete or when the DirScan object goes out of scope), or when the transaction in which the scan was opened terminates.

`~DirScan()`. The DirScan destructor closes the scan object if it is open.

3 A Directory Scan Example

The following function prints the name and loid of each entry in the directory given by the `pathname` argument and returns a count of entries.

```
int print_dir(const char *pathname) {
    DirEntry entry;
    int count = 0;

    // Open a scan over the pool given by "pathname."
    DirScan scan(pathname);

    // Make sure the scan was successfully opened.
    if (scan.rc() != RCOK) {
        cout << "Error scanning directory " << pathname << ": "
             << scan.rc() << endl;
        return 0;
    }

    // Scan until end-of-scan or an error is encountered.
    for(count = 0; scan.next(&entry) == RCOK; ++count)
        cout << entry.name << '\t' << entry.loid << endl;

    cout << pathname << " has " << count << " objects." << endl;

    // Check for errors.
    if(scan.rc().err_num() != SH_EndOfScan){
        cout << "Error scanning directory " << pathname << ": "
             << scan.rc() << endl;
        return 0;
    }

    // The destructor will close the scan object.
    return count;
}
```

A complete program using this function may be found in the `examples/shls` directory of the documentation release. Assuming the shell variable $SHROOT points to the root of a copy of the documentation release, you can compile and run the `shls` example as follows:

```
mkdir shls
cp $SHROOT/examples/shls/* shls
cd shls
make shls
shls
```

The results will be more interesting if you have something in your Shore file system, for example if you did not clean up after running the tests described in the tutorial *Getting Started with Shore*.