NAME

serial_t – Serial Number for Logical ID

SYNOPSIS

```
#include <serial_t.h>

struct serial_t {

    // The type of the only data member of serial_t is defined
    // elsewhere so that each level of the Shore software can
    // wrap the data member with its own definition, be it a
    // class, struct, or union.
    serial_t_data data;

public:

    serial_t( bool ondisk=true);
    serial_t(uint4 start, bool remote);
    serial_t(const serial_t& s);

    // return value true indicates overflow
    bool increment(uint4 amount); // also decrements

    bool is_remote()    const;
    bool is_local()     const;
    bool is_on_disk()   const;
    bool is_in_memory() const;

    bool is_null()    const;

    serial_t& operator=(const serial_t& t);

    operator==(const serial_t& s) const;
    operator!=(const serial_t& s) const;
    operator<=(const serial_t& s) const;
    operator<(const serial_t& s) const;
    operator>=(const serial_t& s) const;
    operator>(const serial_t& s) const;

    /* INPUT and OUTPUT */
    friend ostream& operator<<(ostream&, const serial_t& s);
    friend istream& operator>>(istream&, serial_t& s);

    friend istream& operator>>(istream&, serial_t_data& g);
    friend ostream& operator<<(ostream&, const serial_t_data& g);

    /* all of the following are in on-disk form: */
    static const serial_t max_local;
    static const serial_t max_remote;
    static const serial_t null;
};
```

DESCRIPTION

Class **serial_t** implements IDs that are unique to the volume containing them. See **lid_t(common)** for a description of volume IDs and **lid(ssm)** for information on how the SSM uses them. Serial numbers are currently 4 bytes long, but we plan to make them 8 bytes long in the future.

Two bits our of each serial number are reserved for indicating the type of the serial number. The high-order bit indicates if the serial number is *local,* indicating an intra-volume references, or *remote,* indicating an inter-volume references. The low order bit indicates if the serial number is in on-disk form or has been *swizzled* (ie. converted into in-memory form). Because of this, all un-swizzled serial numbers (the only kind the SSM understands) are odd numbers.

Constructors

Generally, value-added server writers do not need to construct serial numbers as this is done by SSM methods.

serial_t(start, remote)

This constructor generates a serial number *start* as the serial number. Actually, start is left-shifted one bit and the low order bit is set to true (on-disk). If *remote* is **true** the the high order bit is also set to mark the serial number as a remote reference.

Incrementing and Comparisons

increment(amount)

The **increment** method increments the serial number by amount (which may be negative). This is useful when an SSM routine returns a consecutive range of serial numbers by specifying the starting number and the size of the range. To enumerate the range, simply call **increment** once for each element in the range. The return value is **true** if an overflow occurs.

The comparison operators can only be used to compare serial numbers of the same type. For example, with serial numbers A and B, comparing **A < B** is incorrect if A is remote and B is local.

The **is_null** method is equivalent to **A == serial_t::null .**

Formatted I/O Methods

For 4-byte serial numbers, the input/output format is the format for an unsigned integer. For 8-byte serial numbers, stored as two integers, the format is 999.999.

Static Constants

There are a number of static constants.

max_local
Maximum value of a local serial number.

max_remote
Maximum value of a remote serial number.

null Null is a special value often used to represent an unknown or invalid serial number. The default constructor creates a serial number equivalent to null. The null serial number is local.

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

SEE ALSO

lid_t(common), lid(ssm).