SCSI

Specification Revision 0.9.1 (Aug 6, 1990)

A low-level SCSI driver class based on **sg** (4)

ClassName	SCSI
Superclass	Object
Category	Foundation
Other classes referenced	<none>
Version	1 (v1.1)
Maturity Index	Relatively mature
Requires Header Files	SCSI.h
Author	Jiro Nakamura
Organization	Canon, Inc. -- NeXT SE Div.
Created	June 12, 1990
Last Modified	August 6, 1990

CLASS DESCRIPTION

The SCSI class is intended to be used as the foundation for a variety of SCSI interface orientated devices. It provides the basic methods and structures to enable its descendent classes to provide SCSI driver support to their users. It is based on the **sg** (4) generic scanner driver.

Subclasses should first implement the methods that match the commands that their SCSI device supports. These methods should be named with the command name in lower-case as their prefix and `SCSI' in upper-case as their suffix. Therefore, the INQUIRY command becomes **inquirySCSI**. These methods should require that the SCSI device be previously opened.

Using these basic methods, subclasses should implement macro methods that wrap a number of (or even just one) commands into a suitable method call for a user. These macro methods should not have a SCSI suffix. The macro method should also open and close the SCSI driver automatically for the user. The only interface that the user should be provided with are the macro methods. The user should not have to use any methods with a -SCSI suffix or even know that the class is based on a SCSI driver. For example, the ScanSCSI class based on SCSI requires its users to use only three commands:

- (int) findDevice
- (int) initializeScanner
- (int) scanDocument::

All subclasses should implement the pseudo-basic method **findDeviceSCSI: (int)** ***trg.*** This method should return the SCSI target (ID) number of the first device starting from target *trg* on the SCSI bus that the class is designed to handle. The **findDevice** and **findDevice:** methods that are defined in this class both depend on the subclass to implement **findDeviceSCSI:**. See the method

description for **findDeviceSCSI:** for implementation hints

Methods should return 0 if they encounter no errors (command successful) and non-zero for errors. Positive numbers are reserved by the SCSI class. Subclasses should feel free to define and return negative numbers for their own unique errors. To provide more information about the error encountered, methods should fill the class variable *errorString* with the error type in English. Other objects can retrieve *errorString* through the **errorString** method.

Subclass authors may find it handy to reference the following documents:

the **sg** (4) manual page
the include file: <nextdev/scsireg.h>
the SCSI-1 and SCSI-2 ANSI specifications

The basic wrapping format for a SCSI command method is:

```
- myCommandSCSIwith: (struct myCommandStruct *) parameters
{
    // Set up pointer to command block within SCSI
    // request structure
    struct cdb_6 *cdbp = &sr.sr_cdb.cdb_c6;

    // Clear command block
    [self clearCommandBlock: (union cdb *) cdbp];

    // Set up command block, this is assuming your are using
    // 6 byte commands
    cdbp->c6_opcode       =  C6OP_MYCOMMAND;
    cdbp->c6_lun          = lun;
    cdbp->c6_len          = sizeof( *parameters);

    // Set up SCSI request DMA parameters
    sr.sr_dma_dir         = SR_DMA_WR;       // We are sending parameters
    sr.sr_addr            = (char *) parameters;  // from this address
    sr.sr_dma_max          = sizeof( *parameters);   // and so many bytes

    sr.sr_ioto   = 60;            // Timeout if no response for 60 seconds

    return  [self performSCSIRequest];   // Now do an SCSI i/o operation
                                         // with these settings
}
```

The basic wrapping format for a macro method is:

```
- macroMethod: (struct myCommandStruct *)  paramaters
{
    BOOL scsiWasOpen;             // Keeps track whether or not the driver
                                  // was open when we called this method
    int trg;                      // target number of SCSI device

    scsiWasOpen = scsiOpen;

    if(  !scsiOpen)
        if( [self openSCSI])
                    {
                    strcpy(errorString, ªCouldn't open SCSI driver.°);
                    return -1;
                    }

    if(  (trg = [self findMyDeviceSCSI]) == -1)  // Where is our device?
        {
        strcpy(errorString, ªI can't find MyDevice! Check its status...°);
        return -1;                              // I can't find it...
        }
```

```
    [self setTarget: trg   lun:0];    // We *must* set the target, lun before
                                      // using basic methods or else...

    [self  myCommandSCSI_1];                          // Do something
    [self  myCommandSCSIwith: parameter];             // Do something with a
                                                      // parameter

    if( !scsiWasOpen)                                 // If we opened the SCSI
           if( [self closeSCSI] )                     // driver, we should close
                       {                              // it
                       strcpy(errorString, ªCouldn't close SCSI driver.°);
                       return -1;
                       }

    return 0;                                         // Command successful
}
```

COPYRIGHTS

WARRANTY

MODIFICATIONS

Modifications should be returned to the author, Jiro Nakamura, to be incorporated into future versions of this class. He can be contacted at:

In the U.S.A (until June 1993):

Jiro Nakamura
c/o Reppy
493 Ellis Hollow Creek Rd.
Ithaca, NY 14850
USA

In Japan (indefinite):

Jiro Nakamura
2-15-3-702 Higashiyama
Meguro-ku, Tokyo 153
Japan

By E-mail (until June 1993):

ac6y@vax5.cit.cornell.edu (Domain based Internet)
uunet!vax5.cit.cornell.edu!ac6y (UUCP)

Through Canon:

Jiro Nakamura
NeXT SE Dept.
NeXT Computer Marketing Div.
Canon Inc.
Shin-Kawasaki Mitsui Bldg.
890-12, Kashimada
Saiwai-ku, Kawasaki-shi
Kanagawa 211
Japan

INSTANCE VARIABLES

Inherited from Object	Class	isa;
Declared in SCSI	BOOL	scsiOpen;
	int	target,
	lun;	
	char	*errorString[100];
	char	*dev_name;
	int	fd;
	struct	scsi_req sr;
	struct	scsi_adr sa;

scsiOpen	TRUE if the SCSI device is open, FALSE otherwise.
target	Target (SCSI ID) number for all SCSI requests.
lun	Logical unit of the target SCSI device
fd	File descriptor (see **open** (3)) for the generic SCSI device (see **sg** (4)).
sr	generic SCSI driver request structure (see **sg** (4) for more information and <nextdev/scsireg.h> for the structure definition).
sa	generic SCSI driver target/lun address change structure (see **sg** (4) for more information and <nextdev/scsireg.h> for the structure definition).

METHOD TYPES

Opening, closing the driver	openSCSI openSCSIAt: lun: closeSCSI
Setting the target, lun	setTargetSCSI: lun:

Basic SCSI Methods	testUnitReadySCSI requestSenseSCSI: inquirySCSI: readCapacitySCSI:
Pseudo-basic SCSI Methods	findDeviceSCSI: (*subclass responsibility)*
Macro User Methods	inquiry: readCapacity: findDevice findDevice:

Error processing	errorString
Variable accesser methods	statusReq scsiOpen
Private methods	performSCSIRequest; clearCommandBlock: (union cdb *) cdb

clearCommandBlock:

- clearCommandBlock: (union cdb *) *cdbp*

This zeroes out all the bits in *cdbp* in preparation for a command request. The union structure *cdb* is defined in <nextdev/scsireg.h>. This is a private method which should only be used by subclasses of SCSI.

closeSCSI

- (int) closeSCSI

This closes the SCSI generic driver and is required in order to release the generic SCSI driver so that other programs can use it. **closeSCSI** should be closed as soon as all immediate commands to the device have been sent.

errorString

- (char *) errorString

This returns the class variable *errorString* which holds information (in ASCII format) about the last error. It is the responsibility of subclasses of SCSI to put the proper information in *errorString* when an error occurs. (Said in a grave voice.)

findDevice

- (int) findDevice

This returns the target number of the first SCSI device that the subclass can handle. It is actually a wrap for [self findDevice: 0].

findDevice:

- (int) findDevice: (int) *trg*

This is a macro wrap for the subclass define **findDeviceSCSI:** pseudo-basic method. It returns the target number of the first SCSI device starting with *trg* that the subclass can handle. Users can request more information about the particular device with the **inquiry:** command.

findDeviceSCSI:

- (int) findDeviceSCSI: (int) *trg*

Subclasses are mandatorily responsible for implementing this command. It should return the target number of the first SCSI device starting from target number *trg* that the subclass can handle. The subclass method code should look something like:

```
- (int)  findDeviceSCSI: (int) trg
    {
    int tmp;
    struct inquiry_reply ibuffer;
```

```
for( tmp = trg; tmp < 8; tmp ++)
    {
    if( [self setTarget: tmp lun: 0])     // If no device at tmp
            continue;                     // go on to next target
    [self inquirySCSI: &ibuffer];
    if( ibuffer.ir_devicetype == DEVTYPE_MYDEVICE &&
            strncmp(ibuffer.ir_vendorid, MYVENDOR, MYVENDORLEN)==0)
            return tmp;     // We can handle this, return target
            }
return -1;  // Can't find our device, return -1
}
```

Macro method **findDevice:** wraps the above pseudo-basic method(so-called because it has the -SCSI suffix required for basic commands but isn't implemented by the device itself) with **openSCSI** and **closeSCSI** for the user.

inquiry:
inquirySCSI:

- (int) inquiry: (struct inquiry_reply *) *inquiry_buffer*
- (int) inquirySCSI: (struct inquiry_reply *) *inquiry_buffer*

This implements Mandatory SCSI command ªInquiryº (command group 0, code 0x12). It returns the inquiry data in *inquiry_buffer,* which is a structure defined in <nextdev/scsireg.h>.

openSCSI

- (int) openSCSI

This opens the SCSI generic driver and is required before any calls to basic command methods (recognizable by their SCSI suffix). The SCSI driver should be held open for as short a time as possible to prevent conflicts with other programs that use it. A subclass can tell whether the driver is open or not by the *scsiOpen* boolean variable.

openSCSIAt: lun:

- (int) openSCSIAt: (int) *trg* lun: (int) *ln*

This wraps the **openSCSI** and **setTargetSCSI: lun:** commands into one command.

performSCSIRequest

- (int) performSCSIRequest

This calls the generic SCSI driver with the parameters as set in class variable *sr*. This is a private method which should only be implemented by subclasses of SCSI which wish to make SCSI driver command requests.

readCapacity:
readCapacitySCSI:

- (int) readCapacity: (struct readCapacity_reply *) *reply_buffer*
- (int) readCapacitySCSI: (struct readCapacity_reply *)

reply_buffer
This implements Required SCSI command ªRead Capacityº (command group 1, code 0x25) for direct-access devices. It returns read capacity data (logical unit capacity and block length) in *reply_buffer,* which is a structure defined in <*SCSI.h*>.

requestSense:
requestSenseSCSI:
- (int) requestSense: (struct esense_reply *) *reply_buffer*
- (int) requestSenseSCSI: (struct esense_reply *) *reply_buffer*

This implements Mandatory SCSI command ªRequest Senseº (command group 0, code 0x03). It returns the sense data in *reply_buffer,* which is a structure defined in <nextdev/scsireg.h>.

scsiOpen
- (BOOL) scsiOpen

Returns TRUE if the driver is open and accessible, FALSE otherwise.

setTargetSCSI: lun:
- (int) setTargetSCSI: (int) *trg* lun: (int) *ln*

This tells the generic SCSI driver to address all subsequent SCSI request to the device at target (SCSI ID#) *trg*. Logical unit number *ln* is remembered in class variable *lun*, but it is the role of the implementation method to pass this information on when making the SCSI request.

statusReq
- (struct scsi_req *) statusReq

This returns the status of the last SCSI request as a structure defined in <nextdev/scsireg.h>.

testUnitReady
testUnitReadySCSI
- (int) testUnitReady
- (int) testUnitReadySCSI

This implements Mandatory SCSI command ªTest Unit Readyº (command group 0, code 0x00). Returns 0 if no error (unit ready), non-zero otherwise.