

**NAME**

sfile\_hdl\_base\_t, sfile\_read\_hdl\_t – File Descriptor I/O Handler Classes

**SYNOPSIS**

```

#include <sthread.h>

class sfile_hdl_base_t : public w_vbase_t {
public:
    enum { rd = 1, wr = 2, ex = 4 };
    enum { max = 64 };

    NORET          sfile_hdl_base_t(
        int         fd,
        int         mask);
    NORET          ~sfile_hdl_base_t();

    const int      fd;

    virtual void    read_ready() = 0;
    virtual void    write_ready() = 0;
    virtual void    expt_ready() = 0;

    void           enable();
    void           disable();

    static w_rc_t    wait(long timeout = sthread_base_t::WAIT_FOREVER);
    static void      dump(const char* str, ostream& out);

    static bool      is_active(int fd);
};

class sfile_read_hdl_t : public sfile_hdl_base_t {
public:
    NORET          sfile_read_hdl_t(int fd);
    NORET          ~sfile_read_hdl_t();

    w_rc_t          wait(long timeout);
    void            shutdown();
    bool            is_down() { return _shutdown; }
protected:
    // hide base::read_ready
    virtual void    read_ready();
};

class sfile_write_hdl_t : public sfile_hdl_base_t {
public:
    NORET          sfile_write_hdl_t(int fd);
    NORET          ~sfile_write_hdl_t();

    w_rc_t          wait(long timeout);
    void            shutdown();
    bool            is_down() { return _shutdown; }
protected:
    // hide base::write_ready

```

```

        virtual void        write_ready();
    };

```

## DESCRIPTION

File handlers are used in situations when a thread needs to wait for I/O on a unix file descriptor but does not want the operating system to suspend the whole process. File handlers provide a means with which a thread can wait for I/O without affecting other threads that are ready to run.

### Class `sfile_hdl_base_t`

Class `sfile_hdl_base_t` is an abstract base class for handling asynchronous file events. In general, users should not be concerned with this class. They should, instead, be instantiating more refined file handler classes such as `sfile_read_hdl_t`. see the implementation of `sfile_read_hdl_t` in `src/threads/threads.c`.

### `sfile_hdl_base_t(fd, mask)`

The constructor creates a file handler for the file descriptor *fd*. Parameter *mask* is a bitwise ORed value of the following flags:

<code>rd</code>	signifying read intention
<code>wr</code>	signifying write intention
<code>ex</code>	signifying exception intention

### `~sfile_hdl_base_t()`

### `enable()`

The **enable** method enables the file handler to be waited on when the thread package calls the **select** system call.

### `disable()`

The **disable** method disables the file handler from being waited on when the thread package calls the **select** system call.

### `wait()`

The **wait** method waits for some file handlers to be ready. An error is returned if *timeout* milliseconds elapsed before any file handler is ready. **Warning:** this method blocks the entire process on a unix **select** system call.

### `is_active(fd)`

The **is\_active** method returns **true** if a file handler exists for file descriptor *fd*.

### Class `sfile_read_hdl_t`

Class `sfile_read_hdl_t` inherits from `sfile_hdl_base_t` but handles only read events. It is used to block a thread that needs to wait for input on a file descriptor before proceeding. For example, a thread that processes user commands from stdin would create a `sfile_read_hdl_t` on file descriptor 0. The the EXAMPLES section for more details.

**sfile\_read\_hdl\_t(fd)**

The constructor creates a read-intention file handler on file descriptor *fd*.

**~sfile\_read\_hdl\_t()****shutdown()**

The **shutdown** method turns off monitoring of the file descriptor managed by the file handler. Any threads waiting on it, awakened with a **stBADFILEHDL** error code.

**wait()**

The **wait** method suspends the current thread, waiting to read from the file descriptor. The method returns timeout error if *timeout* milliseconds elapse before anything arrives on the file descriptor.

**Class sfile\_write\_hdl\_t**

Class **sfile\_write\_hdl\_t** inherits from **sfile\_hdl\_base\_t** but handles only write events. It is used to block a thread that needs to wait for a file descriptor to be ready for writing.

This class has only recently been implemented. No documentation is available yet. TODO

**ERRORS**

TODO.

**EXAMPLES**

```
stdin_thread_t::run()
{
    sfile_read_hdl_t h(0);    // handle on stdin
    char buf[256];
    for (;;) {
        if (h.wait()) {
            /* handle error */
            ...
        }
        /* stdin is ready -- read user command into buf */
        read(0, buf, sizeof(buf)-1);
        /* process user command */
        ...
    }
}
```

**VERSION**

This manual page applies to Version 2.0 of the Shore Storage Manager.

**SPONSORSHIP**

The Shore project is sponsored by the Advanced Research Project Agency, ARPA order number 018 (formerly 8230), monitored by the U.S. Army Research Laboratory under contract DAAB07-91-C-Q518. Further funding for this work was provided by DARPA through Rome Research Laboratory Contract No. F30602-97-2-0247.

**COPYRIGHT**

Copyright (c) 1994-1999, Computer Sciences Department, University of Wisconsin -- Madison. All Rights Reserved.

**SEE ALSO**

**errors(threads), threads\_t(threads), smutex\_t(threads), scond\_t(threads), sevsem\_t(threads), intro(threads).**