Document: T10/98-183 revision 0
Date: 98-09-01
To: NCITS T10 SCSI Working Group
From: Keith W. Parker <diogenes@europa.com> (503)255-1035
Subject: SSS & IP packets over SCSI

# SSS & IP packets over SCSI

## Purpose of this document

This document (with revisions over time) will contain all necessary information from the SSS draft standard to implement Internet IP packets over SCSI.

The majority of this document is to be included in the draft standard for
"SCSI Socket Services (SSS) Command Set" (NCITS Project T10/1246-D) as the clauses describing the
Clause 2 - "References",
Clause 4 - "SSS General Model",
Clause 5 - "Model for transport of SSS packets over SCSI",
Clause 7 - "Model for transport of Internet IP packets over SCSI",
Clause 8 - "Data Structures for SSS packets over SCSI",
Clause 11 -"SCSI Commands for Internet IP packets over SCSI", &
Clause 12 - "Data Structures for Internet IP packets over SCSI".

This is being presented as a separate document for 2 reasons:
1.) there is an immediate interest and need for transporting Internet IP packets over SCSI, and
2.) even though the implementation of "Internet IP packets over SCSI" is independent of the "Socket API RPC over SCSI" portion of the standard, many people/organizations are intimidated the sheer mass of the total document.

## Socket API RPC over SCSI
**vs.**
## Internet IP packets over SCSI

These are two methods of accomplishing the same objective: providing a high performance (i.e. SCSI bandwidth) Platform/Device Independent (PDI) "Socket" Application Programming Interface (API).

The "**Socket Application Programming Interface (API) Remote Procedure Call (RPC)**" method couples systems together at the **TOP of the Protocol Stack (PS)**. This method has the highest performance potential since it does not use the TCP/IP protocol stack and a Socket API call requires only two (2) SCSI command executions. The down side is that writing the driver for a specific platform is much more complex. Fortunately, this only has to be done once. An additional strong point is superior manageability. This is the method of choice for **purists**.

The "**Internet IP packets over SCSI**" method couples systems together at the **BOTTOM of the Protocol Stack (PS)**. This method has lower performance potential since it uses the TCP/IP protocol stack and each packet transported requires a SCSI command execution (and there may be many packets transported per Socket API call). The up side is that writing the driver for a specific platform is much less complex. However, this only has to be done once. This is the method of choice for **pragmatists**.

*Editor's Note: As my e-mail name "Diogenes" suggests, I have a strong tendency for the purist position. As this document suggests, I also have an appreciation for the pragmatic.*

## Revision History

### 1998-09-01 - "98-183r0" - First release.

This is the first release.
It is intended to be used to begin computer science class discussion.
It is not intended that code will be written to implement this 1st release.
The next release is to be on 1998-10-01.
The 1998-10-01 2nd release will be usable for a first prototype implementation specification.

This document is located at:
"SSS & IP packets over SCSI" T10 document T10/98-183r0
http://www.symbios.com/t10/io/t10/document.98/98-183r0.pdf
ftp://ftp.symbios.com/pub/standards/io/t10/document.98/98-183r0.pdf

The appropriate clauses from this document have been pasted into the first release of the SSS draft standard located at:
"SCSI Socket Services (SSS) Command Set" NCITS project T10/1246-D
http://www.symbios.com/t10/io/t10/drafts/sss/
http://www.symbios.com/t10/io/t10/drafts/sss/sss-r00.pdf
ftp://ftp.symbios.com/pub/standards/io/t10/drafts/sss/sss-r00.pdf
ftp://ftp.symbios.com/pub/standards/io/t10/drafts/sss/

*Editor's Note: The SCSI command numbers used in this release are TENTATIVE!*
*They will not be final until acted upon by the T10 committee.*

## Implementation

There may be factors other than throughput, manageability, or ease of implementation in deciding which method to implement.

Often the design objective is maximum throughput and manageability indicating the Socket API over SCSI RPC is most appropriate. Certain situations (such as using SCSI as an expansion bus for bridges, routers, hubs, or switches) are intrinsically IP packet oriented and would logically use the "Internet IP packets over SCSI" method.

Not only is it possible, it is desirable for a device to support both the "Socket API RPC over SCSI" method and the "Internet IP packets over SCSI" method as alternate/redundant paths to the same application. Many products would be more useful if they could be accessed with the same network applications over SCSI and/or other networks such as Ethernet or the Internet. If the device already has a TCP/IP protocol stack it takes very little effort to implement Internet IP packets over SCSI.

*Editor's Note: Any implementation plans presented here are not to become part of the SSS standard. They are merely the editor's suggestions.*

The general implementation plan is to **first** implement the **pragmatic** "Internet IP packets over SCSI" to get a functional system operational and
**then** implement the **purist** "Socket API RPC over SCSI"
to get the maximum in performance and manageability.

**1.) Barest, minimalist IP over SCSI system** - Unbuffered (one packet transferred per SCSI command) "IP over SCSI" accessing default channel only. Only basic packet transfer functions are to be implemented. No management functions are to be implemented. Everything is "default".

Check out a senior design project implemented on the Linux operating system:
"IP Encapsulation in SCSI Driver" by Randy Scott and Chris Frantz (based on RFC 2143)
http://www.wwa.com/~scottr/sd/
http://www.wwa.com/~scottr/sd/design/scsinet-2.1.25.tar.gz

"Encapsulating IP using SCSI" by Ben Elliston, Linux Journal, August 1998, issue 52
RFC 2143 "IP Encapsulation over the Small Computer Systems Interface"
ftp://ftp.internic.net/rfc/rfc2143.txt
ftp://ds.internic.net/rfc/rfc2143.txt

*Note: Both examples use SCSI command 2Ah in a non-standard manner which conflicts with some SCSI device types (such as disk drives), preventing their use as a functionality extension.*
*Since SSS SCSI commands are to be officially defined as OPTIONAL for ALL device types, it is not necessary (or desirable) to avoid accessing standard SCSI devices such as disk drives to avoid conflicts.*
*Since official SCSI command numbers are being assigned,*
*any device that does not support SSS will simply respond with an "Invalid Command" error code.*

**2.) Buffered IP over SCSI system** - Add buffering for transferring multiple packets per SCSI command. This improves performance potential by significantly reducing the arbitration/selection/command overhead (which is dominant in situations of high bandwidth with short data phases). It also improves performance by allowing a single system context switch to handle many IP packets at once.

**3.) Managed IP over SCSI system** - Add system management functions. Allow accessing channels and functions other than "default".

**4.) SCSI-NIC - SCSI Network Interface Card (NIC)** - (Optional) As an "SCSI Old Computer Trick", implement a fully functional prototype of a single-chip SCSI Network Interface Card (SCSI-NIC).
This may be 802.3 Ethernet, Fibre Channel, Token Ring, or whatever.
Add as many network interfaces as desired by simply plugging more SCSI-NICs onto the SCSI bus(es).
A SCSI-NIC may have any type/number of network ports.

**5.) "Socket API RPC over SCSI"** - Now that there is a successful build of
coupling systems together at the "bottom" of the Protocol Stack (PS),
the code can be extended to couple systems together at the "top" of the Protocol Stack.

**6.) SCSI Beowulf Linux Cluster** -
Use a Beowulf Linux Cluster (currently based on 100 mbps Ethernet) to compare performance between 100 mbps Ethernet, SSS Internet IP packets over SCSI, and SSS Socket API RPC over SCSI.

**Beowulf Consortium**
http://cesdis.gsfc.nasa.gov/beowulf/consortium/consortium.html

"Beowulf is a project to produce parallel Linux clusters from off-the-shelf hardware and freely available software and make the system trivially replicatable. The purpose of the Consortium is to facilitate communication within this confederation of independent groups and aid in the rapid deployment of this technology. Originally the "Beowulf Consortium" was considered in advocacy group. However, the rate at which Beowulf clusters have spread as across the community has been so high that there has been little need for such a group. To be honest, independent of the efforts of the "Consortium", Beowulf clusters now exist or are being built at many sites over the world. "

**Beowulf Project at CESDIS**
Center of Excellence in Space Data and Information Sciences
Universities Space Research Association
http://cesdis.gsfc.nasa.gov/linux/beowulf/

**Beowulf Linux Clusters**
http://cesdis.gsfc.nasa.gov/linux/beowulf/beowulf.html

**7.) SCSI Harbor Processor & Peripheral Modules** -

The SCSI Trade Association (STA) SCSI Harbor Project is currently defining an industry standard for hot-plug SCSI modules. The driving force (for today) is to support 3.5" disk drives in storage servers. The same standard can also be used for implementing hot-pluggable SCSI Harbor Processor Modules and SCSI Harbor Peripheral Modules. Entire systems can be built with these sub-system level hot-pluggable modules, eventually leading to the extinction of the "Stegosaurs" PC configuration.

http://www.scsita.org/
http://www.scsita.org/harbor/

**Keith W. Parker** <diogenes@europa.com>
Portland, Oregon, USA 1-(503)-255-1035
Technical Editor, NCITS Project T10/1246-D
SCSI Socket Services (SSS) Command Set

Diogenes' Unofficial SCSI Socket Services (SSS) web site:
http://www.europa.com/~diogenes/SSS/

## SCSI Socket Services (SSS) Command Set draft standard summary

These are the clauses of the SCSI Socket Services draft standard. The clauses (1, 6, 9, 10, & 13), relating exclusively to the SCSI Remote Procedure Call (RPC) commands and the Socket API Remote Procedure Set (RPS), are not included in this document (since the purpose of this document is to facilitate the implementation of Internet IP packets over SCSI).

**Clause 1 Scope**
*Editor's note: Only a dummy clause for numbering is included in this document.*

**Clause 2 References**
The normative and informative references that apply to this proposed standard.

**Clause 3 Definitions**
Describes the definitions, symbols and abbreviations used in this proposed standard.

**Clause 4 SSS General Model**
This clause provides an overview of the SSS general models.
This clause also specifies the conventions used throughout the proposed standard.

**Clause 5 Model for transport of SSS packets over SCSI**
Describes the transport of SSS packets (RPC & IP) over SCSI.

**Clause 6 Model for transport of RPC packets over SCSI**
*Editor's note: Only a dummy clause for numbering is included in this document.*

**Clause 7 Model for transport of Internet IP packets over SCSI**
Describes the specifics of IP (and other) packets over SCSI.

**Clause 8 Data structures for SSS packets over SCSI**
Describes data structure common to all SSS commands.

**Clause 9 SCSI commands for RPC packets over SCSI**
Describes SSS_RPC_ commands to transport Remote Procedure Call (RPC) request and reply.
*Editor's note: Only a dummy clause for numbering is included in this document.*

**Clause 10 Data structures for RPC packets over SCSI**
Describes data structures unique to SSS_RPC_ commands.
*Editor's note: Only a dummy clause for numbering is included in this document.*

**Clause 11 SCSI commands for Internet IP packets over SCSI**
Describes SSS_PKT_ commands to transport Internet IP (and other) packets.

**Clause 12 Data structures for Internet IP packets over SCSI**
Describes data structures unique to SSS_PKT_ commands.

**Clause 13 SSS Socket API Remote Procedure Set (RPS)**
*Editor's note: Only a dummy clause for numbering is included in this document.*

The rest of this document is excerpted/pasted from/to the proposed standard "SCSI Socket Services (SSS) Command Set", NCITS project T10/1246-D.

**AMERICAN NATIONAL STANDARD (DRAFT) 199N x3.NNN**

**American National Standard for Information Systems - (DRAFT)**

**Information Technology -**

# SCSI Socket Services (SSS) Command Set

## 1. Scope

*Editor's note: Dummy clause to make numbers match.*

# 2. References

## *2.1 Normative References*

The following standards contain provisions that, through reference in the text, constitute provisions of this standard. At the time of publication, the editions indicated were valid. All standards are subject to revision, and parties to agreements based on this standard are encouraged to investigate the possibility of applying the most recent editions of the standards listed below.

Copies of the following documents can be obtained from ANSI: Approved ANSI standards, approved and draft international and regional standards (ISO, IEC, CEN/CENELEC, ITUT), and approved and draft standards of other countries (including BSI, JIS, and DIN). For further information, contact ANSI Customer Service Department at 212-642-4900 (telephone), 212-302-1286 (fax) or via the World Wide Web at http://www.ansi.org.

Additional availability contact information is provided below as needed.

### 2.1.1 Approved references

SCSI-3 Architecture Model, X3.270-1997

SCSI-3 Primary Commands, X3.301-1997

SCSI-3 Stream Commands (SSC)
{Date: 1998/08/05, Rev: 12, Status: NCITS Approval, Project: 0997-D}

Portable Operating System Interface (POSIX) - Part xx: Protocol Independent Interfaces (PII) (IEEE Std. 1003.1g: 199?)

Portable Operating System Interfaces for Computer Environments (IEEE Std. 1003.1: 1990)

### 2.1.2 References under development

At the time of publication, the following referenced standard was still under development. For information on the current status of the document, or regarding availability, contact the relevant standards body as indicated.

Note 1: For more information on the current status of the document, contact the X3 Secretariat at 202-737-8888 (telephone), 202-638-4922 (fax) or via Email at x3sec@itic.nw.dc.us. To obtain copies of this document, contact Global Engineering at 15 Inverness Way East Englewood, CO 80112-5704 at 800-854-7179 (telephone), 303-792-2181 (telephone), or 303-792-2192 (fax).

## 2.2 Informative references

Small Computer System Interface-2 [X3.131-1994] [ISO/IEC 9316-1:1996]

RFC-1831: RPC: Remote Procedure Call Protocol Specification 2
ftp://ds.internic.net/rfc/rfc1831.txt

RFC-1832: XDR: External Data Representation Standard
ftp://ds.internic.net/rfc/rfc1832.txt

RFC-1833: Binding Protocols for ONC RPC Version 2
ftp://ds.internic.net/rfc/rfc1833.txt

"Windows Sockets 2 Application Programming Interface", Rev. 2.2.2, 97-08-07
ftp://ftp.microsoft.com/bussys/winsock/winsock2/WSAPI22.DOC

"Windows Sockets 2 Protocol-Specific Annex", Rev. 2.0.3, 96-05-10
ftp://ftp.microsoft.com/bussys/winsock/winsock2/wsanx203.doc

"Windows Sockets 2 Service Provider Interface" Rev. 2.2.2 97-08-07
ftp://ftp.microsoft.com/bussys/winsock/winsock2/WSSPI22.DOC

"Windows Generic QOS Mapping (Draft)" Ver. 2.9, 09/11/97 5:19PM
ftp://ftp.microsoft.com/bussys/winsock/winsock2/wsgqos.doc

"UNIX Network Programming - Volume 1, Networking APIs: Sockets and XTI", Second Edition
by W. Richard Stevens, Prentice-Hall, ISBN 0-13-490012-X

"Linux Kernel Internals" by M Beck, H Bohme, M Dziadzka, U Kuntz, R Magnus, D Verworner
Addison-Wesley, ISBN 0-201-87741-4

"Power Programming with RPC" by John Bloomer
O'Reilly & Associates, Inc., ISBN 0-937175-77-3

NCITS (National Committee on Information Technology Standards)
http://www.ncits.org/

NCITS Technical Committee T10 (SCSI) Home Page
http://www.symbios.com/t10/

SCSI Trade Association (STA), SCSI Harbor project (hot-pluggable SCSI modules)
http://www.scsita.org/
http://www.scsita.org/harbor/

Diogenes' Unofficial SCSI Socket Services (SSS) web site:
http://www.europa.com/~diogenes/SSS/
Keith W. Parker, Technical Editor, NCITS project T10/1246-D

# 3. Definitions, symbols and abbreviations

*Editor's note: Dummy clause to make numbers match.*

# 4. SSS General Model

Clause 4 provides an overview of the socket device **extension** class and the **extension** command set. This clause also specifies the conventions used throughout the standard.

This standard is intended to be used in conjunction with the SCSI-3 Architecture Model (SAM) standard and with the[X3.270] the SCSI-3 Primary Command Set (SPC).

## *4.1 Accessing/Controlling SCSI Devices with standard network applications*

The purpose of the SSS SCSI Command Extension Set is to allow SCSI devices to be accessed/controlled with standard network applications in a Platform/Device Independent (PDI) manner.

### 4.1.1 Standard network protocols and applications

The ideal of the SSS model is that, once a single piece of system software (the Socket API to SSS SCSI driver) has been installed for a given platform, there would be no need for any platform specific drivers or applications to access/control any conceivable functionality in a SCSI device with SSS extensions.

Instead of writing platform specific applications to run on various platforms, the developer would write interactive web pages to be served up by the device and configure how standard applications such as FTP, NFS, SMB, & e-mail respond to requests.

#### 4.1.1.1 User Interface

The primary user interface to SSS extended SCSI devices is the web-browser.

- HTTP
- CGI forms
- JAVA
- VRML
- RealAudio, RealVideo

Any other desired network application may be used.

- IRC Chat

#### 4.1.1.2 Bulk data transfer

Bulk data transfer would generally be implemented using any desired network file server protocol.

- FTP (File Transfer Protocol),
- NFS (Network File System),
- SMB (Server Message Block - Windows & OS/2)

#### 4.1.1.3 Event notification

- e-mail
- e-mail relay to wireless pager

### 4.1.2 Sample applications of SSS

#### 4.1.2.1 Adding functionality to existing SCSI devices

##### 4.1.2.1.1 Adding "File Drive" services to a "Disk Drive" (SBC & RBC)

For example, a SCSI disk drive may be used with standard device drivers for a SCSI Block device.

The SSS SCSI command extension set may be used to allow the user (with a web browser) to configure the disk drive to appear to be several smaller disk/file drives on separate LUNs.

These smaller virtual disk/file drives may then be made read-only (to defend from a virus attack) and/or appear to be an FTP (File Transfer Protocol) device to be accessed with a standard network FTP application and/or appear to be an NFS (Network File System) device to be accessed with a standard network NFS driver and/or appear to be an SMB (Server Message Block - Windows & OS/2) server.

##### 4.1.2.1.2 Adding web-browser user interface to a Medium Changer (SMC)

##### 4.1.2.1.3 Adding web-browser user interface to a Multimedia (MMC) device

##### 4.1.2.1.4 Adding web-browser user interface to a RAID Controller (SCC-2)

##### 4.1.2.1.5 Adding web-browser user interface to a Enclosure Services (SES) device

##### 4.1.2.1.6 Adding web-browser user interface to a Stream (SSC) device

### 4.1.2.2 SCSI Peripheral Modules

#### 4.1.2.2.1 Advantages

*4.1.2.2.1.1 Consistent user interface (web-browser) across all platforms*

*4.1.2.2.1.2 Platform/Device Independent*

*4.1.2.2.1.3 Single design works with all platforms*

*4.1.2.2.1.4 Eliminates need for platform/device specific drivers & applications*

*4.1.2.2.1.5 Enclosed module instead of exposed circuit boards*

*4.1.2.2.1.6 SCSI Peripheral Module has processing power needed for application*

- Does not place a processing burden on the host processor

*4.1.2.2.1.7 May continue operation while host is powered down*

- Modem / e-Mail server / Ethernet / FAX / Voice Mail telephone line support device

#### 4.1.2.2.2 Examples

*4.1.2.2.2.1 Modem / e-Mail server / Ethernet / FAX / Voice Mail telephone line support device*

- Provides network interface by modem (PPP) and/or Ethernet
- Personal e-mail server
- FAX reception / transmission / forwarding
- Voice Mail
- Continues operation while host system powered down
- 2x Data/FAX/Voice modems, 2x Ethernet, modest CPU, battery back-up/input, VGA/NTSC/PAL video in/out, Audio In/Out, Keyboard/Mouse ports, 2x USB ports

*4.1.2.2.2.2 SCSI SSS X-terminal*

- X-windows at SCSI bandwidth

*4.1.2.2.2.3 Audio-Video editing system*

- Gang operation for added channels

### 4.1.2.3 SCSI Processor Modules

Whether an SSS SCSI device would be considered a Processor Module or a Peripheral Module is determined by the relative power of the processor (and the application using it).

If the power of the CPU (and the application using it) is much more significant that the peripheral devices of the module, then it is a SCSI Processor Module. If the power of the CPU

#### 4.1.2.3.1 Advantages

*4.1.2.3.1.1 Only spend money on CPU, memory, and SCSI when upgrading*

*4.1.2.3.1.2 Enclosed Modules*

*4.1.2.3.1.3 Packaged like SCSI disk drives (with panel space for connectors)*

#### 4.1.2.3.2 Examples

*4.1.2.3.2.1 Upgrade system with SCSI Processor Module instead of new system of motherboard*

*4.1.2.3.2.2 Foundation for SCSI Application Modules*

*4.1.2.3.2.3 Embedded systems*

### 4.1.2.4 SCSI Application Modules

Once low cost SCSI Processor Modules are available it would be financially viable to change from supporting a large and complex software in somebody else's computer to porting the entire application into a SCSI Processor Module and use standard network applications at SCSI bandwidth.

#### 4.1.2.4.1 Advantages

There are many advantages to the SCSI Application Module configuration. While any single advantage may not be sufficient to justify the change, taken together they will.

##### *4.1.2.4.1.1 Connects to any host platform*

- Multiple platform support with a single product

##### *4.1.2.4.1.2 Select CPU for best Performance / Cost ratio*

- No legacy compatibility expense/issues
- Pick best instruction set for application

##### *4.1.2.4.1.3 Security of application*

- Module physically sealed with tamper switches

##### *4.1.2.4.1.4 Total control of application configuration*

- No user files inside SCSI Application Module
- No application files installed in user's system

##### *4.1.2.4.1.5 Reduce customer support expenses on configuration issues*

- User can not modify files inside SCSI Application Module

##### *4.1.2.4.1.6 Reduce Bootlegging*

- Alternate CPU reduces potential

##### *4.1.2.4.1.7 Transaction based billing possible*

#### 4.1.2.4.2 Examples

##### *4.1.2.4.2.1 CAD design system*

##### *4.1.2.4.2.2 Database system*

##### *4.1.2.4.2.3 Audio-Video Editing system*

##### *4.1.2.4.2.4 Full immersion virtual reality user interface system*

### 4.1.2.5 SCSI Modules replace Stegosaurus PC configuration

At the beginning of the 20th century it was acceptable that if you owned an automobile you would be either "mechanically inclined", or wealthy enough to hire someone who was "mechanically inclined". Today the vast majority of automobile users expect to turn a key, push the pedal, and go.

The traditional Stegosaurus PC configuration of a motherboard and register level plug-in adapter boards (with equivalent operating system specific drivers and applications) is a highly tuned version of the early 20th century automobile in requiring technical expertise.

The vast majority of the end user population needs a system no more complex than inserting a video cassette into a VCR.

Fully enclosed, hot-pluggable SCSI modules (with SCSI Socket Services (SSS) providing a Platform/Device Independent (PDI) network style interface) are a solution.

#### 4.1.2.5.1 SCSI as Corpus Callosum

- Platform/Device Independent

#### 4.1.2.5.2 SCSI Expansion boxes replace current PC packaging

- Chassis, power supply and ventilation for hot-pluggable SCSI Harbor modules
- SCSI backplane with automatic active terminators and external connectors at each end
- NO motherboard

#### 4.1.2.5.3 SCSI Trade Association (STA) SCSI Harbor project

- Hot-pluggable SCSI modules

http://www.scsita.org/
http://www.scsita.org/harbor/

#### 4.1.2.5.4 Advantages

- End user friendly
- Mix and match operating systems and processors

#### 4.1.2.5.5 Examples

### *4.2 Socket API use of SSS RPC and Packets*

4.2.1 Socket API via RPC

4.2.2 Socket API via IP packets

### *4.3 Alternate use of SSS RPC and Packets*

4.3.1 Alternate use of SSS RPC

4.3.2 Alternate use of SSS Packets

## *4.4 SSS SCSI Command Extension Set for all SCSI device types*

The SSS SCSI commands are **optional** for **all** SCSI device types. They may be an **optional extended** mechanism for accessing/controlling the (extended) functionality of a SCSI device **or** they may be used as the **only** way of accessing/controlling the functionality of a SCSI device.

They are not to have any effect on the normal operation of the SCSI commands for a given SCSI type device.

### 4.4.1 Non-interference with existing SCSI device types operation

Non-interference with existing SCSI device types operation

## *4.5 Platform/Device Independent (PDI) operation*

The implementation of SSS is to be Platform/Device Independent (PDI).

## *4.6 Initiator Mode Only operation*

The SSS command set allows either symmetrical communication between two Target Mode capable SCSI host adapters or asymmetrical communication between a Target Mode SCSI device and an Initiator Only SCSI device.

This is to accommodate the very large number of SCSI Host Bus Adapters (HBA) that are not supported by a SCSI Target Mode Applications Programming Interface (API). Virtually all personal computer SCSI host bus adapters do not have Target Mode API support.

## *4.7 Receiver Makes Right Endian*

The standard SSS_PKT_HDR_t header wrapping each SSS packet indicates the "endianness" of each packet on a packet by packet basis.

## *4.8 SSS SCSI Command Groups*

There are two major groups of commands. One major group supports the "Socket (or other) API over SCSI RPC" and the other major supports "Internet IP (or other) Packets over SCSI".

Within each major group there are two sub-groups. One sub-group supports the management functions (i.e. housekeeping etc.) and the other sub-group supports the actual transfer of packets of information/commands.

Within each sub-group there is a _GET and a _PUT command. These complementary commands support the ability to have symmetrical communications even when one of the two communicating SCSI devices is capable of only SCSI Initiator functions.

### 4.8.1 SSS SCSI command group summary

#### 4.8.1.1 SSS_RPC_xxxx_xxx - Remote Procedure Call (RPC)

**SSS_RPC_xxxx_xxx** - These commands support the SSS Remote Procedure Call (RPC) mechanism. This connects systems together at the "top" of the protocol stack. This method has the highest performance/management potential since it does not use the TCP/IP protocol stack. It is the most complex to implement.

While the primary focus is Socket API RPC, these commands can also be used to support API RPCs other than the Socket API RPC.

#### 4.8.1.2 SSS_PKT_xxxx_xxx - Packet transport (PKT)

**SSS_PKT_xxxx_xxx** - These commands support the SSS "IP Packets over SCSI" mechanism. This is an alternate method of accomplishing the same objective: providing a high performance (i.e. SCSI bandwidth) Platform/Device Independent (PDI) "Socket" Application Programming Interface (API). This connects systems together at the "bottom" of the protocol stack. This method is much easier to implement if a system already has a TCP/IP protocol stack.
It also addresses the issue of situations that are intrinsically packet oriented such as using SCSI as an expansion bus for bridges, routers, hubs and switches or as an intermediate link in a data stream that has already been broken down into packets.

While the primary focus is Internet IP Packets (IPv4 & IPv6), other packets (802.3, USB, IrDA, etc.) may be transported.

*Editor's Note: This is new to the SSS project. It an extension for the sake of pragmatism, it is much easier (therefore more likely) to implement.*

#### 4.8.1.3 SSS_xxx_MGMT_xxx - Management commands

**SSS_xxx_MGMT_xxx** - These commands support communication management functions. These are handled separately from the transfer (_XFER_) functions because they may be relatively large and infrequently used so it is desirable for them to be unloaded from memory when not needed.

#### 4.8.1.4 SSS_xxx_XFER_xxx - Transfer commands

**SSS_xxx_XFER_xxx** - These commands support the actual transfer of data. These are more compact and often used so they need to be left in memory.

#### 4.8.1.5 SSS_xxx_xxxx_GET & SSS_xxx_xxxx_PUT -

**SSS_xxx_xxxx_GET** & **SSS_xxx_xxxx_PUT** - These commands GET and PUT packets into the transport buffers. They allow symmetric communications even when one of the SCSI devices is "Initiator Only" capable.

## *4.9 SCSI Peripheral Device Types*

The SSS packet commands are optional to all SCSI device types and may be added as an **extension** to **any** SCSI device type. This allows any type of device to be extended so that its functionality can be accessed with standard network tools such as web browsers, FTP, and e-mail without effecting the device's normal operation.

### 4.9.1 Unknown or No Device Type peripheral device type (code 1Fh)

If a SCSI device does not need to implement a particular SCSI device type, it may return a value of 1Fh (Unknown or no device type) in the "Peripheral device type" field of the "standard INQUIRY data" of the INQUIRY command (SPC-2 clause 7.5.1 Standard INQUIRY data).

### 4.9.2 SCSI commands Common to all SCSI device types

Commands common to all SCSI device types are described in
Clause 5 "Model common to all device types"
of "SCSI Primary Commands - 2 (SPC-2)" (NCITS project T10/1236-D).

#### 4.9.2.1 SCSI commands Mandatory for all SCSI device types

**Table ?.? - SCSI Commands Mandatory for All Device Types**

| Code | Size | Type | SCSI Primary Command | SPC clause | |
|---|---|---|---|---|---|
| 00h | 6 | Man. | TEST UNIT READY | 7.24 | |
| 03h | 6 | Man. | REQUEST SENSE | 7.20 | |
| 12h | 6 | Man. | INQUIRY | 7.5 | |
| 1Dh | 6 | Man. | SEND DIAGNOSTIC | 7.23 | |

#### 4.9.2.2 SCSI commands Optional for all SCSI device types

**Table ?.? - SCSI Commands Optional for Processor AND Communication Device Types**

| Code | Size | Type | SCSI Primary Command | SPC clause | |
|---|---|---|---|---|---|
| 1Ch | 6 | Opt. | RECEIVE DIAGNOSTIC RESULTS | 7.16 | |
| 3Bh | 10 | Opt. | WRITE BUFFER | 7.25 | |
| 3Ch | 10 | Opt. | READ BUFFER | 7.15 | |
| 40h | 10 | Opt. | CHANGE DEFFINITION | 7.1 | |
| 4Ch | 10 | Opt. | LOG SELECT | 7.6 | |
| 4Dh | 10 | Opt. | LOG SENSE | 7.7 | |
| A0h | 12 | Opt. | REPORT LUNS | 7.19 | |

### 4.9.3 Processor Device peripheral device type (code 03h)

Processor Devices are described in
Clause 6 "Model for processor devices",
Clause 9 "Commands for processor type devices", and
Clause 10 "Parameters for processor type devices"
of "SCSI Primary Commands - 2 (SPC-2)" (NCITS project T10/1236-D).

**Table ?.? - SCSI Commands Mandatory for Processor Device Type**

| Code | Size | Type | SCSI Primary Command | SPC clause | |
|---|---|---|---|---|---|
| 0Ah | 6 | Man. | SEND(06) | 9.2 | shared |
| 1Dh | 6 | Man. | SEND DIAGNOSTIC | 7.23 | |

### 4.9.4 Communication Device peripheral device type (code 09h)

Communication Devices are described in
Clause 7 "Communication devices"
of "SCSI-3 Stream Device Commands (SSC)" (NCITS project T10/997-D).

**Table ?.? - SCSI Commands Mandatory for Communication Device Type**

| Code | Size | Type | SCSI Stream Command | SSC clause | |
|---|---|---|---|---|---|
| 0Ah | 6 | Man. | SEND MESSAGE(06) | 7.3.4 | shared |
| 1Dh | 6 | Man. | SEND DIAGNOSTIC | SPC 7.23 | |

### 4.9.5 Other peripheral device types

SSS may be used with any other SCSI device types without conflict.

Other devices are described in
of SAM

## *4.10 SSS SCSI commands*

SCSI Group Code #4 (16 byte) commands
OPTIONAL for ALL device types

```
#define SSS_RPC_MGMT_GET        0x90
#define SSS_RPC_MGMT_PUT        0x91
#define SSS_RPC_XFER_GET        0x92
#define SSS_RPC_XFER_PUT        0x93

#define SSS_PKT_MGMT_GET        0x94
#define SSS_PKT_MGMT_PUT        0x95
#define SSS_PKT_XFER_GET        0x96
#define SSS_PKT_XFER_PUT        0x97
```

*Editor's Note: The SCSI command numbers used in this release are TENTATIVE! They will not be final until acted upon by the T10 committee.*

### 4.10.1 SSS commands DCB fields

All commands use the exact same format with the exception that the SSS_Pkt_Count and SSS_Data_len fields specify the actual amounts to be transferred during SSS_xxx_xxxx_PUT commands and the maximum amounts that can be transferred during the SSS_xxx_xxxx_GET commands.

| Byte | Bits | Type | Field Name |
|---|---|---|---|
| 0 | 8 | BE_uchar | SSS_SCSI_Cmd_Num |
| 1 | 8 | BE_uchar | SSS_Func_Code |
| 2-3 | 16 | BE_short | SSS_Pkt_Count |
| 4-7 | 32 | BE_long | SSS_Data_Len |
| 8-11 | 32 | BE_long | SSS_Cmd_Key |
| 12-13 | 16 | BE_short | SSS_Channel_Token |
| 14 | 8 | BE_uchar | SSS_Func_Flags |
| 15 | 8 | BE_uchar | SSS_Control( SAM r18 5.1, 5.6 ; SAM-2 r5a 5.1.2, 5.6 ) |

#### 4.10.1.1 SSS_SCSI_Cmd_Num

A constant with one of the following values:

```
#define SSS_RPC_MGMT_GET        0x90
#define SSS_RPC_MGMT_PUT        0x91
#define SSS_RPC_XFER_GET        0x92
#define SSS_RPC_XFER_PUT        0x93

#define SSS_PKT_MGMT_GET        0x94
#define SSS_PKT_MGMT_PUT        0x95
#define SSS_PKT_XFER_GET        0x96
#define SSS_PKT_XFER_PUT        0x97
```

#### 4.10.1.2 SSS_Func_Code

The command function code with a value defined by the enumeration SSS_PKT_TYPE_e in header file SSS_.H (Clause 8).

#### 4.10.1.3 SSS_Pkt_Count

The number of packets to be transferred in this command. During the SSS_xxx_xxxx_PUT commands this field indicates the actual number of packets to be transfered. During the SSS_xxx_xxxx_GET commands this field indicates the maximum number of packets that can be accepted.

#### 4.10.1.4 SSS_Data_Len

The number of data bytes to be transfered in this command. During the SSS_xxx_xxxx_PUT commands this field indicates the actual number of bytes to be transfered. During the SSS_xxx_xxxx_GET commands this field indicates the maximum number of bytes that can be accepted.

#### 4.10.1.5 SSS_Cmd_Key

A tracking/security field for this command.

#### 4.10.1.6 SSS_Channel_Token

A tracking/security field for this command.

#### 4.10.1.7 SSS_Func_Flags

Flags specific to the command number in the CDB field SSS_Func_Code.

#### 4.10.1.8 SSS_Control

The standard control byte as defined in the SCSI Architecture Model ( SAM r18 5.1, 5.6 ; SAM-2 r5a 5.1.2, 5.6 )

## *4.11 SSS data structures and function prototype*

Several POSIX C language data structures are defined so that from implementation to implementation the key data structures controlling system operation will be familiar.

Three POSIX C header files are defined. The file `sss_.h` is to be included in all implementations of SSS drivers. The file `sss_rpc.h` is to be included in implementations of SSS drivers that support Remote Procedure Call (RPC) over SCSI. The file `sss_pkt.h` is to be included in implementations of SSS drivers that support Internet IP packets over SCSI.

### 4.11.1 SSS header file sss_.h (Clause 8)

The file `sss_.h` (Clause 8)is to be included in all implementations of SSS drivers.

#### 4.11.1.1 typedef struct SSS_CDB16_BE_t

typedef struct SSS_CDB16_BE_t
This data structure is a big-endian representation of the 16 byte Command Descriptor Block (CDB) to be copied directly into the SCSI CDB. NOTE: Beware of possible padding by the compiler.

#### 4.11.1.2 typedef struct SSS_CDB16_t

typedef struct SSS_CDB16_t
This data structure is a host-endian representation of the 16 byte Command Descriptor Block (CDB) to be used for driver control.

#### 4.11.1.3 typedef enum SSS_PKT_TYPE_e

typedef enum SSS_PKT_TYPE_e
This is an enumeration of the SSS packet types to be used in the `sss_pkt_type` field of the `SSS_PKT_HDR_t` structure use with all SSS packets.

#### 4.11.1.4 typedef struct SSS_PKT_HDR_t

typedef struct SSS_PKT_HDR_t
This is the `SSS_PKT_HDR_t` structure to be used with all SSS packets.

#### 4.11.1.5 int sss_pkt_hdr_check( SSS_PKT_HDR_t * );

int sss_pkt_hdr_check( SSS_PKT_HDR_t * );
This optional recommended function is used to validate/check SSS_PKT_HDT_t structures as a sanity check. It may also be used as a debugging/security hook.

### 4.11.2 SSS header file sss_rpc.h (Clause 10)

The file `sss_rpc.h` (Clause 10)is to be included in implementations of SSS drivers that support Remote Procedure Call (RPC) over SCSI.

#### 4.11.2.1 typedef enum SSS_RPC_MGMT_FUNC_CODE_e

typedef enum SSS_RPC_MGMT_FUNC_CODE_e

#### 4.11.2.2 typedef enum SSS_RPC_XFER_FUNC_CODE_e

typedef enum SSS_RPC_XFER_FUNC_CODE_e

### 4.11.3 SSS header file sss_pkt.h (Clause 12)

The file `sss_pkt.h` (Clause 12)is to be included in implementations of SSS drivers that support Internet IP packets over SCSI.

#### 4.11.3.1 typedef enum SSS_PKT_MGMT_FUNC_CODE_e

typedef enum SSS_PKT_MGMT_FUNC_CODE_e

#### 4.11.3.2 typedef enum SSS_PKT_XFER_FUNC_CODE_e

typedef enum SSS_RPC_PKT_FUNC_CODE_e

# 5. Clause ?? Model for transport of SSS packets over SCSI

This clause describes the model for transport of SSS packets over SCSI.

## *5.1 Initiator Only operation*

The SSS command set allows either symmetrical communication between two Target Mode capable SCSI host adapters or asymmetrical communication between a Target Mode SCSI device and an Initiator Only SCSI device.

## *5.2 Receiver Makes Right Endian*

The standard SSS_PKT_HDR_t header wrapping each packet indicated the "endianness" of each packet on a packet by packet basis.

The endian flag is the least significant bit of the sss_pkt_falgs field of the SSS_PKT_HDR_t structure. Even indicates "little-endian" and odd indicates "big-endian".

## *5.3 SSS SCSI Command Groups*

There are two major groups of commands. One major group supports the "Socket (or other) API over SCSI RPC" and the other major supports "Internet IP (or other) Packets over SCSI".

Within each major group there are two sub-groups. One sub-group supports the management functions (i.e. housekeeping etc.) and the other sub-group supports the actual transfer of packets of information/commands.

Within each sub-group there is a _GET and a _PUT command. These complementary commands support the ability to have symmetrical communications even when one of the two communicating SCSI devices is capable of only SCSI Initiator functions. This is to accommodate the very large number of SCSI Host Bus Adapters (HBA) that are not supported by a SCSI Target Mode Applications Programming Interface (API).

## *5.4 Transport of RPC packets (Connection Oriented)*

Delivery of RPC packets.

## *5.5 Transport of multiple packets (Connectionless)*

Delivery of Internet IP packets.
Best effort, non-guaranteed delivery of packets.

Multiple packets can be transferred with each SSS_PKT SCSI command to reduce the overhead of the arbitration/selection/command phases (which is dominant in situations of high bandwidth with short data phases).

Each SSS Packet wrapped with an SSS_PKT_HDR_t header data structure.

## *5.6 SSS Packet Transport between devices model*

The general model is that within both the client and server devices there are packet buffers for both outbound and inbound packets and that the SSS SCSI commands only transport packets from the outbound packet buffer of one device to the inbound packet buffer of the other device.

The higher levels of the device drivers are then notified that packets have been sent and received, as appropriate. Generally, the successful completion of a SSS SCSI command does not indicate that the packet has been processed successfully, only that it has been transported (to the next device) successfully.

A single SSS SCSI command transaction can transport one or more SSS packets. The ability to transport more than one SSS packet per command transaction reduces the overhead (both for the SCSI bus and the operating systems at each end) of the SCSI Arbitration/Selection/Command phases. This may not be very significant for very large packet sizes but it may be significant for smaller packet sizes.

One of the primary purposes of this model is to accommodate the very large number of SCSI Host Bus Adapters (HBA) that are not supported by a SCSI Target Mode Applications Programming Interface (API).

It makes no difference if the SSS packets have been transported with a SSS_xxx_xxxx_PUT command or a SSS_xx_xxxx_GET command. From the perspective of the higher levels of the device drivers the result is the same.

### 5.6.1 Driver operation

#### 5.6.1.1 Incoming

The Interrupt Service Routine (ISR) moves packets into the packet buffers from the SCSI bus.

The ISR then notifies the driver demon with a semaphore (or other mechanism).

The driver demon becomes ready and begins execution to process the received data.

#### 5.6.1.2 Outgoing

The driver demon moves packets into the packet buffers.

The driver demon then notified the ISR with a semaphore (or other mechanism).

The ISR becomes ready and moves the data from the packet buffers to the SCSI bus.

### 5.6.2 Transfer of single packets without header

The disadvantage to this general model is that there may be situations where is desirable to transport packets to or from an application thread's data space or that the successful completion indication for the SCSI command actually indicates that the packet has been processed successfully.

This is easily accomplished by using a `SSS_Func_Code` that supports only a single SSS packet to be transport and may optionally imply successful processing of the packet with an indication of a successful completion of the SSS SCSI command.

# 6. Clause ?? Model for RPC over SCSI

SSS_RPC_MGMT_GET - SCSI Socket Services Packet Management Get
SSS_RPC_MGMT_PUT - SCSI Socket Services Packet Management Put
SSS_RPC_XFER_GET - SCSI Socket Services Packet Transfer Get
SSS_RPC_XFER_PUT - SCSI Socket Services Packet Transfer Put

SCSI Group Code #4 (16 byte) commands
OPTIONAL for ALL device types

# 7. Clause ?? Model for Internet IP packets over SCSI

SSS_PKT_MGMT_GET - SCSI Socket Services Packet Management Get
SSS_PKT_MGMT_PUT - SCSI Socket Services Packet Management Put
SSS_PKT_XFER_GET - SCSI Socket Services Packet Transfer Get
SSS_PKT_XFER_PUT - SCSI Socket Services Packet Transfer Put

SCSI Group Code #4 (16 byte) commands
OPTIONAL for ALL device types

# 8. Clause ?? Data structures for SSS packets over SCSI

This clause describes SCSI commands for SSS support of Internet IP packets over SCSI.

```
/******************** BEGINING OF FILE: sss_.h ******************/
/* sss_.h */

/*
*  Clause ?? Data Structures for SSS support of SSS over SCSI
*  NCITS Project T10/1246-D "SCSI Socket Services (SSS) Command Set"
*
*  This clause is presented as a POSIX C compliant header file defining
*  data structures required to implement "SSS_ over SCSI".
*/
/*********************************************************************/
/*
*   SCSI Socket Services (SSS) default Command Descriptor Block (CDB)
*
*   Common for ALL SSS commands unless a special CDB is implied
*   by the SSS_Func_Code in the 2nd byte of all SSS CDBs.
*/

typedef struct
    SSS_CDB16_BE_t           /* NOTE: THIS IS BIG-ENDIAN */
        {
        BE_uchar  SSS_SCSI_Cmd_Num ;
        BE_uchar  SSS_Func_Code ;
        BE_short  SSS_Pkt_Count ;
        BE_long   SSS_Data_Len ;
        BE_long   SSS_Cmd_Key ;
        BE_short  SSS_Channel_Token ;
        BE_uchar  SSS_Func_Flags ;
        BE_uchar  SSS_Control ;/*SAM r18 5.1, 5.6; SAM-2 r5a 5.1.2, 5.6*/
        } ;

typedef struct
    SSS_CDB16_t             /* NOTE: THIS IS HOST-ENDIAN */
        {
        uchar  SSS_SCSI_Cmd_Num ;
        uchar  SSS_Func_Code ;
        short  SSS_Pkt_Count ;
        long   SSS_Data_Len ;
        long   SSS_Cmd_Key ;
        short  SSS_Channel_Token ;
        uchar  SSS_Func_Flags ;
        uchar  SSS_Control ;/*SAM r18 5.1, 5.6; SAM-2 r5a 5.1.2, 5.6*/
        } ;

/*
* SSS_SCSI_Cmd_Num  = SCSI Command Number = SSS_PKT_PUT = 0x??
* SSS_Func_Code     = Function Code to perform on data-out
* SSS_Pkt_Count     = Number of packets sending
* SSS_Data_Len      = Number of bytes sending
* SSS_Cmd_Key       = Security / tracking Key for this command
* SSS_Channel_Token = Channel / Token value
* SSS_Func_Flags    = dependent on SSS_Func_Code value
* SSS_Control       = Control octet defined in SCSI Architecture Model
*                   ( SAM r18 5.1, 5.6; SAM-2 r5a 5.1.2, 5.6 )
*/
/*********************************************************************/
```

```
/*********************************************************************/
/*
 * This defines the values for the
 * "sss_pkt_type" field of struct SSS_PKT_HDR_t
 */

typedef enum
    SSS_PKT_TYPE_e
        {
        SSS_PKT_TYPEe_NULL    =  0, /* no data field */
        SSS_PKT_TYPEe_VERSION =  1, /* version & default info */
        SSS_PKT_TYPEe_IPv4    =  2, /* IPv4 packet */
        SSS_PKT_TYPEe_IPv6    =  3, /* IPv6 packet */
        SSS_PKT_TYPEe_802_3   =  4, /* EtherNet 802.3 packet */
        SSS_PKT_TYPEe_USB         , /* USB Universal Serial Bus */
        SSS_PKT_TYPEe_IRDA        , /* IRdA InfaRed */
        SSS_PKT_TYPEe_ENUM_END
        } ;

/*********************************************************************/
/*
 *  This header is common to all packets transferred with the
 *  SSS_ SCSI commands
 */

typedef struct
    SSS_PKT_HDR_t
        { /* ENDIAN FLAG must be LSB of first byte
           * or LSB of first field which must be 8-bits in size */
          /* even = little-endian, odd = big-endian */
        uchar               sss_pkt_flags; /* ENDIAN FLAG */
        uchar               sss_pkt_opt_1;
        uchar               sss_pkt_opt_2;
        uchar               sss_pkt_opt_3; /*32-bit alignment */
        long                sss_pkt_type;           /* SSS_PKT_TYPE_e */
        long                sss_pkt_type_version;
        long                sss_pkt_len;
        long                sss_pkt_hdr_len;
        long                sss_pkt_hdr_pad_len;
        long                sss_pkt_data_len;
        long                sss_pkt_data_pad_len;
        long                sss_pkt_dest;           /* 0 => default */
        long                sss_pkt_source;         /* 0 => default */
        } ;

/*********************************************************************/

int sss_pkt_hdr_check
        ( /* test redundant header values as a sanity check */
          /* may also be used for debugging hook           */
          /* may also be used for security hook            */
          /* if applied to all packets, discarding if error */
          /* if error discard and release memory/resources  */

        struct SSS_PKT_HDR_t * sss_pkt_hdr;

        ) ; /* return non-zero if error */

/* ??? need enumeration of SSS_PKT_HDR_CHK_ERR_ ??? */

/*********************************************************************/

/********************** END OF FILE: sss_.h **********************/
```

# 9. Clause ?? SCSI commands for RPC over SCSI

This clause describes SCSI commands for SSS support of Internet IP packets over SCSI.

SSS_RPC_MGMT_GET - SCSI Socket Services Packet Management Get
SSS_RPC_MGMT_PUT - SCSI Socket Services Packet Management Put
SSS_RPC_XFER_GET - SCSI Socket Services Packet Transfer Get
SSS_RPC_XFER_PUT - SCSI Socket Services Packet Transfer Put

SCSI Group Code #4 (16 byte) commands
OPTIONAL for ALL device types

## *9.1 SSS_RPC_MGMT_GET*

SCSI command SSS_RPC_MGMT_GET "SSS RPC Management Get"

| Byte | Bits | Type | |
|---|---|---|---|
| 0 | 8 | BE_uchar | SSS_SCSI_Cmd_Num = SSS_RPC_MGMT_GET = 0x90 ?? |
| 1 | 8 | BE_uchar | SSS_Func_Code |
| 2-3 | 16 | BE_short | SSS_Pkt_Count_max |
| 4-7 | 32 | BE_long | SSS_Data_Len_max |
| 8-11 | 32 | BE_long | SSS_Cmd_Key |
| 12-13 | 16 | BE_short | SSS_Channel_Token |
| 14 | 8 | BE_uchar | SSS_Func_Flags |
| 15 | 8 | BE_uchar | Control ( SAM r18 5.1, 5.6 ; SAM-2 r5a 5.1, 5.6 ) |

```
Byte  |Bits| Type     | Field Name
------+----+----------+--------------------------
 0    |  8 | BE_uchar | SSS_SCSI_Cmd_Num
 1    |  8 | BE_uchar | SSS_Func_Code
 2-3  | 16 | BE_short | SSS_Pkt_Count
 4-7  | 32 | BE_long  | SSS_Data_Len
 8-11 | 32 | BE_long  | SSS_Cmd_Key
12-13 | 16 | BE_short | SSS_Channel_Token
14    |  8 | BE_uchar | SSS_Func_Flags
15    |  8 | BE_uchar | Control( SAM r18 5.1, 5.6 ; SAM-2 r5a 5.1.2, 5.6 )
```

| | |
|---|---|
| SCSI_Cmd_Num | SSS_RPC_MGMT_GET = 0x90 ?? |
| Func_Code | Function Code |
| Pkt_Count_max | Maximum Number of packets in data phase of command |
| Data_Len_max | Maximum Length (in bytes) of data-in phase |
| Cmd_Key | Security / Tracking key for this command |
| Channel_ Token | Channel Number or Token |
| Func_Flags | Func_Code specific flags |
| Control | Control byte defined in SCSI Architecture Model<br>SAM r18 sections 5.1, 5.6 ; SAM-2 r5a sections 5.1, 5.6 |

### *9.2 SSS_RPC_MGMT_PUT*

SCSI command SSS_RPC_MGMT_PUT "SSS RPC Management Put"

| Byte | Bits | Type | |
|---|---|---|---|
| 0 | 8 | BE_uchar | SCSI_Cmd_Num = SSS_RPC_MGMT_PUT = 0x91 ?? |
| 1 | 8 | BE_uchar | Func_Code |
| 2-3 | 16 | BE_short | Pkt_Count |
| 4-7 | 32 | BE_long | Data_Len |
| 8-11 | 32 | BE_long | Cmd_Key |
| 12-13 | 16 | BE_short | Channel_Token |
| 14 | 8 | BE_uchar | Func_Flags |
| 15 | 8 | BE_uchar | Control ( SAM r18 5.1, 5.6 ; SAM-2 r5a 5.1, 5.6 ) |

```
Byte  |Bits| Type     | Field Name
------+----+----------+--------------------------
 0    |  8 | BE_uchar | SSS_SCSI_Cmd_Num
 1    |  8 | BE_uchar | SSS_Func_Code
 2-3  | 16 | BE_short | SSS_Pkt_Count
 4-7  | 32 | BE_long  | SSS_Data_Len
 8-11 | 32 | BE_long  | SSS_Cmd_Key
12-13 | 16 | BE_short | SSS_Channel_Token
14    |  8 | BE_uchar | SSS_Func_Flags
15    |  8 | BE_uchar | Control( SAM r18 5.1, 5.6 ; SAM-2 r5a 5.1.2, 5.6 )
```

| | |
|---|---|
| SCSI_Cmd_Num | SSS_RPC_MGMT_PUT = 0x92 ?? |
| Func_Code | Function Code |
| Pkt_Count | Number of packets in data phase of command |
| Data_Len | Length (in bytes) of data-out phase |
| Cmd_Key | Security / Tracking key for this command |
| Channel _Token | Channel Number or Token |
| Func_Flags | Func_Code specific flags |
| Control | Control byte defined in SCSI Architecture Model<br>SAM r18 sections 5.1, 5.6 ; SAM-2 r5a sections 5.1, 5.6 |

### *9.3 SSS_RPC_XFER_GET*

SCSI command SSS_RPC_XFER_GET "SSS RPC Transfer Get"

| Byte | Bits | Type | |
|---|---|---|---|
| 0 | 8 | BE_uchar | SSS_SCSI_Cmd_Num = SSS_RPC_XFER_GET = 0x92 ?? |
| 1 | 8 | BE_uchar | SSS_Func_Code |
| 2-3 | 16 | BE_short | SSS_Pkt_Count_max |
| 4-7 | 32 | BE_long | SSS_Data_Len_max |
| 8-11 | 32 | BE_long | SSS_Cmd_Key |
| 12-13 | 16 | BE_short | SSS_Channel_Token |
| 14 | 8 | BE_uchar | SSS_Func_Flags |
| 15 | 8 | BE_uchar | Control ( SAM r18 5.1, 5.6 ; SAM-2 r5a 5.1, 5.6 ) |

```
Byte  |Bits| Type     | Field Name
------+----+----------+--------------------------
 0    |  8 | BE_uchar | SSS_SCSI_Cmd_Num
 1    |  8 | BE_uchar | SSS_Func_Code
 2-3  | 16 | BE_short | SSS_Pkt_Count
 4-7  | 32 | BE_long  | SSS_Data_Len
 8-11 | 32 | BE_long  | SSS_Cmd_Key
12-13 | 16 | BE_short | SSS_Channel_Token
14    |  8 | BE_uchar | SSS_Func_Flags
15    |  8 | BE_uchar | Control( SAM r18 5.1, 5.6 ; SAM-2 r5a 5.1.2, 5.6 )
```

| | |
|---|---|
| SCSI_Cmd_Num | SSS_RPC_XFER_GET = 0x92 ?? |
| Func_Code | Function Code |
| Pkt_Count_max | Maximum Number of packets in data phase of command |
| Data_Len_max | Maximum Length (in bytes) of data-in phase |
| Cmd_Key | Security / Tracking key for this command |
| Channel_ Token | Channel Number or Token |
| Func_Flags | Func_Code specific flags |
| Control | Control byte defined in SCSI Architecture Model<br>SAM r18 sections 5.1, 5.6 ; SAM-2 r5a sections 5.1, 5.6 |

### *9.4 SSS_RPC_XFER_PUT*

SCSI command SSS_RPC_PUT "SSS RPC Transfer Put"

| Byte | Bits | Type | |
|---|---|---|---|
| 0 | 8 | BE_uchar | SCSI_Cmd_Num = SSS_RPC_XFER_PUT = 0x93 ?? |
| 1 | 8 | BE_uchar | Func_Code |
| 2-3 | 16 | BE_short | Pkt_Count |
| 4-7 | 32 | BE_long | Data_Len |
| 8-11 | 32 | BE_long | Cmd_Key |
| 12-13 | 16 | BE_short | Channel_Token |
| 14 | 8 | BE_uchar | Func_Flags |
| 15 | 8 | BE_uchar | Control ( SAM r18 5.1, 5.6 ; SAM-2 r5a 5.1, 5.6 ) |

```
Byte  |Bits| Type     | Field Name
------+----+----------+--------------------------
 0    |  8 | BE_uchar | SSS_SCSI_Cmd_Num
 1    |  8 | BE_uchar | SSS_Func_Code
 2-3  | 16 | BE_short | SSS_Pkt_Count
 4-7  | 32 | BE_long  | SSS_Data_Len
 8-11 | 32 | BE_long  | SSS_Cmd_Key
12-13 | 16 | BE_short | SSS_Channel_Token
14    |  8 | BE_uchar | SSS_Func_Flags
15    |  8 | BE_uchar | Control( SAM r18 5.1, 5.6 ; SAM-2 r5a 5.1.2, 5.6 )
```

| | |
|---|---|
| SCSI_Cmd_Num | SSS_RPC_XFER_PUT = 0x93 ?? |
| Func_Code | Function Code |
| Pkt_Count | Number of packets in data phase of command |
| Data_Len | Length (in bytes) of data-out phase |
| Cmd_Key | Security / Tracking key for this command |
| Channel _Token | Channel Number or Token |
| Func_Flags | Func_Code specific flags |
| Control | Control byte defined in SCSI Architecture Model<br>SAM r18 sections 5.1, 5.6 ; SAM-2 r5a sections 5.1, 5.6 |

# 10. Clause ?? Data structures for RPC over SCSI

This clause describes SCSI commands for SSS support of Internet IP packets over SCSI.

```
/******************** BEGINING OF FILE: sss_rpc.h ******************/
/* sss_rpc.h */

/*
*  Clause ?? Data Structures for SSS support of RPC over SCSI
*  NCITS Project T10/1246-D "SCSI Socket Services (SSS) Command Set"
*
*  This clause is presented as a POSIX C compliant header file defining
*  data structures required to implement "RPC over SCSI".
*/

/*********************************************************************/

/*********************************************************************/
/*
* Common to SCSI commands SSS_RPC_MGMT_GET and SSS_RPC_MGMT_PUT
*
* These are the common defined values for the
* "SSS_Func_Code" fields of the CDB16 for
*  SCSI commands SSS_RPC_MGMT_GET and SSS_RPC_MGMT_PUT
*/

typedef enum                          /* SSS_RPC_MFCe_ */
    SSS_RPC_MGMT_FUNC_CODE_e
        { /* passed in SSS_Func_Code field of SS_PKT_XFER_xxx */
        SSS_RPC_MFCe_NOP        =  0, /* ignores pkts, always OK */
        SSS_RPC_MFCe_GET_ABORT  =  1, /*aborts disconnected SSS_RPC_MGMT_GET*/

        SSS_RPC_MFCe_VERS_GET   =  2, /* xfer version & defaults */
        SSS_RPC_MFCe_VERS_PUT   =  3, /* xfer version & defaults */

        SSS_RPC_MFCe_PKT_GET    =  4, /* xfer any packet types */
        SSS_RPC_MFCe_PKT_PUT    =  5, /* xfer any packet types */

        SSS_RPC_MFCe_ENUM_END
        } ;

/*********************************************************************/
```

```
/*********************************************************************/
/*
* Common to SCSI commands SSS_RPC_XFER_GET and SSS_RPC_XFER_PUT
*
* These are the common defined values for the
* "SSS_Func_Code" fields of the CDB16 for
*  SCSI commands SSS_RPC_XFER_GET and SSS_RPC_XFER_PUT
*/

typedef enum                          /* SSS_RPC_XFCe_ */
    SSS_RPC_XFER_FUNC_CODE_e
        { /* passed in SSS_Func_Code field of SS_PKT_XFER_xxx */
        SSS_RPC_XFCe_NOP        =  0, /* ignores pkts, always OK */
        SSS_RPC_XFCe_GET_ABORT  =  1, /*aborts disconnected SSS_PKT_XFER_GET*/

        SSS_RPC_XFCe_VERS_GET   =  2, /* xfer version & defaults */
        SSS_RPC_XFCe_VERS_PUT   =  3, /* xfer version & defaults */

        SSS_RPC_XFCe_PKT_GET    =  4, /* xfer any packet types */
        SSS_RPC_XFCe_PKT_PUT    =  5, /* xfer any packet types */

        SSS_RPC_XFCe_ENUM_END
        } ;

/*********************************************************************/


/********************** END OF FILE: sss_rpc.h **********************/
```

# 11. Clause ?? SCSI commands for Internet IP packets over SCSI

This clause describes SCSI commands for SSS support of Internet IP packets over SCSI.

SSS_PKT_MGMT_GET - SCSI Socket Services Packet Management Get
SSS_PKT_MGMT_PUT - SCSI Socket Services Packet Management Put
SSS_PKT_XFER_GET - SCSI Socket Services Packet Transfer Get
SSS_PKT_XFER_PUT - SCSI Socket Services Packet Transfer Put

SCSI Group Code #4 (16 byte) commands
OPTIONAL for ALL device types

### *11.1 SSS_PKT_MGMT_GET*

SCSI command SSS_PKT_MGMT_GET "SSS Packet Management Get"

| Byte | Bits | Type | |
|---|---|---|---|
| 0 | 8 | BE_uchar | SSS_SCSI_Cmd_Num = SSS_PKT_MGMT_GET = 0x94 ?? |
| 1 | 8 | BE_uchar | SSS_Func_Code |
| 2-3 | 16 | BE_short | SSS_Pkt_Count_max |
| 4-7 | 32 | BE_long | SSS_Data_Len_max |
| 8-11 | 32 | BE_long | SSS_Cmd_Key |
| 12-13 | 16 | BE_short | SSS_Channel_Token |
| 14 | 8 | BE_uchar | SSS_Func_Flags |
| 15 | 8 | BE_uchar | Control ( SAM r18 5.1, 5.6 ; SAM-2 r5a 5.1, 5.6 ) |

```
Byte  |Bits| Type     | Field Name
------+----+----------+--------------------------
 0    |  8 | BE_uchar | SSS_SCSI_Cmd_Num
 1    |  8 | BE_uchar | SSS_Func_Code
 2-3  | 16 | BE_short | SSS_Pkt_Count
 4-7  | 32 | BE_long  | SSS_Data_Len
 8-11 | 32 | BE_long  | SSS_Cmd_Key
12-13 | 16 | BE_short | SSS_Channel_Token
14    |  8 | BE_uchar | SSS_Func_Flags
15    |  8 | BE_uchar | Control( SAM r18 5.1, 5.6 ; SAM-2 r5a 5.1.2, 5.6 )
```

| | |
|---|---|
| SCSI_Cmd_Num | SSS_PKT_MGMT_GET = 0x94 ?? |
| Func_Code | Function Code |
| Pkt_Count_max | Maximum Number of packets in data phase of command |
| Data_Len_max | Maximum Length (in bytes) of data-in phase |
| Cmd_Key | Security / Tracking key for this command |
| Channel_ Token | Channel Number or Token |
| Func_Flags | Func_Code specific flags |
| Control | Control byte defined in SCSI Architecture Model<br>SAM r18 sections 5.1, 5.6 ; SAM-2 r5a sections 5.1, 5.6 |

### *11.2 SSS_PKT_MGMT_PUT*

SCSI command SSS_PKT_MGMT_PUT "SSS Packet Management Put"

| Byte | Bits | Type | |
|---|---|---|---|
| 0 | 8 | BE_uchar | SCSI_Cmd_Num = SSS_PKT_MGMT_PUT = 0x95 ?? |
| 1 | 8 | BE_uchar | Func_Code |
| 2-3 | 16 | BE_short | Pkt_Count |
| 4-7 | 32 | BE_long | Data_Len |
| 8-11 | 32 | BE_long | Cmd_Key |
| 12-13 | 16 | BE_short | Channel_Token |
| 14 | 8 | BE_uchar | Func_Flags |
| 15 | 8 | BE_uchar | Control ( SAM r18 5.1, 5.6 ; SAM-2 r5a 5.1, 5.6 ) |

```
Byte  |Bits| Type     | Field Name
------+----+----------+--------------------------
 0    |  8 | BE_uchar | SSS_SCSI_Cmd_Num
 1    |  8 | BE_uchar | SSS_Func_Code
 2-3  | 16 | BE_short | SSS_Pkt_Count
 4-7  | 32 | BE_long  | SSS_Data_Len
 8-11 | 32 | BE_long  | SSS_Cmd_Key
12-13 | 16 | BE_short | SSS_Channel_Token
14    |  8 | BE_uchar | SSS_Func_Flags
15    |  8 | BE_uchar | Control( SAM r18 5.1, 5.6 ; SAM-2 r5a 5.1.2, 5.6 )
```

| | |
|---|---|
| SCSI_Cmd_Num | SSS_PKT_MGMT_PUT = 0x95 ?? |
| Func_Code | Function Code |
| Pkt_Count | Number of packets in data phase of command |
| Data_Len | Length (in bytes) of data-out phase |
| Cmd_Key | Security / Tracking key for this command |
| Channel _Token | Channel Number or Token |
| Func_Flags | Func_Code specific flags |
| Control | Control byte defined in SCSI Architecture Model<br>SAM r18 sections 5.1, 5.6 ; SAM-2 r5a sections 5.1, 5.6 |

### *11.3 SSS_PKT_XFER_GET*

SCSI command SSS_PKT_XFER_GET "SSS Packet Transfer Get"

| Byte | Bits | Type | |
|---|---|---|---|
| 0 | 8 | BE_uchar | SSS_SCSI_Cmd_Num = SSS_PKT_XFER_GET = 0x96 ?? |
| 1 | 8 | BE_uchar | SSS_Func_Code |
| 2-3 | 16 | BE_short | SSS_Pkt_Count_max |
| 4-7 | 32 | BE_long | SSS_Data_Len_max |
| 8-11 | 32 | BE_long | SSS_Cmd_Key |
| 12-13 | 16 | BE_short | SSS_Channel_Token |
| 14 | 8 | BE_uchar | SSS_Func_Flags |
| 15 | 8 | BE_uchar | Control ( SAM r18 5.1, 5.6 ; SAM-2 r5a 5.1, 5.6 ) |

```
Byte  |Bits| Type     | Field Name
------+----+----------+--------------------------
 0    |  8 | BE_uchar | SSS_SCSI_Cmd_Num
 1    |  8 | BE_uchar | SSS_Func_Code
 2-3  | 16 | BE_short | SSS_Pkt_Count
 4-7  | 32 | BE_long  | SSS_Data_Len
 8-11 | 32 | BE_long  | SSS_Cmd_Key
12-13 | 16 | BE_short | SSS_Channel_Token
14    |  8 | BE_uchar | SSS_Func_Flags
15    |  8 | BE_uchar | Control( SAM r18 5.1, 5.6 ; SAM-2 r5a 5.1.2, 5.6 )
```

| | |
|---|---|
| SCSI_Cmd_Num | SSS_PKT_XFER_GET = 0x96 ?? |
| Func_Code | Function Code |
| Pkt_Count_max | Maximum Number of packets in data phase of command |
| Data_Len_max | Maximum Length (in bytes) of data-in phase |
| Cmd_Key | Security / Tracking key for this command |
| Channel_ Token | Channel Number or Token |
| Func_Flags | Func_Code specific flags |
| Control | Control byte defined in SCSI Architecture Model<br>SAM r18 sections 5.1, 5.6 ; SAM-2 r5a sections 5.1, 5.6 |

### *11.4 SSS_PKT_XFER_PUT*

SCSI command SSS_PKT_XFER_PUT "SSS Packet Transfer Put"

| Byte | Bits | Type | |
|---|---|---|---|
| 0 | 8 | BE_uchar | SCSI_Cmd_Num = SSS_PKT_XFER_PUT = 0x97 ?? |
| 1 | 8 | BE_uchar | Func_Code |
| 2-3 | 16 | BE_short | Pkt_Count |
| 4-7 | 32 | BE_long | Data_Len |
| 8-11 | 32 | BE_long | Cmd_Key |
| 12-13 | 16 | BE_short | Channel_Token |
| 14 | 8 | BE_uchar | Func_Flags |
| 15 | 8 | BE_uchar | Control ( SAM r18 5.1, 5.6 ; SAM-2 r5a 5.1, 5.6 ) |

```
Byte  |Bits| Type     | Field Name
------+----+----------+--------------------------
 0    |  8 | BE_uchar | SSS_SCSI_Cmd_Num
 1    |  8 | BE_uchar | SSS_Func_Code
 2-3  | 16 | BE_short | SSS_Pkt_Count
 4-7  | 32 | BE_long  | SSS_Data_Len
 8-11 | 32 | BE_long  | SSS_Cmd_Key
12-13 | 16 | BE_short | SSS_Channel_Token
14    |  8 | BE_uchar | SSS_Func_Flags
15    |  8 | BE_uchar | Control( SAM r18 5.1, 5.6 ; SAM-2 r5a 5.1.2, 5.6 )
```

| | |
|---|---|
| SCSI_Cmd_Num | SSS_PKT_XFER_PUT = 0x97 ?? |
| Func_Code | Function Code |
| Pkt_Count | Number of packets in data phase of command |
| Data_Len | Length (in bytes) of data-out phase |
| Cmd_Key | Security / Tracking key for this command |
| Channel _Token | Channel Number or Token |
| Func_Flags | Func_Code specific flags |
| Control | Control byte defined in SCSI Architecture Model<br>SAM r18 sections 5.1, 5.6 ; SAM-2 r5a sections 5.1, 5.6 |

# 12. Clause ?? Data Structures for Internet IP packets over SCSI

```
/******************** BEGINING OF FILE: sss_pkt.h ******************/
/* sss_pkt.h */

/*
*  Clause ?? Data Structures for SSS support of Internet IP packets over SCSI
*  NCITS Project T10/1246-D "SCSI Socket Services (SSS) Command Set"
*
*  This clause is presented as a POSIX C compliant header file defining
*  data structures required to implement "Internet IP packets over SCSI".
*/

/*********************************************************************/

/*********************************************************************/
/*
* Common to SCSI commands SSS_PKT_MGMT_GET and SSS_PKT_MGMT_PUT
*
* These are the common defined values for the
* "SSS_Func_Code" fields of the CDB16 for
*  SCSI commands SSS_PKT_MGMT_GET and SSS_PKT_MGMT_PUT
*/

typedef enum
    SSS_PKT_MGMT_FUNC_CODE_e         /* SSS_PKT_MFCe_ */
        { /* passed in SSS_Func_Code field of SS_PKT_XFER_xxx */
        SSS_PKT_MFCe_NOP        =  0, /* ignores pkts, always OK */
        SSS_PKT_MFCe_GET_ABORT  =  1,/*aborts disconnected SSS_PKT_MGMT_GET */

        SSS_PKT_MFCe_VERS_GET   =  2, /* xfer version & defaults */
        SSS_PKT_MFCe_VERS_PUT   =  3, /* xfer version & defaults */

        SSS_PKT_MFCe_PKT_GET    =  4, /* xfer any packet types */
        SSS_PKT_MFCe_PKT_PUT    =  5, /* xfer any packet types */

        SSS_PKT_MFCe_ENUM_END
        } ;

/*********************************************************************/
```

```
/*********************************************************************/
/*
* Common to SCSI commands SSS_PKT_XFER_GET and SSS_PKT_XFER_PUT
*
* These are the common defined values for the
* "SSS_Func_Code" fields of the CDB16 for
*  SCSI commands SSS_PKT_XFER_GET and SSS_PKT_XFER_PUT
*/

typedef enum                          /* SSS_PKT_XFCe_ */
    SSS_PKT_XFER_FUNC_CODE_e
        { /* passed in SSS_Func_Code field of SS_PKT_XFER_xxx */
        SSS_PKT_XFCe_NOP       =  0, /* ignores pkts, always OK */
        SSS_PKT_XFCe_GET_ABORT =  1,/* aborts disconnected SSS_PKT_XFER_GET */

        SSS_PKT_XFCe_VERS_GET  =  2, /* xfer version & defaults */
        SSS_PKT_XFCe_VERS_PUT  =  3, /* xfer version & defaults */

        SSS_PKT_XFCe_PKT_GET   =  4, /* xfer any packet types */
        SSS_PKT_XFCe_PKT_PUT   =  5, /* xfer any packet types */

        SSS_PKT_XFCe_IP_ANY_GET     =  6, /* xfer any IP packets    */
        SSS_PKT_XFCe_IP_ANY_PUT     =  7, /* xfer any IP packets    */

        SSS_PKT_XFCe_IP_IPv4_GET    =  8, /* xfer IPv4 packets only */
        SSS_PKT_XFCe_IP_IPv4_PUT    =  9, /* xfer IPv4 packets only */

        SSS_PKT_XFCe_IP_IPv6_GET    = 10, /* xfer IPv6 packets only */
        SSS_PKT_XFCe_IP_IPv6_PUT    = 11, /* xfer IPv6 packets only */

        SSS_PKT_XFCe_ENUM_END
        } ;

/*********************************************************************/

/********************** END OF FILE: sss_pkt.h **********************/
```

# 13. SSS Socket API Remote Procedure Set (RPS)

*Editor's note: Dummy clause to make numbers match.*