iSCSI Boot on SPARC Functional Specification

(FWARC 2008/466)

1 Version History

1.0 Revised 11-June-2009

1.1 Revised 12-June-2009

1.2 Revised 21-June-2009

1.3 Revised 9-July-2009

1.4 Revised 10-July-2009

1.5 Revised 13-July-2009

1.6 Revised 13-July-2009

1.7 Revised 13-July-2009

1.8 Revised 15-September-2009

2 Project Description

This provides the OpenBoot code necessary to allow booting Solaris from an iSCSI target.

The code is provided as a package included in the existing network boot package (/packages/obp-tftp), thus offering the iSCSI boot capability to any network adapter which currently provides ordinary tftp boot capability. This slightly differs from the x64 implementation, in that the network card on SPARC is not itself required to contain iSCSI specifics – other than the network driver, all code resides in the platform OBP.

2.1 Architecture Summary

The iSCSI package establishes a TCP/IP connection with the remote target, logs in and authenticates itself, and then provides a conduit for SCSI commands and responses. The existing OBP generic disk driver is used to provide a file system interface. The TCP/IP stack is re-used from the previous project of Wanboot (FWARC 2004/461).

Due to the different semantics of Network devices and Disk devices, the boot command will specify a network device with a series of options specifying the remote target, which will be passed internally to */iscsi-hba/disk*, a device which exports disk device semantics. Secondary booters and Solaris will see a */chosen:bootpath* property specifying */iscsi-hba/disk*, and use that to boot. Solaris must look at other */chosen* properties to determine the underlying location of the root partition just booted.

The target may also be specified in a DHCP *Root Path* option, see RFC 4173. This simplifies the operator's task in only needing to specify “boot net:dhcp” rather than typing the rather long set of

details to identify an iSCSI disk. The specifics are documented in §4 (DHCP Interface) below.

3 Bindings

3.1 obp-tftp package modifications, affecting network nodes

The package *obp-tftp* is modified to add support to iSCSI boot, in addition to the current tftpboot and wanboot. This primarily involves providing information parsed from device arguments and opening an interface to tcp/ip for iSCSI's use. The interface to tcp/ip is private to iSCSI, not intended to be used by other applications.

3.1.1 obp-tftp Methods

The following new methods are specific to iSCSI support, and are not normally even visible to users. These are documented here simply because they are presented as *external* from *obp-tftp*. These methods all presume only one connection exists per device instance so there is no need for references indicating which connection is being addressed.

3.1.1.1 iscsi-connect (--)

Establishes the TCP connection with the remote target. The information needed to determine where to connect has been provided on the network-device arguments or obtained from DHCP.

3.1.1.2 iscsi-read (addr len -- actual)

Performs a TCP read from the remote target over the connection opened by *iscsi_connect*.

3.1.1.3 iscsi-write (addr len -- actual)

Performs a TCP write to the remote target over the connection opened by *iscsi_connect.*

3.1.1.4 iscsi-disconnect (--)

Breaks the TCP connection and reclaims resources allocated.

3.1.2 obp-tftp Device Arguments

iSCSI support in the obp-tftp package supports a series of device argument keywords to identify the destination iSCSI target, following the “keyword=value” format described in FWARC 2002/461 and FWARC 2004/579.

A network device path with these keywords may result in a tremendously long boot command. Note that the command line (including expanded “*net*” devalias) may not exceed 256 characters. Should arguments exceed that length, the arguments must be placed in the *network-boot-arguments* NVRAM variable.

All of the below arguments (or the values provided by DHCP or internal defaults) will be stored as

properties in */chosen*, see §3.2.1 (/chosen Properties) below.

3.1.2.1 iscsi-target-ip

Dotted-decimal formatted IP addresss (e.g., 255.255.255.255) of remote iSCSI target.

3.1.2.2 iscsi-target-name

iSCSI qualified name of desired disk target, as described in RFC 3720 §3.2.6.3 (iSCSI Name Structure). This will usually be a string in the form “iqn.1986-03.com.sun:02:...”.

3.1.2.3 iscsi-port

Optional decimal formatted integer from 1 to 65535, representing the TCP port number of the remote iSCSI implementation. This will default to 3260 if not specified.

3.1.2.4 iscsi-partition

Optional string specifying bootable partition on the target. Defaults to null string if not specified. The secondary booter will later take a null string to mean partition “a”.

3.1.2.5 iscsi-lun

Optional hexadecimal dash-separated field encoding the 64-bit LUN of the remote target. Will default to zero if not specified.

In its trivial manifestation (lun number smaller than 65535), this is simply a hexadecimal encoding. For larger values, the formatting of this 64-bit field is complex. We quote in its entirety the paragraph from RFC 4173 §5 describing how this field must be formatted:

The "LUN" field is a hexadecimal representation of the LU number. If the LUN field is blank, then LUN 0 is assumed. If the LUN field is not blank, the representation MUST be divided into four groups of four hexadecimal digits, separated by "-". Digits above 9 may be either lower or upper case. An example of such a representation would be 4752-3A4F-6b7e-2F99. For the sake of brevity, at most three leading zero ("0") digits MAY be omitted in any group of hexadecimal digits. Thus, the "LUN" representation 6734-9-156f-127 is equivalent to 6734-0009-156f-0127. Furthermore, trailing groups containing only the "0" digit MAY be omitted along with the preceding "-". So, the "LUN" representation 4186-9 is equivalent to 4186-0009-0000-0000. Other concise representations of the LUN field MUST NOT be used.

3.1.2.6 iscsi-initiator-id

Optional string containing iSCSI name of initiator. This is a string similar to the *iscsi-target-name*, this time specifying the initiator (boot prom) side of the connection. If not specified, a host-unique default will be provided. The default assumed will be of the form:

`iqn.1986-03.com.sun:boot.<system-mac-address>`

3.2 /chosen node

3.2.1 /chosen Properties

These properties reflect either device arguments specified to the boot command, values obtained through DHCP or RARP, or defaults. They are always created (along with *host-ip, router-ip* and *subnet-mask* described in FWARC 2002/561) during an iSCSI boot.

3.2.1.1 iscsi-target-ip

Type: Prop-encoded array, property-encoded string.

Contents: Dotted-decimal formatted IP address (255.255.255.255) of remote iSCSI target.

3.2.1.2 iscsi-target-name

Type: Prop-encoded array, property-encoded string.

Contents: iSCSI qualified name of desired disk target, as described in RFC 3720 §3.2.6.3 (iSCSI Name Structure).

3.2.1.3 iscsi-network-bootpath

Type: Prop-encoded array, property-encoded string.

Contents: The complete *device path* to which the *device-specifier* of the iSCSI boot command was resolved.

3.2.1.4 iscsi-port

Type: Prop-encoded array, property-encoded string.

Contents: Decimal formatted integer from 1 to 65535, representing the TCP port number of the remote iSCSI implementation. This will default to 3260 if not specified.

3.2.1.5 iscsi-partition

Type: Prop-encoded array, property-encoded string.

Contents: Optional string specifying the bootable partition on the target.

3.2.1.6 iscsi-lun

Type: Prop-encoded array, property-encoded string.

Contents: hexadecimal dash-separated field encoding the 64-bit LUN of the remote target. See §3.1.2.5 above for more details on the formatting.

3.2.1.7 iscsi-initiator-id

Type: Prop-encoded array, property-encoded string.

Contents: iSCSI name of initiator. This is a string similar to the *iscsi-target-name*, this time specifying the initiator (boot prom) side of the connection. The format is specified in RFC 3720 §3.2.6.3. If not specified, the initiator id will default to:

```
iqn.1986-03.com.sun:boot.<system-mac-address>
```

3.2.1.8 iscsi-tpgt

Type: Prop-encoded array, property-encoded string.

Contents: Decimal formatted integer from 1 to 65535, representing the Target Portal Group Tag returned by the target during login.

3.3 /iscsi-hba node

The */iscsi-hba* node is a pseudo-node providing a disk HBA interface to export disk semantics while data is being transported over a network interface. The node itself contains code behaving like a SCSI HBA, exporting methods conformant with the binding presented in IEEE 1275 Annex E, SCSI Host Adapter package.

When a system is booted using iSCSI, the boot command must be presented with a network node with arguments, but the secondary booters and host operating system will see only that the */chosen*:*bootpath* property refers to something like "*/iscsi-hba/disk:a"* (the argument here is obtained from the *iscsi-partition* keyword).

This pseudo-node shall be created dynamically, and not be present in the device tree unless a boot command has been issued specifying a network boot with iSCSI arguments.

3.3.1 /iscsi-hba properties

3.3.1.1 obp-tftp-ihandle

Type: Prop-encoded array, encoded with *encode-int*.

Contents: *ihandle* of the *obp-tftp* package, at the time of open.

This is a temporary property, to convey hidden information from *obp-tftp* to */iscsi-hba.* The property is created during the open of the network device and is deleted at the completion of the */iscsi-hba* open.

3.3.1.2 compatible

Type: prop-encoded array, property-encoded string.

Contents: "SUNW,iscsi-hba"

3.3.2 /iscsi-hba methods

3.3.2.1 open **(- - okay?)**

Prepare this device for subsequent use.

3.3.2.2 close **(- -)**

Close this previously *open*ed device.

3.3.2.3 dma-alloc **(size - - virt)**

Allocate a memory region for later use.

3.3.2.4 dma-free **(dma-free - -)**

Free memory allocated with *dma-alloc*.

3.3.2.5 dma-map-in **(virt size cacheable? - - devaddr)**

Convert virtual address to device bus DMA address.

3.3.2.6 dma-map-out **(virt devaddr size - -)**

Free DMA mapping set up with *dma-map-in*.

3.3.2.7 max-transfer **(- - max-len)**

Returns the maximum DMA transfer length supported by the hardware.

3.3.2.8 set-address **(unit target - -)**

In iSCSI, this method performs no action other than consuming two items off the stack because the addresses being used are forced by the original open of the network device. The original definition from the binding was: *Sets the SCSI target number and unit number to which subsequent commands apply.*

3.3.2.9 set-timeout **(msecs - -)**

Sets the maximum length of time in milliseconds that the driver will wait for the completion of a command. The default value of zero means to wait indefinitely. A hardware error result is reported for a command that times out.

3.3.2.10 device-present? (lun target - - present?)

Returns the result of a SCSI *inquiry* sent to the iSCSI target. Note, this method is not documented in any previous binding, but is required for all HBAs supporting Sun's scsidisk.fth implementation.

3.3.2.11 show-children (- -)

Support method for *show-iscsi* debug command. Will open a network connection to target specified by */chosen:iscsi-network-bootpath~~device~~* property, and display the list of target strings returned by iSCSI target discovery.

3.3.2.12 no-data-command (cmd-addr - - error?)

Executes a simple SCSI command, automatically retrying under certain conditions.

cmd-addr is the address of a 6-byte command buffer containing an SCSI command that does not have a data transfer phase. Executes the command, retrying indefinitely with the same retry criteria as *retry-command*.

error? is nonzero if an error occurred, zero otherwise.

NOTE – *no-data-command* is a convenience function. It provides no capabilities that are not present in *retry-command*, but for those commands that meet its restrictions, it is easier to use.

3.3.2.13 execute-command

(buf-addr buf-len dir cmd-addr cmd-len - - hw-err? | statbyte 0)

Executes the SCSI command, which is stored in memory at cmd-addr and whose length is cmd-len. Dir is true if the data transfer phase of the SCSI command will transfer data from the device to memory, and false otherwise. *buf-addr* is the address of the memory buffer to be used for the data transfer phase, and *buf-len* is the expected maximum length of the data transfer phase. The memory buffer must be contained within a DMA-accessible region that was returned by a previous execution of *dma-alloc*. If *buf-len* is zero, indicating that the command is not expected to have a data transfer phase, both *buf-addr* and *dir* are ignored. *Hw-err?*, the returned hardware error status, is nonzero if the command could not be executed at all (perhaps due to the device not responding to the selection attempt). If hw-err? is zero, statbyte is the status byte returned by the status phase of the command.

3.3.2.14 retry-command

(buf-addr buf-len dir cmd-addr cmd-len #retries - - 0 | hw-err? stat | sensebuf 0 stat)

Executes a SCSI command, automatically retrying under certain conditions. *retry-command* is similar to *execute-command* except that *retry-command* automatically retries under certain failure conditions and automatically executes the “request sense” SCSI command as necessary. *#retries* is the maximum number of times that the command will be retried; if *#retries* is –1, the command will be retried indefinitely. *retry-command* returns 0 if the command eventually succeeds. Otherwise, it returns the status byte returned by the last attempted command on top of the stack (–1 if the command failed due to a hardware error). The second number on the stack (*hw-err?*) indicates whether or not the extended

sense information is available. If *hw-err?* is zero, the third number on the stack (*sensebuf*) is the address of a memory buffer containing the extended sense information returned by the "request sense" command that was executed after the last attempt to execute the desired command. The criteria for whether or not to retry the command are as follows:

a) If the requested number of retries have already been performed, do not retry.
b) If the failure is due to a hardware error, do not retry.
c) If the failure was due to a "device busy" condition reported in the status byte, retry.
d) Otherwise, execute the "get extended status" command and attempt to determine whether or not the failure could be retried based on the data in the returned sense buffer, as follows:
    1) Unknown error class (not 7) is not retryable.
    2) Filemark is not retryable.
    3) End of media is not retryable.
    4) Illegal length indicator is not retryable.
    5) sense key = No Sense is retryable.
    6) sense key = Recoverable error is retryable.
    7) sense key = Not Ready is retryable.
    8) sense key = Unit Attention is retryable.
    9) Transaction aborted due to Incoming SCSI Bus reset is retryable
    10) Otherwise, the error is not retryable.

3.3.2.15 short-data-command (data-len cmd-addr cmd-len - - error? | data-adr 0)

Executes a simple SCSI command, automatically retrying under certain conditions.

cmd-addr is the address and *cmd-len* the length of a command buffer containing an SCSI command whose data transfer phase is expected to transfer less than 256 bytes in an incoming direction. *data-len* is the expected length (1..255) of the data transfer. Executes the command, retrying indefinitely with the same retry criteria as *retry-command*.

error? is nonzero if an error occurred, zero otherwise. If *error?* is zero, *data-adr* is the address of a buffer containing the data transferred by the execution of the command.

NOTE—*short-data-command* is a convenience function, eliminating the need for allocating a DMA buffer. It is primarily intended for use with "informational" SCSI commands like "read block limits" and "inquiry".

3.4 /iscsi-hba/disk node

This node is a standard block device node, per IEEE 1275 §3.7.2 ("block" devices), with additions from the following specifications:

- FWARC 2008/070,
- IEEE 1275 Recommended Practices "Device Support Extensions" §12, and
- IEEE 1275 Recommended Practices #248, "Extension to Core – size method".

Implementation note: This node uses the unmodified scsidisk.fth code used by all other Sun HBAs.

3.4.1 disk node properties

3.4.1.1 lba64

Type: Boolean property, no value.

The presence of this property indicates support for the *read-blocks64*, *write-blocks64* and *#blocks64* methods described in §3.4.2.10 , §3.4.2.11 , and §3.4.2.13 .

3.4.1.2 device_type

Type: prop-encoded array, property-encoded string.

Contents: “block”

3.4.1.3 compatible

Type: prop-encoded array, property-encoded string.

Contents: “sd”

3.4.1.4 name

Type: prop-encoded array, property-encoded string.

Contents: “disk”

3.4.2 disk node methods

3.4.2.1 open **(- - okay?)**

Prepare this device for subsequent use.

3.4.2.2 close **(- -)**

Close this previously *open*ed device.

3.4.2.3 read **(addr len - - actual)**

Read device into memory, return actual byte count.

3.4.2.4 write **(addr len - - actual)**

Write memory buffer to device, return actual byte count.

3.4.2.5 seek **(pos.lo pos.hi - - status)**

Set device position for next *read* or *write*.

3.4.2.6 load (addr - - size)

Load a client program from device to memory.

3.4.2.7 size (- - d.size)

Return the size of the device in bytes.

Return, as a double number "*d.size*", the number of bytes of storage associated with the device or instance. If the size cannot be determined, return the double number -1.

3.4.2.8 read-blocks (addr block# #blocks - - #read)

Read *#blocks*, starting at *block#*, from device into memory.

3.4.2.9 write-blocks (addr block# #blocks - - #written)

Write *#blocks* from memory into device, starting at *block#*.

3.4.2.10 read-blocks64 (addr d.block# #blocks - - #read)

Read *#blocks*, starting at *d.block#*, from device into memory (Note: *d.block#* is a double number, see IEEE 1275 §7.3.2.3).

Read *#blocks* records of length block-size bytes from the device (starting at block *d.block#*) into memory (starting at *addr*.) Return *#read*, the number of blocks actually read.

If the device is not capable of random access (e.g., a sequential access tape device) *d.block#* is ignored.

3.4.2.11 write-blocks64 (addr d.block# #blocks - - #written)

Write *#blocks* from memory into device, starting at *d.block#* (Note: *d.block#* is a double number, see IEEE 1275 §7.3.2.3).

Write *#blocks* records of length block-size bytes from memory (starting at *addr*) to the device (starting at block number *d.block#*.) Return *#written*, the number of blocks actually written.

If the device is not capable of random access (e.g., a sequential access tape device), *d.block#* is ignored.

3.4.2.12 #blocks (- - blocks)

Return the size of the device in blocks.

Return, as an unsigned number "*blocks*", the number of blocks of storage associated with the device or instance, where a "block" is a unit of storage consisting of the number of bytes returned by the package's "*block-size*" method. If the size cannot be determined, or if the number of blocks exceeds the range of an unsigned number, return the maximum unsigned integer (which, because of Open Firmware's assumption of two's complement arithmetic, is equivalent to the signed number -1).

3.4.2.13 #blocks64 **(- - d.#blocks)**

Return the size of the device in blocks.

Return, as a double number "*d.#blocks*", the number of blocks of storage associated with the device or instance, where a "block" is a unit of storage consisting of the number of bytes returned by the package's "*block-size*" method. If the size cannot be determined, or if the number of blocks exceeds the range of a double number, return the double number -1.

3.4.2.14 block-size **(- - n)**

Return “granularity” for accesses to this device.

3.4.2.15 eject **(- -)**

After determining that the device is present, send a SCSI command 1B (start/stop unit) with the LoEJ and Start bits set to the combination of 0b10, indicating Eject Media. (Note, the Eject command does not appear to be documented in any previous binding, but is omnipresent in existing drivers).

3.4.2.16 max-transfer **(- - n)**

Return size of largest possible transfer.

3.5 Security Keys

3.5.1 User Interface

Part of the iSCSI protocol includes CHAP authentication, to ensure we are reaching the correct disks (see RFC 3723, §2.4.1). This authentication protocol contains a username and password, which must be kept private to prevent spoofing. During the boot process, these are obtained from Key Storage (see FWARC 2002/182), which is a form of NVRAM variable not exposed to general users. Solaris and secondary booters may obtain the keys with the OBP Client Interface *SUNW,get-security-key*, and thus use the same authentication parameters as the initial boot.

3.5.1.1 *set-ascii-security-key* **(“key-name< >key-value<eol>” --)**

key-name contains the name of the key (see security key registry), up to 64 characters long.

key-value contains the ASCII characters corresponding to desired key value, up to 64 characters long.

This is a parallel interface to the *set-security-key* method defined in FWARC 2002/182 and 2003/143. The original method allows only hexadecimal input, this method accepts ASCII strings. Since iSCSI passwords are usually specified in ASCII, hexadecimal input is both overkill and inconvenient.

This interface does not provide the ability to enter all possible key values – the use is restricted to keys composed of printable ASCII characters other than space (“!” through “~”). For keys containing more exotic characters, the string must be converted to hexadecimal and the original *set-security-key* method must be used.

3.5.2 Security Key Registry

The following two Security Key names are added to the Security Key registry.

3.5.2.1 chap-user

This is the username (sometimes known as short name, userid or simply name), which is transmitted over the network in the clear and is thus not considered strictly secret. In Solaris' *iscsiadm* this is set with the *-H* or *--CHAP-name* switch. It may be anywhere from 1 to 16 characters in length. We are maintaining this name as a security key simply to keep it together with the below *chap-password*, simplifying code by providing a single method used to retrieve both values. This will usually be set with a command such as:

```
ok set-ascii-security-key chap-user admin
```

3.5.2.2 chap-password

This is the authentication password, never transmitted over the network, which must be from 12 to 16 characters long. This is sometimes known as the CHAP secret or CHAP password. In Solaris' *iscsiadm* this is set with the *-C* or *--CHAP-secret* switch. This value is used as the hidden value in exchanging of public encrypted values to validate that both sides are using the same password. This will usually be set with a command such as:

```
ok set-ascii-security-key chap-password abcdefghijklmnop
```

4 DHCP Interface

A DHCP server may be configured to provide the iSCSI boot information required, with a *Root Path* option. This is specified in detail in RFC 4173, the summary of the format is:

```
iscsi:<servername>:<protocol>:<port>:<LUN>:<targetname>
```

The above string must start with the literal six characters “iscsi:”, followed by the five variables separated by colons.

The <*servername*> will be specified as an IP address (since OBP doesn't know about name-address translations) in dotted decimal form, as all openboot IP addresses are specified.

The <*protocol*> is required to be “6”, indicating TCP.

The <*port*> is a decimal string indicating the IP port used to communicate with the iSCSI server – usually 3260.

The <*LUN*> indicates the SCSI LUN holding the boot disk, in dashed hexadecimal format, as specified in §3.1.2.5 above.

The <*targetname*> is the long string used to identify iSCSI targets, as specified in RFC 3720 §3.2.6.3, ISCSI Name Structure (See also RFC 3722). An item to note is that this *Root Path* option consists of colon-separated items, and the last item contains at least one colon (and in some cases it will contain two). This works only because it is the last item in the DHCP variable list.

5 *show-iscsi* command

The *show-iscsi* command is provided as a debug facility, to inquire from the iSCSI target server which target-ids are available, if the server provides SendTargets capability (RFC 3720 §12.3).

For convenience, the *show-iscsi* command takes the same format argument as a manual configuration boot command. It ignores the *iscsi-target-id* keyword (which is nonetheless required to indicate an iscsi operation), and prints a list of valid target-ids and corresponding LUNs on the specified target. This format is convenient for the case where a non-DHCP boot command fails, the exact same argument(s) may be provided to the *show-iscsi* command to determine what the proper target name on the destination IP address could be.

6 Source Components

There are three major source directories affected by this work:

- obp/pkg/netinet – the existing tftp and http boot (wanboot) package, which exists in the running OBP under */packages/obp-tftp,* receives minor modifications to support iSCSI.
- obp/pkg/iscsi – a new module, the iSCSI code itself, which is placed in the running OBP under */iscsi-hba.*

7 Description of iSCSI boot internal mechanics

The semantics of network boot (where *open* can perform no I/O, only *load* may initiate activity to the device) compared to disk boot (where *open* must read disk labels and determine partitions before completing) have required us to use two items in the device tree to carry out a boot from an iSCSI disk – a network device and the pseudo-device */iscsi-hba/disk*.

The user must specify a boot string based on the network device which connects to the iSCSI target; either “boot net:dhcp”, or the manually specified keyword sequence “boot net:iscsi-target-id=...”. The network driver brings in the *obp-tftp* package, which parses the keywords. When the *load* method is invoked by the *boot* command, *obp-tftp* will proceed to ask RARP or DHCP for more information if specified, and then proceed to carry out one of three forms of boot: tftp, http or iSCSI.

In the case of iSCSI, after creating the */chosen* properties specified in 3.2.1 above, it will open the */iscsi-hba* pseudo-node to start the disk boot process. The driver for the */iscsi-hba* pseudo-node will obtain the information about the iscsi target from properties in */chosen* and initiate the TCP/IP connection to the iSCSI server. Once a connection has been established, the */chosen:bootpath* property will be re-written specifying */iscsi-hba/disk* as the boot device – this causes Solaris to find a disk device, and produce file I/O requests to that disk device rather than trying to perform network activity over a network device.

The secondary booter *ufsboot* is free to re-write the arguments to */iscsi-hba/disk*, which it does as part of its file-system initialization (usually an empty argument list is replaced with “:a”). Solaris will see a boot path consisting of “/iscsi-hba/disk:a” as a result.

8 References:

- RFC 3720 – Internet Small Computer Systems Interface (iSCSI)
- RFC 3721 – iSCSI Naming and Discovery
- RFC 3722 – String Profile for iSCSI names
- RFC 3723 – Securing Block Storage protocols over IP
- RFC 3783 – Command Ordering considerations with iSCSI
- RFC 4173 – Bootstrapping clients using iSCSI
- RFC 3980 – T11 NAA Naming format for iSCSI
- RFC 5048 – iSCSI Corrections and Clarifications
- FWARC 2002/182 – Key Storage Signature Verification
- FWARC 2003/143 – Extension of FWARC 2002/182, Security Keystore
- FWARC 2002/561 – OBP-TFTP Wanboot Extensions
- FWARC 2004/579 – Miscellaneous Wanboot Changes
- PSARC 2008/427 – iSCSI support in Solaris
- IEEE 1275 Recommended Practice "Device Support Extensions"
- IEEE 1275 Recommended Practice Proposal #248, "Extension to Core – size method"