# TCG Storage
# Security Subsystem Class: Enterprise

**Specification Version 1.0**
**Revision 1.0**
**January 27, 2009**

**Contacts:**

storagewg@trustedcomputinggroup.org

TCG

## TABLE OF CONTENTS

# 1 Introduction

## 1.1 Document Purpose

The Storage Workgroup specifications are intended to provide a comprehensive architecture for putting storage devices under policy control as determined by the trusted platform host, the capabilities of the storage device to conform with the policies of the trusted platform, and the lifecycle state of the storage device as a Trusted Peripheral.

## 1.2 Security Subsystem Classes

The Core Specification (see [2]) defines the TCG-related functions for a TCG Trusted Storage Device. However, not all trusted storage devices might support all functionality. There are multiple “classes” of Core Specification compliance, called Security Subsystem Classes (SSCs).

Security Subsystem Classes explicitly define the minimum acceptable Core Specification capabilities of a storage device in a specific “class”. A storage device in a specific class MAY have only some of the capabilities (tables, methods, access controls) defined in the Core Specification and MAY include additional capabilities definitions.

## 1.3 Scope and Intended Audience

This SSC specification is an implementation profile for storage devices built to:

- protect the confidentiality of stored user data and
- minimize the time to bring devices online.

A single threat model is assumed: unauthorized access to user data on the device once it leaves the owner’s control. The specification’s scope is storage devices deployed in systems that implement Fibre Channel (FC), Serial Attached SCSI (SAS), and Serial ATA (SATA) interfaces.

The intended audience for this specification is both trusted storage device manufacturers and developers that want to use these devices in their systems.

## 1.4 Goals

The goal of this specification is to define an implementation profile for storage devices that ensures interoperability between different vendor solutions. This is achieved by:

- Identification of a minimum subset of required functionality from the TCG SWG Core Architecture specification (see [2]);
- Definition of additional functionality needed to satisfy enterprise-class storage use cases;
- Definition of expected storage device behavior for all required functionality.

## 1.5 Key Words

The key words "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", and "MAY" in this specification are to be interpreted as described in [1]. These keywords are capitalized when used to unambiguously specify requirements over protocol and features behavior that affect the interoperability and security of the implementation. When these words are not capitalized, they are meant in their natural-language sense.

Additionally, the following terms are used in this specification to describe the requirement of particular features, including tables, methods, and usages thereof.

- **Mandatory (M):** The feature SHALL be supported by the storage device in order to be compliant with this specification. A compliance test SHALL validate the feature is operational.
- **Optional (O):** The feature MAY be supported by the storage device. If implemented, a compliance test SHALL validate the feature is operational.

## 1.6 Precedence

In the event of conflicting information in this specification and other documents, the precedence for requirements is:

1) this specification;
2) the Storage Interface Interactions Specification (see[7]);
3) select proposals to change the Core Specification (see [2]) (See appendix 10); and
4) the Core Specification (see [2]).

## 1.7 References

[1]. IETF RFC 2119, 1997, “Key words for use in RFCs to Indicate Requirement Levels”.

[2]. Trusted Computing Group (TCG),, 2007, “TCG Storage Architecture Core Specification”, Version 1.0, Revision 0.9 – Draft.

[3]. NIST, “Computer Security Division, Cryptographic Toolkit”, http://csrc.nist.gov

[4]. NIST FIPS-197, 2001, “Advanced Encryption Standard (AES)”

[5]. [INCITS T10/1731-D], “Information technology - SCSI Primary Commands - 4 (SPC-4)”

[6]. [ANSI INCITS 452-2008], “Information technology - AT Attachment 8 - ATA/ATAPI Command Set (ATA8-ACS)”

[7]. Trusted Computing Group (TCG), “TCG Storage Interface Interactions Specification”, Version 1.0, Revision 1.0

## 1.8 Definition of Terms

| Term | Definition |
|---|---|
| IF-RECV | An interface command used to retrieve security protocol data from the TPer. Reference the Core Specifiacation [2] |
| IF-SEND | An interface command used to transmit security protocol data to the TPer. Reference the Core Specifiacation [2] |
| Locking SP | A security provider that incorporates the Locking Template. The Locking Template is defined in the Core Specification. [2] |
| SSC | Security Subsystem Class [2] specifications describe profiled sets of TCG functionality |
| TCG Reset | A high-level reset type defined in the Core Specification. [2] |
| TPer | The TCG security subsystem within a storage device. Reference the Core Specifiacation [2] |
| VU | Parameters that are vendor unique |

# 2 Overview

*Begin Informative Content*

This specification is an implementation profile for trusted storage devices commonly deployed within Enterprise-class systems. It provides storage device implementation requirements needed to guarantee interoperability between storage devices from different vendors. Enterprise-class systems often deploy a mix of cross-vendor storage devices and interoperability is therefore key, both for non-trusted and trusted storage devices.

This specification defines a limited set of TCG Trusted Storage functionality that, combined with Full Disk Encryption (FDE), protects the confidentiality of user data at rest. Only a single threat scenario is addressed: removal of the storage device from its host system involving a power cycle of the storage device and subsequent unauthorized access to data stored on that device.

This specification assumes that hosts in Enterprise systems could have limited (computational) capabilities and/or operate within a system that has strict response time requirements. Based on this assumption, the objective of this specification is to define strict boundaries on the host-device communication protocol and data structures used in the TCG Storage Architecture. This prevents the host from having to maintain security configuration information on a per storage device basis and allows it to expect similar behavior for each SSC compliant storage device within the system.

To avoid requiring that the host perform dynamic discovery of features and values, the storage device behavior is unambiguously defined, and as such creation and deletion of tables and/or rows within tables is not required. This specification defines 2 SPs, the tables that are host-accessible within each SP, and the values within each table. The SPs and tables MAY be present in the storage device when it leaves manufacturing, See section 8.1.

This specification addresses a limited set of use scenarios. These scenarios are:

- Deploy storage device & Take Ownership: the device is integrated into its target system and ownership transferred by actively setting or changing the device's owner credential.
- Activate or Enroll Device: LBA ranges are configured, data encryption and access control credentials (re)generated and/or set on the storage device.
- Lock & Unlock Device: active unlocking of one or more LBA ranges by the host and locking of those ranges under host control via either an explicit lock or implicit lock triggered by a reset event.
- Repurpose & End-of-Life: erasure of data within one or more LBA ranges and reset of locking credential(s) for storage device repurposing or decommissioning.

*End Informative Content*

# 3 SSC Features and Capability Definitions

## 3.1 Interface Communications Protocol

An Enterprise SSC-compliant storage device SHALL implement the synchronous communications protocol (see section 4.4) using the SCSI (T10) or ATA (T13) defined security protocol commands. This SSC's implementation of the synchronous communications protocol calls for a single ComPacket / Packet / Subpacket combination per interface command and defines two Active ComIDs for communications using ComPackets.

## 3.2 Cryptographic Features

The storage device SHALL implement Full Disk Encryption for all host accessible user data stored on media. The storage device SHALL support AES 128 or AES 256 (see [4]).

## 3.3 Authentication

The storage device SHALL support password authorities and authentication with a maximum credential password size of 32-bytes.

## 3.4 Table Management

The tables and table rows required by this specification MAY be present in the storage device when the device leaves the manufacturer. The creation or deletion of tables in SPs post-manufacturing is outside the scope of this specification. The creation or deletion of rows in tables post-manufacturing is outside the scope of this specification.

## 3.5 Issuance

The SPs required by this specification MAY be present in the storage device when the storage device leaves the manufacturer. The issuance of SPs post-manufacturing is outside the scope of this specification.

## 3.6 SSC Discovery

Discovery is a process for the Host to examine the storage device's configurations and capabilities.

### 3.6.1 Discovery levels

Discovery is a process used to determine the capabilities of the TPer.

- **Level 0:** This discovery request is sent as an IF-RECV command. See appendix 10.2.14. The Security Protocol SHALL be 0x01 and the ComID SHALL be 0x0001. The TPer SHALL support the requirements in 3.6.2.
- **Level 1:** These TCG methods request basic TPer capabilities (i.e. via Properties) using simple host messaging requirements. The required support is defined in this specification.
- **Level 2:** TCG methods retrieve specified table cell values. The required support is defined in this specification.

## 3.6.2 Level 0 Discovery

The TPer SHALL support the LEVEL 0 DISCOVERY response data format defined in 10.2.14.

### 3.6.2.1 TPer feature

The TPer SHALL return the TPer Feature in the Level 0 Discovery response with:

Feature Code = 0x0001
Version = 0x1 or any version that supports the defined features in this SSC.
Length = 0x0C
SyncSupported = 1
StreamingSupported = 1

### 3.6.2.2 Locking Feature

The TPer SHALL return the Locking Feature in the Level 0 Discovery response with:

The Feature Code = 0x0002
Version = 0x1 or any version that supports the defined features in this SSC.
Length = 0x0C
LockingSupported = 1
MediaEncryption = 1

### 3.6.2.3 Enterprise SSC Feature

The TPer SHALL return the Enterprise feature in the Level 0 Discovery response.

**Table 1 - Enterprise SSC Descriptor Format**

| Byte \ Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| 0 | (MSB) | | | Feature Code | | | | |
| 1 | | | | | | | | (LSB) |
| 2 | Version | | | | Reserved | | | |
| 3 | Length | | | | | | | |
| 4 | (MSB) | | | Base ComID | | | | |
| 5 | | | | | | | | (LSB) |
| 6 | (MSB) | | | Number of ComIDs | | | | |
| 7 | | | | | | | | (LSB) |
| 8 | Reserved for future common SSC parameters | | | | | | | Range Crossing |
| 9 - 19 | Reserved for future common SSC parameters | | | | | | | |

The TPer SHALL support:

Feature Code = 0x0100
Version = 0x1 or any version that supports the defined features in this SSC.
Length = 0x10
Base ComID = base ComID supported by the TPer (e.g., 0x07FE)
Number of ComIDs = 0x0002 min

Range Crossing = VU 0 = The Storage device supports commands addressing consecutive LBAs in more than one LBA range if all the LBA ranges addressed are unlocked. See 8.3.10.
1 = The storage device terminates commands addressing consecutive LBAs in more than one LBA range. See 8.3.10.

# 4 Communications

## 4.1 Communication Properties

The TPer SHALL support fixed size communication buffers with a minimum size of 1024 bytes each. The number of buffers SHALL be sufficient to support concurrent communications with each ComID and/or open session as supported by the TPer, whichever is greater.

For each ComID, the physical size of the buffers SHALL be reported to the host via the `Properties` method (see section 6.2.1.1).

The TPer SHALL terminate any IF-SEND command whose transfer length is greater than the reported MaxComPacketSize size for the corresponding ComID. For details, reference “Invalid Transfer Length parameter on IF-SEND” in the Storage Interface Interactions Specification [7].

Data generated in response to methods contained within an IF-SEND command payload subpacket (including the required ComPacket / Packet / Subpacket overhead data) SHALL fit entirely within the response buffer. If the method response and its associated protocol overhead do not fit completely within the response buffer, the TPer

1) SHALL terminate processing of the IF-SEND command payload,
2) SHALL NOT return any part of the method response if the Sync Protocol is being used, and
3) SHALL return an empty response list with a TCG status code of RESPONSE_OVERFLOW in that method’s response status list.

## 4.2 Supported Security Protocols

The TPer SHALL support IF-RECV commands with a Security Protocol values of 0x00, 0x01 and 0x02.
The TPer SHALL support IF-SEND commands with a Security Protocol values of 0x01 and 0x02.

If the host sends an IF-SEND or IF-RECV to an unsupported Security Protocol, the TPer SHALL terminate the command as defined in the Storage Interface Interactions Specification. Reference “Invalid Security Protocol ID parameter” in [7].

For an IF-RECV command with Security Protocol set to 0x00 and Security Protocol Specific set to 0x0000 (Return list of supported protocols), the TPer SHALL respond in accordance to the SCSI (see [5]) or ATA (see [6]) specifications for Security Protocol In and Trusted Receive.

For an IF-RECV command with Security Protocol set to 0x00 and Security Protocol Specific set to 0x0001 (Return a certificate), the TPer SHALL respond in accordance to the SCSI (see [5]) or ATA (see [6]) specifications for Security Protocol In and Trusted Receive.

For an IF-RECV command with Security Protocol set to 0x01 and Security Protocol Specific set to 0x0001 (Level 0 Discovery), the TPer SHALL return one or more 512-byte blocks that describe the attributes of the TCG security protocol corresponding to Security Protocol 0x01. The return data structure SHALL comply to the requirements in 3.6.2.

The TPer SHALL support IF-SEND and IF-RECV commands with Security Protocol set to 0x02 for the Protocol Stack Reset Command function defined in appendix 10.2.19. If the host sends an IF-SEND with Security Protocol set to 0x02 and an invalid or unsupported Request code in the payload, the TPer SHALL prepare a response with "No Response Available" as defined in appendix 10.2.19.

## 4.3 ComIDs

For the purpose of communication using Security Protocol 0x01, the TPer SHALL support:.

- ComIDs values 0x07FE and 0x07FF and Extended ComID values 0x07FE0000 and 0x07FF0000 for communications using the Synchronous Protocol, and
- ComID value 0x0001 for Device Level 0 Discovery. See section 3.6 and appendix 10.2.14.

Mandatory ComIDs SHALL be Active (in the “Issued” or “Associated” state).

If the host uses an inactive or unsupported ComID in an IF-SEND or IF-RECV, the TPer SHALL respond as defined in appendix 10.2.21.

## 4.4 Synchronous Protocol

The TPer SHALL support the Synchronous Protocol for communications using ComPackets.

### 4.4.1 Protocol States and State Transitions

Figure 1 describes the synchronous protocol states and state transitions.

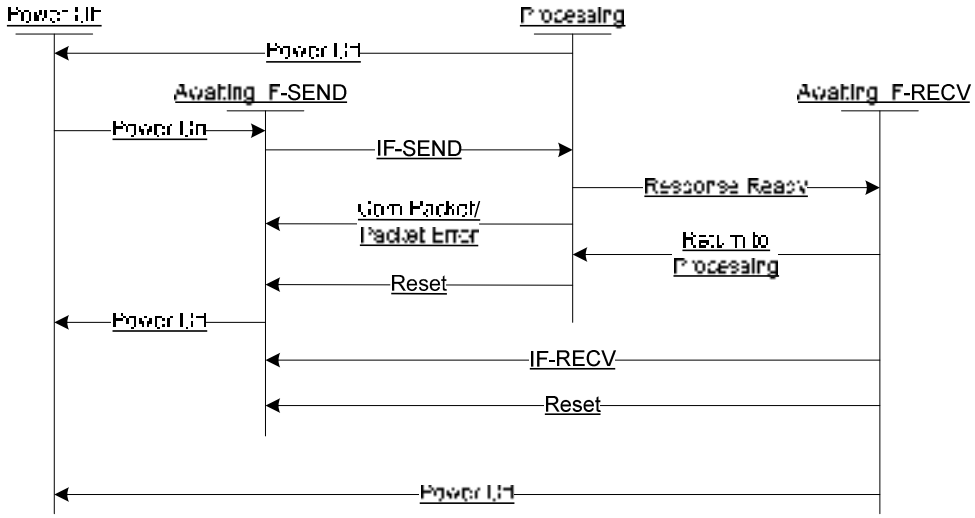



**Figure 1– Synchronous Protocol Stack State Diagram**

This specification defines the following protocol states for each valid ComID:

**State “Power-Off”** – In this state, power is removed from the TPer and it is completely unresponsive.

**State “Awaiting IF-SEND”** – In this state, the TPer command interface is ready and there are no outstanding IF-SEND/IF-RECV commands for the specified ComID. A command is “outstanding” if it has entered the “Processing” or “Awaiting IF-RECV” state. A command is not considered “outstanding” if it is sitting in the TPer command queue awaiting initial processing by the device.

While in this state, if IF-SEND is received or dequeued with the ComID for this state machine, the TPer MAY request command payload transfer and SHALL return interface status to the host.

While in this state, if IF-RECV is received or dequeued with the ComID for this state machine, the TPer SHALL return a response ComPacket the specified ExtendedComID with the Length, OutstandingData, and MinTransfer fields set per “All Response(s) returned – no further data” defined in 10.2.12.

**State “Processing”** – In this state, the TPer has begun processing the payload of an IF-SEND command.

While in this state, the TPer SHALL terminate any received or dequeued IF-SEND commands as defined in the Storage Interface Interactions Specification [7] “Synchronous Protocol Violation”.

While in this state, the TPer SHALL return a response ComPacket for any received or dequeued IF-RECV commands for the specified ExtendedComID with the Length, OutstandingData, and MinTransfer fields set per “Response(s) to come, no Response(s) available” defined in 10.2.12.

**State “Awaiting IF-RECV”** – The TPer has completely processed the TCG data payload and has the associated TCG response ready for retrieval by the host.

While in this state, if IF-RECV is received or dequeued with the ComID for this state machine and a transfer length less than the amount of response data staged for the ComID, the TPer SHALL return a response ComPacket for the specified ExtendedComID with the Length, OutstandingData, and MinTransfer fields set per “Response ready, insufficient transfer length request” defined in 10.2.12.

While in this state, the TPer SHALL terminate any received or dequeued IF-SEND command with a status as defined in the Storage Interface Interactions Specification [7] “Synchronous Protocol Violation”.

This specification defines the following protocol state transitions for each valid ComID:

**“Power Off : Awaiting IF-SEND”** - This transition occurs automatically when the TPer is powered on.

**“Awaiting IF-SEND : Processing”** - This transition occurs when an IF-SEND command with the ComID associated with this state machine is received or dequeued and successfully completes data transfer of the command payload.

**“Awaiting IF-SEND : Power Off”** - This transition occurs when the TPer is powered off.

**“Processing : Awaiting IF-SEND”** - This transition occurs when the TPer receives:

- an interface initiated TCG reset (see 4.5.1),
- a Protocol Stack Reset Command for the ComID of this state machine (see 4.5.2), or
- when the TPer detects an error in the IF-SEND payload that prevents the TPer from resolving an intended session for the IF-SEND command payload (see section 4.4.2.3) .

**“Processing : Awaiting IF-RECV”** - This transition occurs when the TPer has completely processed the contents of the IF-SEND command and has a complete response available for retrieval by the host. A separate response MAY be generated for each method in the IF-SEND.

**“Processing : Power Off”** - This transition occurs when the TPer is powered off.

**“Awaiting IF-RECV : Awaiting IF-SEND”** - This transition occurs when the TPer receives:

- an interface initiated TCG reset (see 4.5.1),
- a Protocol Stack Reset Command for the ComID of this state machine (see 4.5.2),
- an IF-RECV able to retrieve the entire response resulting from the IF-SEND, or
- an IF-RECV for the last of multiple responses resulting from the IF-SEND.

**“Awaiting IF-RECV : Power Off”** - This transition occurs when the TPer is powered off.

**“Awaiting IF-RECV : Processing”** - This transition occurs when the IF-SEND contained multiple method invocations, the TPer is preparing a separate response for each method, an IF-RECV able to retrieve the the current response was received and additional methods need to be processed.

## 4.4.2 Payload Encoding

### 4.4.2.1 Stream Encoding Modifications

The TPer SHALL support tokens listed in Table 2. If an unsupported token is encountered, the TPer SHALL treat this as a streaming protocol violation and return an error per the definition in section 4.4.2.3.

**Table 2 – Supported Tokens**

| Acronym | Meaning |
|---|---|
| | Tiny atom |
| | Short atom |
| | Medium atom |
| | Long atom |
| SL | Start List |
| EL | End List |
| SN | Start Name |
| EN | End Name |
| CALL | Call |
| EOD | End of Data |
| EOS | End of session |
| ST | Start transaction |
| ET | End of transaction |

For supported atom tokens the TPer SHALL support token atoms with the B bit set to 0 or 1 and the S bit set to 0.

### 4.4.2.2 TCG Packets

Within a single IF-SEND/IF-RECV command, the TPer SHALL support a ComPacket containing one Packet, which contains one Subpacket. Host MAY discover TPer support of capabilites beyond this requirement in the parameters returned in response to a `Properties` method.

The TPer MAY ignore Credit Control Subpackets sent by the host. The host MAY discover TPer support of Credit Management with Level 0 Discovery. See section 3.6.2.

The TPer MAY ignore the AckType and Acknowledgement fields in the Packet header on commands from the host and set these fields to zero in its responses to the host. The host MAY discover TPer support of the TCG packet acknowledgement/retry mechanism with Level 0 Discovery. See section 3.6.2.

The TPer MAY ignore TCG packet sequence numbering and not enforce any sequencing behavior. The discovery of TPer packet sequence numbering.support is outside the scope of this SSC.

#### 4.4.2.3 Payload Error Response

The TPer SHALL respond according to the following rules if it encounters a streaming protocol violation:

- If the error is on Session Manager or is such that the TPer cannot resolve a valid session ID from the payload (i.e. errors in the ComPacket header or Packet header), then the TPer SHALL discard the payload and immediately transition to the “Awaiting IF-SEND” state.
- If the error occurs after the TPer has resolved the session ID, then the TPer SHALL close the session and prepare a `CloseSession` method for retrieval by the host.

## 4.5 Storage Device Resets

### 4.5.1 Interface Resets

Interface resets that generate TCG reset events are defined in the Storage Interface Interactions Specification [7] “Reset Mapping”.
Interface initiated TCG reset events SHALL result in:

1. All open sessions SHALL be aborted;
2. All uncommitted transactions SHALL be aborted;
3. All pending session startup activities SHALL be aborted;
4. All TCG command and response buffers SHALL be invalidated;
5. All related method processing SHALL be aborted;
6. For each ComID, the state of the synchronous protocol stack SHALL transition to “Awaiting IF-SEND” state;
7. No notification of these events SHALL be sent to the host.

### 4.5.2 Protocol Stack Reset Commands

An IF-SEND containing a Protocol Stack Reset Command SHALL be supported as defined in appendix 10.2.19.

# 5 Data Types

The TPer SHALL support TCG streaming protocol as defined in the Core Specification [2] as modified by TCG Stream Typing Removal Proposal (see appendix 10.2.2).

# 6 Session Manager

## 6.1 Session Timeouts

During session startup, if the host specifies a timeout outside of the supported TPer timeout interval, the TPer rejects the session startup command and returns a failed `SyncSession` method call with TCG status INVALID_PARAMETER.

## 6.2 Session Manager Method Requirements

TPer support for the Session Manager method in Table 3.

**Table 3 – Session Manager Methods**

| Method Name | Method Type |
|---|---|
| Properties | Session Manager |
| StartSession | Session Manager |
| SyncSession | Session Manager |
| CloseSession | Session Manager |

### 6.2.1 Session Manager Methods

#### 6.2.1.1 Properties

The TPer SHALL implement the `Properties` method with the constraints stated in this subsection.

When a `Properties` method is received, the TPer SHALL return the following parameters:

MaxPacketSize = min 1024 bytes - 20 bytes (ComPacket Hdr)
MaxComPacketSize = min 1024 bytes
MaxResponseComPacketSize = min 1024 bytes
MaxSessions = min 1
MaxIndTokenSize = min 256 bytes
MaxAuthentications = min 2
MaxTransactionLimit = min 1

The TPer SHALL ignore any parameters not supported by the TPer when the host tries to set its value and not include it nor its value in the return data (behavior specified in the Core Specification [2]).

The TPer shall return all property name/value pairs for capabilities that it supports. For capabilities not supported by the TPer (e.g,, Read-Only sessions), the associated property name/value pair (in this case, MaxReadSessions) shall be omitted from the TPer's response.

#### 6.2.1.2 StartSession

The TPer SHALL implement the `StartSession` method with the constraints stated in this subsection.

TPer support of the following parameters is mandatory:

- HostSessionID
- SPID
- Write (support of Write = True mandatory)

Attempts to use unsupported parameters SHALL result in a SyncSession response with TCG status INVALID_PARAMETER.

#### 6.2.1.3 SyncSession

The TPer SHALL implement the `SyncSession` method with the constraints stated in this subsection.

Device support of the following parameters is mandatory:

- HostSessionID
- SPSessionID

#### 6.2.1.4 CloseSession

The `CloseSession` method on the session manager SHALL only be invoked by the TPer in response to an erroneous IF-SEND from the host.

# 7 Templates

## 7.1 Definitions

For the purpose of this section, the following definitions SHALL apply:

- The TPer returns an INVALID_COMMAND TCG status when the host attempts to invoke a method designated “Excluded”.
- The TPer returns a NOT_AUTHORIZED TCG status when the host attempts to invoke a method on a table/object not permitted by the access control definitions.

## 7.2 Supported Templates

The TPer SHALL support the following modified templates:

- Base
- Admin
- Locking
- Crypto

### 7.2.1 TPer Template Requirements

The template requirements in Table 4 are mandatory.

**Table 4 – TPer Templates**

| Template Name | SSC Reference Section | Minimum Number Instantiable | Maximum Number Instantiable |
|---|---|---|---|
| Admin | Section 7.4 | 1 | 1 |
| Base | Section 7.3 | 2 | VU |
| Locking | Section 7.5 | 1 | 1 |
| Crypto | Section 7.6 | 2 | VU |

## 7.3 Base Template

This subsection defines the modified Base Template as applicable for this specification.

### 7.3.1 Base Template Table Requirements

Support for the Base Template table access requirements in Table 5 is mandatory.

**Table 5 – Base Template Tables**

| Table Name | Table Type |
|---|---|
| Authority | Object |
| C_PIN | Object |

### 7.3.2 Base Template Method Requirements

Support for the Base Template method requirements in Table 6 is mandatory.

**Table 6 – Base Template Methods**

| Method Name | Method Type |
|---|---|
| Get | Object |
| Set | Object |
| Next | Table |
| Authenticate | SP |
| GetACL | Table |

## 7.3.3 Base Template Method Details

### 7.3.3.1 Get

The UID for the Get method is: = 00 00 00 06 00 00 00 06

The TPer SHALL implement the Get method with the constraints stated in section 5 and modified according to the below definition:

```
TableUID.Get [
ObjectUID.Get [
       Cellblock : cell_block,
       ParamCheck = boolean ]
=>
[ Result : get_result,
ParamCheck = uinteger_2 ]
```

Method behavior is modified as follows:

The host MAY use the ParamCheck parameter to request a check value for a PIN credential value in the Get method result.

If the ParamCheck parameter value is True, the TPer SHALL perform a check value calculation as defined in section 7.3.3.3 on the result values portion of the method return result. The check value is returned as the ParamCheck portion of the method result.

If the ParamCheck parameter is omitted or if its value is False, the ParamCheck portion of the method result SHALL be omitted and only the contents of the requested Cellblock returned.

If the get_result contains no data, the TPer shall not return the ParamCheck Name-Value pair.

For a Get method example of ParamCheck calculations, see appendix 11.2.

#### 7.3.3.2 Set

The UID for the `Set` method is: = 00 00 00 06 00 00 00 07

The TPer SHALL implement the `Set` method with the constraints stated in section 5 and modified according to the below definition:

```
Table.Set [
Object.Set [
       Where : cell_block,
       Values : set_values,
       ParamCheck = uinteger_2 ]
=>
[ ]
```

Method behavior is modified as follows:

The host SHALL be required to provide at least one value to set in the "Values" parameter. In the absence of any Values, the TPer SHALL return INVALID_PARAMETER.

For Byte and Array tables, the TPer MAY ignore EndRow in the Where parameter.

For Array tables, Object tables, and Objects, the TPer MAY ignore StartColumn and EndColumn in the Where parameter.

The host MAY use the ParamCheck parameter to provide a check value for a PIN credential value.

If the ParamCheck parameter is supplied, the TPer SHALL first perform a calculation as defined in section 7.3.3.3. The TPer SHALL compare its computed value with that supplied in the ParamCheck parameter. If the TPer computed value matches the supplied ParamCheck value, then the device SHALL continue execution of the `Set` method. If the TPerTPer's computed value does not match the supplied ParamCheck value, then the method SHALL fail with TCG status INVALID_PARAMETER.

The TPer SHALL NOT calculate nor validate a check when the ParamCheck parameter value is omitted.

For a `Set` method example of ParamCeck calculations, see appendix 11.1.

.

#### 7.3.3.3 Check Value Algorithm

The TPer SHALL implement the ParamCheck Longitudinal Redundancy Check (LRC) for `Get` and `Set` method calls on a PIN value. The LRC word size is 16-bits and the seed (initial value) is 0x5056.

Only the PIN parameter value is used for the LRC calculation. Tokens, names and ParamCheck parameter are not input to the LRC calculation.

Words are input to the LRC calculation in stream order, i.e. in Big Endian order.

If a PIN parameter is not an even number of bytes in length, a pad byte of 0x00 is prepended as the MSB.

The calculation algorithm is:

- Initialize the LRC calculation (initial value 0x5056)
- For value in list of values
  - Convert value into its TCG byte representation using minimal width stream encoding
  - Discard any bytes associated with TCG token headers
  - If sizeof(value) is not a multiple of 2
    - Prepend one byte equal to 0x00 to the MSB of the value
  - Feed value into the LRC calculation
- Get final LRC result

In pseudo code, the above MAY be expressed as:

```
lrc = 0x5056  // Initialize LRC
for value in values   // 'values' represents all values from any name/value
                      // pairs in the Values parameter of the Set method
                      // or in the get_result of the Get method.
   bytes = value.tcg_encode()  // Convert value to its TCG encoding
   bytes = bytes.strip_tcg_header() // Remove TCG token headers
   if len(bytes) modulo 2 != 0
      bytes = bytes.prepend(0x00)
   // What follows is the heart of the LRC calculation…
   for word in bytes  // 'word' is 2 bytes
      lrc = lrc xor word
return lrc
```

## 7.4 Admin Template

This subsection defines the modified Admin Template as applicable for this specification.

### 7.4.1 Admin Template Table Requirements

There are no Admin Template table requirements.

### 7.4.2 Admin Template Method Requirements

There are no Admin Template method requirements.

## 7.5 Locking Template

This subsection defines the modified Locking Template as applicable for this specification.

### 7.5.1 Locking Template Table Requirements

The Locking Template table access requirements in Table 7 are mandatory.

**Table 7 - Locking Template Tables**

| Table Name | Table Type |
|---|---|
| LockingInfo | Array |
| Locking | Object |

### 7.5.2 Locking Template Method Requirements

The following Locking Template method requirements in Table 8 are mandatory.

**Table 8 - Locking Template Methods**

| Method Name | Method Type | Requirement |
|---|---|---|
| Erase | Object | Mandatory |

### 7.5.3 Locking Template Method Details

#### 7.5.3.1 Erase

The UID for the `Erase` method is: 00 00 00 06 00 00 08 03

The `Erase` method is specific for this specification and SHALL be supported by the device. This method is used to cryptographically erase user data within a specific LBA Range and to reset the access control (‘Locking”) of that LBA Range.

The `Erase` method is an object method and is defined as:

```
LockingObjectUID.Erase [ ]
=>
[ ].
```

When invoked, the method's side effects are:

- The TPer SHALL eradicate the current data encryption key for the LBA Range managed by the Locking object on which the method is invoked;
- The TPer SHALL generate a new data encryption key for the LBA Range managed by the Locking object on which the method is invoked;
- The TPer SHALL reset the `ReadLockEnabled`, `WriteLockEnabled`, `ReadLocked`, and `WriteLocked` column values to "False" for the Locking object on which the method is invoked;
- The TPer SHALL set the associated BandMaster credential to the MSID Credential value (see appendix 9) and, when applicable, set Tries to zero.
- The TPer SHALL NOT change `RangeStart` and `RangeLength`.

The method call fails with TCG SWG status NOT_AUTHORIZED if:

- The referenced object does not exist;
- The referenced object is not an object stored in the Locking Table.

## 7.6 Crypto Template

This subsection defines the modified Crypto Template as applicable for this specification.

### 7.6.1 Crypto Template Table Requirements

There are no Crypto Template table requirements.

### 7.6.2 Crypto Template Method Requirements

The Crypto Template methods access requirements in Table 9 are mandatory.

**Table 9 - Crypto Template Methods**

| Method Name | Method Type |
|---|---|
| Random | SP |

### 7.6.3 Crypto Template Method Details

#### 7.6.3.1 Random

The TPer SHALL implement the `Random` method with the constraints stated in this subsection. TPer support of the following parameters is mandatory:

- Count

Attempts to use unsupported parameters SHALL result in a method failure response with TCG status INVALID_PARAMETER.

The TPer SHALL support Count parameter values less than or equal to 32.

# 8 SP Implementation Details

## 8.1 SP life cycle

Enterprise SSC-compliant TPers in which SPs are created in manufacturing SHALL conform to the life cycle defined in this section.

SPs created in manufacturing in an Enterprise SSC-compliant TPer SHALL support the "Manufactured" state as defined in this specification. Other SP states and their state-specific implications are outside the scope of this specification and considered vendor specific.

The "Manufactured" state is the standard operational state of an SP created in the manufacturing process, which defines the initial required access control settings and preconfigurations of an SP based on the Templates incorporated into the SP, prior to personalization.

For Enterprise SSC-compliant TPers that support issuance, refer to the Core Specification [2] for the life cycle states and life cycle management. The Core Specification [2] describes the life cycle states for SPs that are created through the issuance process. TPer requirements attendant to issuance are outside the scope of this document.

## 8.2 Admin SP

This subsection defines specific requirements as applicable to the Admin SP.

The Admin SP SHALL instantiate the Base Template, Admin Template and Crypto Template subject to requirements in section 7 of this specification.

### 8.2.1 Authorities & Credentials

The Admin SP SHALL implement the Anybody, Admins, Makers, and SID Authorities.

#### 8.2.1.1 Authority table

**Table 10 - Admin SP Authority table**

| UID | Name | Common Name | IsClass | Class | Enabled | Secure | HashAnd Sign | Present Certificate | ► |
|---|---|---|---|---|---|---|---|---|---|
| 00 00 00 09 00 00 00 01 | "Anybody" | "Anybody" | FALSE | Null | TRUE | None | None | FALSE | a |
| 00 00 00 09 00 00 00 03 | "Makers" | "Maker" | TRUE | Null | TRUE | None | None | FALSE | b |
| 00 00 00 09 00 00 00 06 | "SID" | "TPerOwner" | FALSE | Null | TRUE | None | None | FALSE | c |

**Continued**

| ► | Operation | Credential | Response Sign | Response Exch | Clock Start | Clock End | Limit | Uses | Log | LogTo |
|---|---|---|---|---|---|---|---|---|---|---|
| a | None | Null | Null | Null | <date_0_value> | <date_0_value> | 0 | 0 | None | Null |
| b | None | Null | Null | Null | <date_0_value> | <date_0_value> | 0 | 0 | None | Null |
| c | Password | 00 00 00 0B 00 00 00 01 | Null | Null | <date_0_value> | <date_0_value> | 0 | 0 | None | Null |

#### 8.2.1.2 Credential Table Group (C_PIN) table

The Admin S P C_PINs are defined in Table 11. PIN values are a maximum size of 32-bytes each.

**Table 11 - Admin SP C_PIN table**

| UID | Name | Common Name | PIN | CharSet | TryLimit | Tries | Persistence |
|---|---|---|---|---|---|---|---|
| 00 00 00 0B 00 00 00 01 | "SID" | "" | <PIN0_value> | Null | VU | VU | VU |
| 00 00 00 0B 00 00 84 02 | "MSID" | "" | <PIN1_value> | Null | VU | VU | VU |

This specification defines a SSC specific non-changeable PIN known as MSID. This PIN value SHALL be set at manufacturing time and MAY be used as an initial value for other PIN authority credential values on the device.

The SID `PIN` value SHALL be equal to the MSID Credential value at manufacturing time. Refer to section 9.1 for an informal discussion on the role of MSID Credential.

### 8.2.2 ACE table

The ACE definitions in Table 12 are required for compliance to this specification. The text in parentheses is only to provide clarification.

**Table 12 - Admin SP ACE table**

| UID | Name | Common Name | Boolean Expr | RowStart | RowEnd | ColStart | ColEnd |
|---|---|---|---|---|---|---|---|
| 00 00 00 08 00 00 00 01 | "Anybody" | "" | 00 00 00 09 00 00 00 01 (Anybody) | Null | Null | "" | "" |
| 00 00 00 08 00 00 00 03 | "Makers" | "" | 00 00 00 09 00 00 00 03 (Makers) | Null | Null | "" | "" |
| 00 00 00 08 00 00 02 01 | "SID" | "" | 00 00 00 09 00 00 00 06 (SID) | Null | Null | "" | "" |
| 00 00 00 08 00 00 8C 03 | "SID_SetSelf" | "" | 00 00 00 09 00 00 00 06 (SID) | Null | Null | "PIN" | "PIN" |
| 00 00 00 08 00 00 8C 04 | "MSID_Get" | "" | 00 00 00 09 00 00 00 01 (Anybody) | Null | Null | "PIN" | "PIN" |
| 00 00 00 08 00 00 8C 05 | "SID_Set Makers" | "" | 00 00 00 09 00 00 00 06 (SID) | Null | Null | "Enabled" | "Enabled" |

### 8.2.3 AccessControl table

The access control definitions in Table 13 are required for compliance to this specification. The text in parentheses is presented only for clarification. The CommonName fields in Table 13 are only for clarification and MAY exceed the length limit for names.

**Table 13 - Admin SP AccessControl table**

| Row Number | UID | InvokingID | MethodID | CommonName | ACL | Log | AddACE ACL | RemoveACE ACL | GetACL ACL |
|---|---|---|---|---|---|---|---|---|---|
| VU | VU | 00 00 00 00<br>00 00 00 01<br>(ThisSP) | 00 00 00 06<br>00 00 00 0C<br>(Authenticate) | Anybody-Authenticate-AdminSP | 00 00 00 08<br>00 00 00 01<br>(Anybody) | None | Null | Null | 00 00 00 08<br>00 00 00 01<br>(Anybody) |
| VU | VU | 00 00 00 09<br>00 00 00 00<br>(Authority table) | 00 00 00 06<br>00 00 00 08<br>(Next) | Makers-Next-Authority table | 00 00 00 08<br>00 00 00 03<br>(Makers) | None | Null | Null | 00 00 00 08<br>00 00 00 03<br>(Makers) |
| VU | VU | 00 00 00 09<br>00 00 00 01<br>(Anybody Authority object) | 00 00 00 06<br>00 00 00 06<br>(Get) | Anybody-Get-Anybody Authority Object | 00 00 00 08<br>00 00 00 01<br>(Anybody) | None | Null | Null | 00 00 00 08<br>00 00 00 01<br>(Anybody) |
| VU | VU | 00 00 00 09<br>00 00 00 03<br>(Makers Authority object) | 00 00 00 06<br>00 00 00 06<br>(Get) | Makers-Get-Makers Authority Object | 00 00 00 08<br>00 00 00 03<br>(Makers) | None | Null | Null | 00 00 00 08<br>00 00 00 03<br>(Makers) |
| VU | VU | 00 00 00 09<br>00 00 00 06<br>(SID Authority object) | 00 00 00 06<br>00 00 00 06<br>(Get) | SID-Get-SID Authority Object | 00 00 00 08<br>00 00 02 01<br>(SID) | None | Null | Null | 00 00 00 08<br>00 00 02 01<br>(SID) |
| VU | VU | 00 00 00 0B<br>00 00 00 00<br>(C_PIN table) | 00 00 00 06<br>00 00 00 08<br>(Next) | Makers-Next-C_PIN table | 00 00 00 08<br>00 00 00 02<br>(Makers) | None | Null | Null | 00 00 00 08<br>00 00 00 02<br>(Makers) |
| VU | VU | 00 00 00 0B<br>00 00 00 01<br>(SID C_PIN object) | 00 00 00 06<br>00 00 00 07<br>(Set) | SID_SetSelf-Set-SID_C_PIN object | 00 00 00 08<br>00 00 8C 03<br>(SID_SetSelf) | None | Null | Null | 00 00 00 08<br>00 00 02 01<br>(SID) |
| VU | VU | 00 00 00 0B<br>00 00 84 02<br>(MSID C_PIN object) | 00 00 00 06<br>00 00 00 06<br>(Get) | MSID_Get-Get-MSID C_PIN Object | 00 00 00 08<br>00 00 8C 04<br>(MSID_Get) | None | Null | Null | 00 00 00 08<br>00 00 02 01<br>(SID) |
| VU | VU | 00 00 00 09<br>00 00 00 03<br>Makers Authority Object | 00 00 00 06<br>00 00 00 07<br>(Set) | SID_Set Makers - Set-Makers Authority Object | 00 00 00 08<br>00 00 8C 05<br>(SID_Set Makers) | None | Null | Null | 00 00 00 08<br>00 00 02 01<br>(SID) |
| VU | VU | 00 00 00 00<br>00 00 00 01<br>(ThisSP) | 00 00 00 06<br>00 00 06 01<br>(Random) | Anybody-Random | 00 00 00 08<br>00 00 00 01<br>(Anybody) | None | Null | Null | 00 00 00 08<br>00 00 00 01<br>(Anybody) |

## 8.3 Locking SP

This subsection defines specific requirements for the Locking SP.

The SPID for the Locking SP is 00 00 02 05 00 01 00 01.

The Locking SP SHALL instantiate the Base Template, Locking Template and Crypto Template subject to constraints set forth in section 7 of this specification.

### 8.3.1 Locking SP authorities

Each LBA range has an associated authority allowing an authorized entity to manage the associated LBA Range. Management includes locking and unlocking of that range, enabling or disabling Range locking, and setting both start LBA and size of the range. The Authority table contains rows only for supported locking ranges.

**Table 14 - Locking SP Authority table**

| UID | Name | CommonName | IsClass | Class | Enabled | Secure | HashAnd Sign | Present Certificate | ▶ |
|---|---|---|---|---|---|---|---|---|---|
| 00 00 00 09 00 00 00 01 | "Anybody" | "Anybody" | FALSE | Null | TRUE | None | None | FALSE | a |
| 00 00 00 09 00 00 80 01 | "BandMaster0" | "BandMaster" | FALSE | 00 00 00 09 00 00 84 03 (BandMasters) | TRUE | None | None | FALSE | b |
| 00 00 00 09 00 00 80 02 | "BandMaster1" | "BandMaster" | FALSE | 00 00 00 09 00 00 84 03 (BandMasters) | TRUE | None | None | FALSE | c |
| • | • | • | • | • | • | • | • | • | - |
| 00 00 00 09 00 00 84 00 | "BandMaster 1023" | "BandMaster" | FALSE | 00 00 00 09 00 00 84 03 (BandMasters) | TRUE | None | None | FALSE | d |
| 00 00 00 09 00 00 84 01 | "EraseMaster" | "EraseMaster" | FALSE | Null | TRUE | None | None | FALSE | e |
| 00 00 00 09 00 00 84 03 | "BandMasters" | "BandMasters" | TRUE | Null | TRUE | None | None | FALSE | f |

**Continued**

| ► | Operation | Credential | Response Sign | Response Exch | Clock Start | Clock End | Limit | Uses | Log | LogTo |
|---|---|---|---|---|---|---|---|---|---|---|
| a | None | Null | Null | Null | <date_0_value> | <date_0_value> | 0 | 0 | None | Null |
| b | Password | 00 00 00 0B 00 00 80 01 (BandMaster0 “PIN”) | Null | Null | <date_0_value> | <date_0_value> | 0 | 0 | None | Null |
| c | Password | 00 00 00 0B 00 00 80 02 (BandMaster1 “PIN”) | Null | Null | <date_0_value> | <date_0_value> | 0 | 0 | None | Null |
| - | • | • | • | • | • | • | • | • | • | • |
| d | Password | 00 00 00 0B 00 00 84 00 (BandMaster 1023 “PIN”) | Null | Null | <date_0_value> | <date_0_value> | 0 | 0 | None | Null |
| e | Password | 00 00 00 0B 00 00 84 01 (EraseMaster “PIN”) | Null | Null | <date_0_value> | <date_0_value> | 0 | 0 | None | Null |
| f | Password | Null | Null | Null | <date_0_value> | <date_0_value> | 0 | 0 | None | Null |

#### 8.3.1.1 BandMaster0 Authority

BandMaster0 SHALL be associated with the Global_Range; each additional Range SHALL have a dedicated BandMaster authority. The BandMaster authorities are defined in Table 14.

#### 8.3.1.2 EraseMaster Authority

The EraseMaster authority is defined in Table 14.

*Begin Informative Content*

The EraseMaster is a single dedicated authority used to reset one or more LBA Ranges by invoking the `Erase` method on the Locking object representing that Range. This is typically done for repurposing the storage device or for recovery of a LBA Range for which the unlock credential value is lost or blocked.

*End Informative Content*

### 8.3.2 Credential Table (C_PIN)

The Locking SP C_PINs are defined in Table 15. PIN values are a maximum size of 32-bytes each. The C_PIN table contains rows only for supported locking ranges.

**Table 15 - Locking C_PIN table**

| UID | Name | CommonName | PIN | CharSet | TryLimit | Tries | Persistence |
|---|---|---|---|---|---|---|---|
| 00 00 00 0B<br>00 00 80 01 | "BandMaster0" | "" | <PIN2_value> | Null | VU | VU | VU |
| 00 00 00 0B<br>00 00 80 02 | "BandMaster1" | "" | <PIN3_value> | Null | VU | VU | VU |
| • | • | • | • | • | • | • | • |
| 00 00 00 0B<br>00 00 84 00 | "BandMaster<br>1023" | "" | <PIN1025_value> | Null | VU | VU | VU |
| 00 00 00 0B<br>00 00 84 01 | "EraseMaster" | "" | <PIN1026_value> | Null | VU | VU | VU |

The PIN value for all ranges SHALL be set to the MSID Credential value at manufacturing time.

### 8.3.3 Access Control Elements

The Locking SP Access Control Elements (ACEs) are defined in Table 16. The ACE table contains rows only for supported locking ranges.

**Table 16 - Locking SP ACE table**

| UID | Name | Common Name | Boolean Expr | Row Start | Row End | Col Start | Col End |
|---|---|---|---|---|---|---|---|
| 00 00 00 08 00 00 00 01 | "Anybody" | "" | 00 00 00 09 00 00 00 01 (Anybody) | Null | Null | "" | "" |
| 00 00 00 08 00 00 80 01 | "BandMaster0" | "" | 00 00 00 09 00 00 80 01 (BandMaster0) | Null | Null | "" | "" |
| 00 00 00 08 00 00 84 01 | "BandMaster0_ SetSelf" | "" | 00 00 00 09 00 00 80 01 (BandMaster0) | Null | Null | "PIN" | "PIN" |
| 00 00 00 08 00 00 88 01 | "BandMaster0_ SetBand" | "" | 00 00 00 09 00 00 80 01 (BandMaster0) | Null | Null | "ReadLock Enabled" | "LockOn Reset" |
| 00 00 00 08 00 00 80 02 | "BandMaster1" | "" | 00 00 00 09 00 00 80 02 (BandMaster1) | Null | Null | "" | "" |
| 00 00 00 08 00 00 84 02 | "BandMaster1_ SetSelf" | "" | 00 00 00 09 00 00 80 02 (BandMaster1) | Null | Null | "PIN" | "PIN" |
| 00 00 00 08 00 00 88 02 | "BandMaster1_ SetBand" | "" | 00 00 00 09 00 00 80 02 (BandMaster1) | Null | Null | "RangeStart" | "LockOn Reset" |
| • | • | • | • | • | • | • | • |
| 00 00 00 08 00 00 84 00 | "BandMaster1023" | "" | 00 00 00 09 00 00 84 00 (BandMaster1023 | Null | Null | "" | "" |
| 00 00 00 08 00 00 88 00 | "BandMaster1023_ SetSelf" | "" | 00 00 00 09 00 00 84 00 (BandMaster1023 | Null | Null | "PIN" | "PIN" |
| 00 00 00 08 00 00 8C 00 | "BandMaster1023_ SetBand" | "" | 00 00 00 09 00 00 84 00 (BandMaster1023 | Null | Null | "RangeStart" | "LockOn Reset" |
| 00 00 00 08 00 00 8C 01 | "EraseMaster" | "" | 00 00 00 09 00 00 84 01 (EraseMaster) | Null | Null | "" | "" |
| 00 00 00 08 00 00 8C 02 | "EraseMaster_ SetSelf" | "" | 00 00 00 09 00 00 84 01 (EraseMaster) | Null | Null | "PIN" | "PIN" |
| 00 00 00 08 00 00 8C 05 | "AnyMaster" | "" | 00 00 00 09 00 00 84 03 or 00 00 00 09 00 00 84 01 (BandMasters or EraseMaster) | Null | Null | "" | "" |
| 00 00 00 08 00 00 8C 06 | "BandMasters" | "" | 00 00 00 09 00 00 84 03 (BandMasters) | Null | Null | "" | "" |
| 00 00 00 08 00 02 00 01 | "Anybody_ GetBand" | "" | 00 00 00 09 00 00 00 01 (Anybody) | Null | Null | "UID" | "ActiveKey" |
| 00 00 00 08 00 03 BF FF | "Get_K_AES _Mode" | "" | 00 00 00 09 00 00 00 01 (Anybody) | Null | Null | "Mode" | "Mode" |

### 8.3.4 AccessControl table

The Locking SP access control definitions in Table 17 are required for compliance to this specification. The text in parentheses is presented only for clarification. The CommonName fields are not required to be accessible by this specification and MAY exceed the length limit for names to clarify the access control. The AccessControl table contains rows only for supported locking ranges.

**Table 17 - Locking SP AccessControl table**

| Row Number | UID | InvokingID | MethodID | CommonName | ACL | Log | AddACE ACL | RemoveACE ACL | GetACL ACL |
|---|---|---|---|---|---|---|---|---|---|
| VU | VU | 00 00 00 00<br>00 00 00 01<br>(ThisSP) | 00 00 00 06<br>00 00 00 0C<br>(Authenticate) | Anybody-Authenticate-LockingSP | 00 00 00 08<br>00 00 00 01<br>(Anybody) | None | Null | Null | 00 00 00 08<br>00 00 00 01<br>(Anybody) |
| VU | VU | 00 00 00 09<br>00 00 00 00<br>(Authority table) | 00 00 00 06<br>00 00 00 08<br>(Next) | Anybody-Next-Authority table | 00 00 00 08<br>00 00 00 01<br>(Anybody) | None | Null | Null | 00 00 00 08<br>00 00 8C 05<br>(AnyMaster) |
| VU | VU | 00 00 00 09<br>00 00 00 01<br>(Anybody Authority Object) | 00 00 00 06<br>00 00 00 06<br>(Get) | Anybody-Get-Anybody Authority object | 00 00 00 08<br>00 00 00 01<br>(Anybody) | None | Null | Null | 00 00 00 08<br>00 00 00 01<br>(Anybody) |
| VU | VU | 00 00 00 09<br>00 00 84 03<br>(BandMasters Authority Object) | 00 00 00 06<br>00 00 00 06<br>(Get) | AnyMaster-Get-BandMasters Authority object | 00 00 00 08<br>00 00 8C 05<br>(AnyMaster) | None | Null | Null | 00 00 00 08<br>00 00 8C 05<br>(AnyMaster) |
| VU | VU | 00 00 00 09<br>00 00 84 01<br>(EraseMaster Authority Object) | 00 00 00 06<br>00 00 00 06<br>(Get) | EraseMaster-Get-Erase Authority object | 00 00 00 08<br>00 00 8C 01<br>(EraseMaster) | None | Null | Null | 00 00 00 08<br>00 00 8C 01<br>(EraseMaster) |
| VU | VU | 00 00 00 09<br>00 00 80 01<br>(BandMaster0 Authority Object) | 00 00 00 06<br>00 00 00 06<br>(Get) | BandMaster0-Get-BandMaster0 Authority object | 00 00 00 08<br>00 00 80 01<br>(BandMaster0) | None | Null | Null | 00 00 00 08<br>00 00 80 01<br>BandMaster0 |
| . | . | . | . | . | . | None | . | . | . |
| VU | VU | 00 00 00 09<br>00 00 84 00<br>(BandMaster) 1023 Authority Object) | 00 00 00 06<br>00 00 00 06<br>(Get) | BandMaster1023-Get-BandMaster1023 Authority object | 00 00 00 08<br>00 00 84 00<br>(BandMaster 1023) | None | Null | Null | 00 00 00 08<br>00 00 84 00<br>(BandMaster1023) |
| VU | VU | 00 00 00 0B<br>00 00 00 00<br>(C_PIN table) | 00 00 00 06<br>00 00 00 08<br>(Next) | AnyMaster-Next-C_PIN table | 00 00 00 08<br>00 00 8C 05<br>(AnyMaster) | None | Null | Null | 00 00 00 08<br>00 00 8C 05<br>(AnyMaster) |
| VU | VU | 00 00 00 0B<br>00 00 84 01<br>(EraseMaster C_PIN object) | 00 00 00 06<br>00 00 00 07<br>(Set) | EraseMaster_SetSelf-Set-EraseMaster C_PIN object | 00 00 00 08<br>00 00 8C 02<br>(EraseMaster_SetSelf) | None | Null | Null | 00 00 00 08<br>00 00 8C 01<br>(EraseMaster) |
| VU | VU | 00 00 00 0B<br>00 00 80 01<br>(BandMaster0 C_PIN object) | 00 00 00 06<br>00 00 00 07<br>(Set) | BandMaster0_SetSelf-Set-BandMaster0 C_PIN object | 00 00 00 08<br>00 00 84 01<br>(BandMaster0_SetSelf) | None | Null | Null | 00 00 00 08<br>00 00 80 01<br>(BandMaster0) |
| . | . | . | . | . | . | . | . | . | . |

| Row Number | UID | InvokingID | MethodID | CommonName | ACL | Log | AddACE ACL | RemoveACE ACL | GetACL ACL |
|---|---|---|---|---|---|---|---|---|---|
| VU | VU | 00 00 00 0B 00 00 84 00 (BandMaster1023 C_PIN object) | 00 00 00 06 00 00 00 07 (Set) | BandMaster1023_ SetSelf-Set-BandMaster1023 C_PIN object | 00 00 00 08 00 00 88 00 (BandMaster 1023_SetSelf) | None | Null | Null | 00 00 00 08 00 00 84 00 (BandMaster1023) |
| VU | VU | 00 00 08 01 00 00 00 01 (LockingInfo table) | 00 00 00 06 00 00 00 06 (Get) | Anybody-Get-LockingInfo table | 00 00 00 08 00 00 00 01 (Anybody) | None | Null | Null | 00 00 00 08 00 00 8C 05 (AnyMaster) |
| VU | VU | 00 00 08 02 00 00 00 00 (Locking table) | 00 00 00 06 00 00 00 08 (Next) | AnyMaster-Next-Locking table | 00 00 00 08 00 00 8C 05 (AnyMaster) | None | Null | Null | 00 00 00 08 00 00 8C 05 (AnyMaster) |
| VU | VU | 00 00 08 02 00 00 00 01 (Global_Range Locking object) | 00 00 00 06 00 00 00 07 (Set) | BandMaster 0_SetBand-Set-Global_Range Locking object | 00 00 00 08 00 00 88 01 (BandMaster 0_SetBand) | None | Null | Null | 00 00 00 08 00 00 80 01 (BandMaster0) |
| . | . | . | . | . | . | . | . | . | . |
| VU | VU | 00 00 08 02 00 00 04 00 (Band1023 _ Locking object) | 00 00 00 06 00 00 00 07 (Set) | BandMaster1023_ SetBand-Set-Band1023 _ Locking object | 00 00 00 08 00 00 8C 00 (BandMaster 1023_SetBand)) | None | Null | Null | 00 00 00 08 00 00 84 00 (BandMaster1023) |
| VU | VU | 00 00 08 02 00 00 00 01 (Global_Range Locking object) | 00 00 00 06 00 00 00 06 (Get) | Anybody _GetBand-Get-Global_Range Locking object | 00 00 00 08 00 02 00 01 (Anybody _GetBand) | None | Null | Null | 00 00 00 08 00 00 80 01 (BandMaster0) |
| . | . | . | . | . | . | . | . | . | . |
| VU | VU | 00 00 08 02 00 00 04 00 (Band1023 _ Locking object) | 00 00 00 06 00 00 00 06 (Get) | Anybody _GetBand-Get-Band1023 _ Locking object | 00 00 00 08 00 02 00 01 (Anybody _GetBand) | None | Null | Null | 00 00 00 08 00 00 84 00 (BandMaster1023) |
| VU | VU | 00 00 08 02 00 00 00 01 (Global_Range Locking object) | 00 00 00 06 00 00 08 03 (Erase) | EraseMaster-Erase-Global_Range Locking object | 00 00 00 08 00 00 8C 01 (EraseMaster) | None | Null | Null | 00 00 00 08 00 00 80 01 (BandMaster0) |
| . | . | . | . | . | . | . | . | . | . |
| VU | VU | 00 00 08 02 00 00 04 00 (Band1023 _Range Locking object) | 00 00 00 06 00 00 08 03 (Erase) | EraseMaster-Erase-Band1023 _Range Locking object | 00 00 00 08 00 00 8C 01 (EraseMaster) | None | Null | Null | 00 00 00 08 00 00 84 00 (BandMaster1023) |
| VU | VU | 00 00 80 01 00 00 00 00 (DataStore) | 00 00 00 06 00 00 00 06 (Get) | Anybody-Get-DataStore | 00 00 00 08 00 00 00 01 (Anybody) | None | Null | Null | 00 00 00 08 00 00 00 01 (Anybody) |
| VU | VU | 00 00 80 01 00 00 00 00 (DataStore) | 00 00 00 06 00 00 00 07 (Set) | BandMasters-Set-DataStore | 00 00 00 08 00 00 8C 06 (BandMasters) | None | Null | Null | 00 00 00 08 00 00 8C 06 (BandMasters) |
| VU | VU | 00 00 00 00 00 00 00 01 (ThisSP) | 00 00 00 06 00 00 06 01 (Random) | Anybody-Random | 00 00 00 08 00 00 00 01 (Anybody) | None | Null | Null | 00 00 00 08 00 00 00 01 (Anybody) |
| VU | VU | 00 00 08 05 00 00 00 01 (Global_Range_ AES_128) | 00 00 00 06 00 00 00 06 (Get) | Anybody-Get-Global_Range_ AES128 | 00 00 00 08 00 03 BF FF (Get_K_AES_ Mode) | None | Null | Null | 00 00 00 08 00 00 00 01 (Anybody) |

| Row Number | UID | InvokingID | MethodID | CommonName | ACL | Log | AddACE ACL | RemoveACE ACL | GetACL ACL |
|---|---|---|---|---|---|---|---|---|---|
| . | . | • | . | . | . | . | . | . | . |
| VU | VU | 00 00 08 05 00 00 04 00 (Band1023_ AES_128) | 00 00 00 06 00 00 00 06 (Get) | Anybody-Get-Band1023_ AES128 | 00 00 00 08 00 03 BF FF (Get_K_AES_ Mode) | None | Null | Null | 00 00 00 08 00 00 00 01 (Anybody) |
| VU | VU | 00 00 08 06 00 00 00 01 (Global_Range_ AES_256) | 00 00 00 06 00 00 00 06 (Get) | Anybody-Get-Global_Range_ AES256 | 00 00 00 08 00 03 BF FF (Get_K_AES_ Mode) | None | Null | Null | 00 00 00 08 00 00 00 01 (Anybody) |
| . | . | • | . | . | . | . | . | . | . |
| VU | VU | 00 00 08 06 00 00 04 00 (Band1023_ AES_256) | 00 00 00 06 00 00 00 06 (Get) | Anybody-Get-Band1023_ AES256 | 00 00 00 08 00 03 BF FF (Get_K_AES_ Mode) | None | Null | Null | 00 00 00 08 00 00 00 01 (Anybody) |

### 8.3.5 Locking Objects Definition

The LBA Range (“Locking”) objects are defined in Table 18. The Locking table SHALL at minimum have a single Locking object (“Global_Range”) and MAY contain one or more additional Locking objects. The implementation of greater than 1023 locking ranges is beyond the scope of this specification.

**Table 18 - Locking SP Locking table**

| ► | UID | Name | Common Name | Range Start | Range Length | Read Lock Enabled | Write Lock Enabled | Read Locked | Write Locked | LockOn Reset |
|---|---|---|---|---|---|---|---|---|---|---|
| a | 00 00 08 02 00 00 00 01 | "Global_Range" | "Locking" | 0 | 0 | FALSE | FALSE | FALSE | FALSE | Power Cycle |
| b | 00 00 08 02 00 00 00 02 | "Band1" | "Locking" | 0 | 0 | FALSE | FALSE | FALSE | FALSE | Power Cycle |
| - | • | • | • | • | • | • | • | • | • | • |
| c | 00 00 08 02 00 00 04 00 | "Band1023" | "Locking" | 0 | 0 | FALSE | FALSE | FALSE | FALSE | Power Cycle |

**Continued**

| ► | ActiveKey [1] | NextKey | ReEncrypt State | ReEncrypt Request | AdvKey Mode | Verf Mode | ContOn Reset | Last ReEncrypt LBA | Last ReEnc Stat | General Status |
|---|---|---|---|---|---|---|---|---|---|---|
| a | 00 00 08 05/6 00 00 00 01 | VU | VU | VU | VU | VU | VU | VU | VU | VU |
| b | 00 00 08 05/6 00 00 00 02 | VU | VU | VU | VU | VU | VU | VU | VU | VU |
| - | • | • | • | • | • | • | • | • | • | • |
| c | 00 00 08 05/6 00 00 04 00 | VU | VU | VU | VU | VU | VU | VU | VU | VU |

[1] UID of K_AES_128 or K_AES_256 table row for the band

The TPer SHALL support the LockOnReset column values of "[ 0 ]" ("Power Cycle") and "[ ]" (the empty set).

*Begin Informative Content*

Changing the size and/or location of LBA Ranges will result in loss of data.

*End Informative Content*

### 8.3.6 K_AES_128 Table

The encryption keys and mode are defined in Table 19 for Locking Objects using the AES 128 encryption algorithm.

**Table 19 – K_AES _128 table**

| UID | Name | CommonName | Key | Mode |
|---|---|---|---|---|
| 00 00 08 05 00 00 00 01 | "Global_Range-_AES_128" | "" | VU | VU |
| 00 00 08 05 00 00 00 02 | "Band1 - AES_128" | "" | VU | VU |
| • | • | • | • | • |
| 00 00 08 05 00 00 04 00 | "Band1023_ AES_128" | "" | VU | VU |

### 8.3.7 K_AES_256 Table

The encryption keys and mode are defined in Table 20 for Locking Objects using the AES 256 encryption algorithm.

**Table 20 - K_AES _256 table**

| UID | Name | CommonName | Key | Mode |
|---|---|---|---|---|
| 00 00 08 06 00 00 00 01 | "Global_Range-_AES_256" | "" | VU | VU |
| 00 00 08 06 00 00 00 02 | "Band1 - AES_256" | "" | VU | VU |
| • | • | • | • | • |
| 00 00 08 06 00 00 04 00 | "Band1023_ AES_256" | "" | VU | VU |

### 8.3.8 LockingInfo table

The LockingInfo information is defined in Table 21

**Table 21 - LockingInfo table**

| Row Number | UID | Name | Version | Encrypt Support | MaxRanges | Max ReEncryptions | Keys AvailableCfg |
|---|---|---|---|---|---|---|---|
| 1 | 00 00 08 01<br>00 00 00 01 | VU | VU | Media Encryption | VU | VU | VU |

### 8.3.9 DataStore table

The DataStore table provides a generic non-volatile storage in the TPer for host access and modification. TCG access control enforces write access authorization to only bandmasters but allows unconstrained read access.

Table UID: 00 00 80 01 00 00 00 00

Name: “DataStore”

Type: Byte

Size: 1024 bytes min

*Begin Informative Content*

Use cases exist in which the host needs to store a limited amount of data in a TPer. Such a use case is one where the drive is moved between different hosts and the new host needs reference information to subsequently to get the drive unlock keys.

*End Informative Content*

The DataStore byte table is defined in Table 22.

**Table 22 - DataStore table**

| Row Number | Byte Value |
|---|---|
| 0 | Byte_0 |
| • | • |
| n | Byte_n |

All bytes of the DataStore table SHALL be set to a value of 0x00 at manufacturing time.

### 8.3.10 Device Behavior Under Locking

The storage device SHALL terminate read commands that address consecutive LBAs in one or more LBA ranges for which ReadLockEnabled=True and ReadLocked=True.

The storage device SHALL terminate write commands that address consecutive LBAs in one or more LBA ranges for which WriteLockEnabled=True and WriteLocked=True.

When a command is terminated due to range locking, the storage device SHALL terminate the command with a “Data Protection Error” as defined in the Storage Interface Interactions Specification (see [7]).

If the storage device receives a read or write command that addresses consecutive LBAs in more than one LBA range and the LBA ranges are not locked, the storage device SHALL either:

- Process the data transfer without an error,

or

- Terminate the command with “Other Invalid Command Parameter” as defined in the Storage Interface Interactions Specification (see [7]).

The storage device's range crossing behavior SHALL be reported in Level 0 Discovery (see the 'Range Crosing' bit in section 3.6.2.3).

The device SHALL always abort the following commands:

For SCSI commands:

- READ LONG(10);
- READ LONG(16);
- WRITE LONG(10), (WR_UNCOR = 0);
- WRITE LONG(16), (WR_UNCOR = 0).

For ATA devices:

- READ LONG (obsolete);
- WRITE LONG (obsolete);
- SCT READ LONG;
- SCT WRITE LONG.

# 9 Appendix – MSID

## 9.1 Use of MSID

*Begin Informative Content*

The MSID Credential value is set at manufacturing time by the storage device vendor and is typically printed on the storage device label. It represents the device's initial storage device SID Credential value (owner's password) and is electronically readable over the interface by the host.

The MSID Credential value is used as an initial value for SID and an initial value for any BandMaster and EraseMaster Credentials on the Locking SP. During enrollment, the host reads the MSID value from the MSID Credential and uses that to authenticate and change all other Credential values on the storage device.

Having the MSID Credential value electronically available to the host constitutes a risk to the overall security of the device. It is therefore very important that the host executes a Take-Ownership scenario the moment a new device is inserted into the system, whereby it

- Invokes the `Erase()` method on every Range on the device;
- Replaces SID, any BandMaster, and EraseMaster Credentials with host known values.

If any of the above mentioned steps fails, then the host should reject the device from the system, as it could be compromised due to malicious behavior.

*End Informative Content*

# 10 Appendix –Core Specification proposals - Normative

## 10.1 Required Core Specification proposals

The proposals to change the TCG Storage Architecture Core Specification, Revision 0.9 and required for implementation of this specification are included in this appendix.

## 10.2 Core Specification change proposals for this SSC specification

- Transmitting 0-Length Byte Values
- Stream Type Removal
- Authenticate Method, Authority Parameter
- Rename Method Table Name
- Anybody Authority Authentication
- Symmetric Key ChallengeResponse
- Global Locking Range Identification
- Fixed Location Optional Parameters
- Authentication Within Transactions
- Zero-Length Locking Range Handling
- Next Method
- Synchronous Communications
- TypeOr Name Removal
- Interface Level Device Discovery
- ComPacket/Packet/Subpacket Header Alignment Proposal
- Session Manager Session Number
- Next Method Behavior Modification
- K_AES media encryption key tables
- Protocol Stack Reset Command
- Properties Clarification
- Inactive or Unsupported ComID in CDB Proposal

### 10.2.1 Transmitting 0-Length Byte Values 0

*Start Proposal*

# Transmitting 0-Length Byte Values

Author: Jason Cox

Revision:

0.1 07.31.07 Start of draft

# 1 Goal

The goal of this proposal is:

- Incorporate a mechanism into the Core Specification messaging structure that allows a 0-length byte value to be transmitted in the message stream.

# 2 Proposal

Currently, there is no mechanism in the TCG Storage Architecture Core Specification that allows a 0-length byte value to be transmitted as part of the message stream. However, the max bytes type permits such a value to be set as a valid column value.

In order to enable transmission of a 0-length byte value, one of the existing token structures must be modified to enable it to encode a 0-length data byte value.

The Short atom, as described in section 3.2.3.2.1.2 is the smallest of the atom types that is able to encode a bytes value. The capability to encode a 0-length byte value using this size atom is simply a matter of changing the description of the valid length of the atom.

Note that this proposal limits 0-length values for only bytes encoded with the Short atom. 0-length integers shall be illegal values.

# 2.1 Concepts

This section provides a detailed review of the concepts to be incorporated into the Core Specification.

## 2.1.1 Short Atom Token

Short atoms consist of a one-byte header and between 0 and 15 bytes of data.

**Table 01 Short Atom Description**

| Header (1 byte) | | | | | Data | | |
|---|---|---|---|---|---|---|---|
| Short Atom | | byte/ integer | sign/ continued | length | (0...15 bytes) | | |
| 1 | 0 | B | S | *n* *n* *n* *n* | *d* | ... | *d* |

The encoding is as follows:

**Table 02 Short Atom Encoding**

| | |
|---|---|
| Short Atom indicator | These two bits are set to 10b to indicate the atom is a short atom. |
| Byte/integer indicator | ***Value*** ***Interpretation***<br>0b The data bytes represent an integer value and the S bit indicates if that value is signed.<br>1b The data bytes represent a byte sequence and the S bit indicates whether or not this value is continued into another atom. |
| Sign/continued indicator | ***Value*** ***Interpretation***<br>0b The interpretation of the data depends on the byte/integer indicator bit.<br>B==0b The data is treated as unsigned integer data.<br>B==1b The data is either the complete byte sequence, or the final segment of a continued byte sequence.<br>1b The interpretation of the data depends on the byte/integer indicator bit.<br>B==0b The data is treated as signed integer data.<br>B==1b The data is a non-final segment of a multi-byte continued value. |

| | |
|---|---|
| Length | These bits specify the length of the following data byte sequence. The permitted range is from 0 to 15, inclusive. |

### 2.1.2 Example Encoding

The encoding of a 0-length byte value is displayed in the following table.

**Table 03 0-Length Byte Encoding**

| Header (1 byte) | | | | | | | | Data | | |
|---|---|---|---|---|---|---|---|---|---|---|
| Short Atom | | byte/ integer | sign/ continued | length | | | | (0...15 bytes) | | |
| 1 | 0 | B | S | *n* | *n* | *n* | *n* | *d* | ... | *d* |
| 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | *no data* | | |

A 0-length byte value is encoded using only 1 byte:

1 0 1 0 0 0 0 0

This value would be encoded in the token stream as:

A0

*End Proposal*

10.2.2 Stream Type Removal

*Start Proposal*

# Stream Type Removal Proposal

Author: Jason Cox

Revision:

| | | |
|---|---|---|
| 0.1 | 07.26.07 | Start of draft (Jason Cox) |
| 0.2 | 08.10.07 | Minor modifications of method signatures (Jason Cox) |
| 0.3 | 08.16.07 | Expanded ToC, added pagination, added Example section content, some other modifications |

# 1 Goal

The goals of this proposal are:

- To simplify the messaging structure and reduce messaging overhead by removing type identifiers for method parameters and results.
- To clarify the construction of method invocations and responses in the messaging stream.
- To reduce the complexity and size of the Type table by removing types currently associated with method parameters and results.
- To clarify column type requirements.

# 2 Proposal

The original goal of adding type identifiers to method parameters and results was to allow implementation of a first level type checking, possibly performed upon receipt of a method. To achieve that end, type identifiers were added to method parameters and results.

Each different type associated with a method parameter or result was added to the Type table, which resulted in the addition of many rows to the Type table. Due to the complexity of describing some method parameters and results in this manner, many of the added types are complicated combinations of other types.

This proposal defines the removal of type identifiers from the message stream (method parameters and results), while still enabling general identifiers for method parameter types through the use of basic interface types, and the use of types for columns. This includes the removal of method-associated types from the Type table.

In addition, this proposal will briefly outline a possible implementation to enable first level type checking that is possible even without the inclusion of type identifiers for method parameters and results.

## 2.1 Concepts

This section provides a detailed review of the concepts to be incorporated into the Core Specification.

### 2.1.1 Interface Types

Interface data types are introduced in Section 3.2 of the TCG Storage Core Architecture Specification. This introduction is divided into several parts:

- Pseudo-Code – this section describes the formatting used in method signatures in the Core Specification.
- Messaging Data Types – this section introduces two data types used for messaging – Named values and List values.
- Method Parameter/Column Value Typing and Encoding – this section introduces the mechanism defined by the Core Spec for inclusion of method parameter and result type identifiers in the message stream.

Because of the manner in which data is encoded and transferred across the interface, the actual types used in method parameter and result values can be described using a limited set of basic types: **Byte string values** and **N length integer values** (either signed or unsigned). All data is transferred across the interface as one of these two fundamental types (bytes or integers).

- **Byte-string values** are a sequence of n bytes that can be used to represent strings, blobs, bit vectors, etc.
- **N byte integer values** are whole numbers that can be either signed or unsigned.

Due to the nature of method parameters and results, there are two additional constructs defined for messaging that serve as grouping mechanisms for the fundamental types: **Named values** and **List values**.

- **Named values.** The name (a byte-value) followed by its value (any messaging type). Named values are used to send the optional parameters in method calls.
- **List values.** Zero or more values of some type, grouped into an ordered list. List values are used to encode method parameter lists and return results.

Method parameters and results are made up of byte and (signed or unsigned) integer values that can be grouped using Named values and List values.

## 2.1.2 Abstract Types

Abstract types are representations of grouped interface types, or interface types that have limits on their legal values. These representations are used primarily for documentation purposes, as part of the pseudo-code method signatures to simplify the description of those methods.

Abstract types do not affect the operation or regular encoding of a method, nor are they used as column types or represented in the Type table (though they may resemble some of these types in structure, name, or both). The primary goals of the abstract type constructs are to simplify the pseudo-code description of the methods themselves, and to provide insight into grouping using the List and Named value tokens introduced previously.

### 2.1.2.1 Abstract Types definitions

The following sections describe the pseudo-code parameters that each of these abstract types represent when they appear in a pseudo-code method signature.

#### *2.1.2.1.1 access_control_list*

An access_control_list is a list of uidrefs, specifically uidrefs to objects in the ACE table. The length of the list is implementation/SSC-specific.

Format:

```
[ uidref ... ]
```

#### *2.1.2.1.2 boolean*

This abstract type is similar to an enumeration column type, and has a valid range of the integer 0 to the integer 1, where 0 is used to represent "False" and 1 is used to represent "True".

Format:

```
uinteger
```

In the messaging stream, "False" will be represented as `0x00` and "True" will be represented as `0x01`.

#### *2.1.2.1.3 cell_block*

This type represents a grouping of Named values that are used to identify a portion of a table. In messaging, this grouping is enclosed by List value delimiters, and each component is enclosed by Named value delimiters.

Because this is a group of Named values, its separate components are optional. However, there are default requirements if components are omitted. These requirements are as follows:

- Table – this Named value has the Name "Table" and a value that is a uid to a table.
    - If the value with Name "Table" is omitted, then the operation defaults to the table upon which the method was invoked.
    - Table shall be omitted if the method was invoked to operate on an object.
- startRow – this Named value has the Name "startRow". This Named value type can be assigned one of two values – either a uid of an object or a RowNumber that corresponds to the RowNumber value of an Array table row. Only one of these two values will appear in the messaging stream. The "typeOr" identifier and accompanying curly brackets ("{", "}") have no effect on the values as represented in the message.
    - If the value with Name "startRow" is omitted and the method is invoked to operate on a table, then the operation defaults to the first row of the table.
    - The value with Name "startRow" may be omitted if the method is invoked to operate on an object. If it is not omitted, it shall be the uid of the object on which the method is to operate, and shall be the same as the value assigned to endRow.

  - If both the value with Name "startRow" and the value with Name "endRow" are included in the type parameterization, then the value with Name "startRow" shall have the same type (uid or uinteger) as the value with Name "endRow".

- endRow – this Named value has the Name "endRow". This Named value type can be assigned one of two values – either a uid of an object or a RowNumber that corresponds to the RowNumber value of an Array table row. Only one of these two values will appear in the messaging stream. The "typeOr" identifier and accompanying curly brackets ("{", "}") have no effect on the values as represented in the message.
  - If the value with Name "endRow" is omitted and the method is invoked to operate on a table, then the operation defaults to the last row of the table.
  - The value with Name "endRow" shall be omitted if the method is invoked to operate on an object. If it is not omitted, it shall be the uid of the object on which the method is to operate, and shall be the same as the value assigned to startRow.
  - If both the value with Name "startRow" and the value with Name "endRow" are included in the type parameterization, then the value with Name "endRow" shall have the same type (uid or uinteger) as the value with Name "startRow".
- startColumn – this Named value has the Name "startColumn". This Named value type has a max bytes value that is represented by here using the name abstract type.
  - If the value with Name "startColumn" is omitted, then the operation defaults to the first column of the table or object.
- endColumn – this Named value has the Name "endColumn". This Named value type has a max bytes value that is represented by here using the name abstract type.
  - if the value with Name "endColumn" is omitted, then the operation defaults to the last column of the table or object.

Format:

```
[ Table = uid, startRow = typeOr { UID = uid, Row = RowNumber }, endRow = typeOr
{ UID = uid, Row = RowNumber }, startColumn = name, endColumn = name ]
```

#### *2.1.2.1.4 clock_time*

This type represents a grouping of Named values that are used to identify time values, and is similar to the column type of the same name. In messaging, this grouping is enclosed by List value delimiters, and each component is enclosed by Named value delimiters.

Because this is a group of Named values, its separate components are optional. Components that are omitted are considered to have a value of 0.

The components are as follows:

- Year – this Named value has the Name "Year" and a value that is implicitly defined as being of uinteger of size 2. This Named value abstract type represents the year in a timestamp. Valid values are unsigned integers ranging from 1970 to 9999
- Month – this Named value has the Name "Month" and a value that is implicitly defined as being of uinteger of size 2. This Named value abstract type represents the month in a timestamp. Valid values are unsigned integers ranging from 1 to 12, which correspond to the months of the year as follows:
  - January = 1 (0x01)
  - February = 2 (0x02)
  - March = 3 (0x03)

- April = 4 (0x04)
- May = 5 (0x05)
- June = 6 (0x06)
- July = 7 (0x06)
- August = 8 (0x08)
- September = 9 (0x09)
- October = 10 (0x0A)
- November = 11 (0x0B)
- December = 12 (0x0C)

- Day – this Named value has the Name "Day" and a value that is implicitly defined as being of uinteger of size 1. This Named value abstract type represents the day of the month in a timestamp. Valid values are unsigned integers ranging from 1 to 31.
- Hour – this Named value has the Name "Hour" and a value that is implicitly defined as being of uinteger size 1. This Named value abstract type represents the hour of the day in a timestamp. Valid values are unsigned integers ranging from 0 to 23.
- Minute – this Named value has the Name "Minute" and a value that is implicitly defined as being of uinteger size 1. This Named value abstract type represents the minute of the hour in a timestamp. Valid values are unsigned integers ranging from 0 to 59.
- Seconds – this Named value has the Name "Seconds" and a value that is implicitly defined as being of uinteger size 1. This Named value abstract type represents the second of the minute in a timestamp. Valid values are unsigned integers ranging from 0 to 59.
- Fraction – this Named value has the Name "Fraction" and a value that is implicitly defined as being of uinteger size 2. This Named value abstract type represents fractions of a second in a timestamp, measured in milliseconds. Valid values are unsigned integers ranging from 0 to 999.

Format:

```
[ Year = uinteger, Month = uinteger, Day = uinteger, Hour = uinteger, Minute = uinteger, Second = uinteger, Fraction = uinteger ]
```

#### *2.1.2.1.5 columns*

This is a list of two lists of Named values, where the List value delimiters enclose the entire list and both subordinate lists, and the Named value delimiters enclose each component of each subordinate list.

The Named values in both subordinate lists represent column names and their associated types. Each Name portion of the Named value will be the host-supplied name of a column to be created in the new table, and the associated value is the uidref to the type to be assigned for that column.

The ordering of and within the subordinate lists determines the ordering of the columns and the indexed columns in the newly created table. The first subordinate list contains the columns whose combination of values is required to be unique within the table. The columns described within that list are ordered "first".

The second subordinate list contains the rest of the columns of the table. The columns described within the second subordinate list are ordered according to their order in the list, all of which come after the columns defined in the first subordinate list.

For Byte tables, the external grouping will be empty. For tables with no host-assigned indexed columns, the first subordinate list will be empty. For tables with no host assigned non-indexed columns, the second list will be empty. For tables with no host assigned columns, both lists will be empty.

Format:

```
      [ IsIndex = [ ColumnName = uidref { TypeUID } ... ], IsColumn = [ ColumnName =
uidref { TypeUID } ... ] ]
```

Byte table format pseudo-code example:

```
      [ ]
```

Array/Object table with no indexed columns pseudo-code example:

```
      [ IsIndex = [ ]   IsColumn = [ ColumnName1 = uidref1 ColumnName2 = uidref2
ColumnName3 = uidref3 ] ]
```

#### *2.1.2.1.6 date*

This type represents a grouping of Named values that are used to identify time values, and is similar to the column type of the same name. In messaging, this grouping is enclosed by List value delimiters, and each component is enclosed by Named value delimiters.

Because this is a group of Named values, its separate components are optional. Components that are omitted are considered to have a value of 0.

The components are as follows:

- Year – this Named value has the Name "Year" and a value that is implicitly defined as being of uinteger of size 2. This Named value abstract type represents the year in a timestamp. Valid values are unsigned integers ranging from 1970 to 9999
- Month – this Named value has the Name "Month" and a value that is implicitly defined as being of uinteger of size 2. This Named value abstract type represents the month in a timestamp. Valid values are unsigned integers ranging from 1 to 12, which correspond to the months of the year as follows:
    - January = 1 (0x01)
    - February = 2 (0x02)
    - March = 3 (0x03)
    - April = 4 (0x04)
    - May = 5 (0x05)
    - June = 6 (0x06)
    - July = 7 (0x06)
    - August = 8 (0x08)
    - September = 9 (0x09)
    - October = 10 (0x0A)
    - November = 11 (0x0B)
    - December = 12 (0x0C)
- Day – this Named value has the Name "Day" and a value that is implicitly defined as being of uinteger of size 1. This Named value abstract type represents the day of the month in a timestamp. Valid values are unsigned integers ranging from 1 to 31.

Format:

```
      [ Year = uinteger, Month = uinteger, Day = uinteger ]
```

#### *2.1.2.1.7 hash_protocol*

This abstract type is similar to an enumeration column type, and is used to identify a selected hash algorithm. This type has valid values in the range of integers from 0-15. These integers have the following values:

- 0 = none
- 1 = SHA 1
- 2 = SHA 256
- 3 = SHA 384
- 4 = SHA 512
- 5-15 = reserved

Format:

```
uinteger
```

In the messaging stream, these values will be represented as follows:

- 0x00 represents none
- 0x01 represents SHA 1
- 0x02 represents SHA 256
- 0x03 represents SHA 384
- 0x04 represents SHA 512
- 0x05 – 0x0F are reserved.

#### *2.1.2.1.8 lag*

This type represents a grouping of two Named value pairs, used to describe seconds and milliseconds, and is similar to the column type of the same name. The components are encapsulated with the interface type List value delimiters ("[", "]"). Each of the components is encapsulated with the Named value delimiters. The components are optional.

The components are as follows:

- Seconds – this component is a Named value pair with a Name of "Seconds" and a value of uinteger. This value has an implicit size requirement of 2.
- Milliseconds – this component is a Named value pair with a Name of "Milliseconds" and a value of uinteger. This value has an implicit size requirement of 2.

Format:

```
[ Seconds = uinteger, Milliseconds = uinteger ]
```

#### *2.1.2.1.9 name*

This type is a representation of the max bytes type, and in most methods in which it is used it is assigned to parameters that are associated with a table's Name column or CommonName column. As such, it has an implicit size restriction of 32 bytes.

Format:

```
bytes
```

#### *2.1.2.1.10 package*

This abstract type is a bytes type that is parsed according to the rules described in the Core Specification section on GetPackage and SetPackage.

Format:

```
Bytes
```

#### *2.1.2.1.11 package_purpose*

This abstract type is similar to an enumeration column type, and is used to identify a selected purpose for the package in which it is being included. This type has valid values in the range of integers from 1-32. These integers have the following values:

- 1 = Issuance
- 2 = Key Wrapping
- 3 = Backup
- 4-32 = reserved

Format:

```
uinteger
```

In the messaging stream, these values will be represented as follows:

- 0x01 represents Issuance
- 0x02 represents Key Wrapping
- 0x03 represents Backup
- 0x04 – 0x20 are reserved.

#### *2.1.2.1.12 ref*

The ref abstract type represents a reference to a table row that is expressed using a uinteger type with a size of 8, and corresponds to a row's RowNumber column value.

In the pseudo-code method signatures, the ref abstract type is often followed by curly brackets ("{", "}") that are used to define the limitation of a valid value for that ref. These valid values are typically represented as a reference to a specific table, which indicates that to ultimately be considered valid, the ref must be to a row in that table.

Limitations expressed with curly brackets have no effect on the appearance of the associated ref value as it appears in the message stream. Because this abstract type describes the inclusion of a RowNumber, it represents a uinteger value that has an implicit size restriction of 8 bytes in the uinteger value.

Format:

```
uinteger
```

#### *2.1.2.1.13 row_address*

This abstract type is used to describe a parameter that can be either a ref or a uidref. It is similar to the alternative column type. For additional information on the component types (ref and uidref), see their respective entries in this section.

Only one of these two values will appear in the messaging stream. The "typeOr" identifier and accompanying curly brackets ("{", "}") have no effect on the values as represented in the message.

Format

```
typeOr { RowAddress = ref, UIDAddress = uidref }
```

In the message stream itself, the value will one of the following:

- `ref`
- `uidref`

#### *2.1.2.1.14 row_date*

This type represents a list of lists of Named values. Each interior list represents a row, so there are multiple interior lists (a list of lists). The Named values represent column names and the values to be associated with them. The number of interior lists (i.e. the number of rows that may be represented by this type "at one time") may be limited by SSC or implementation.

Format:

```
[ [ ColumnName = Value ... ] ... ]
```

#### *2.1.2.1.15 table_kind*

This abstract type is similar to an enumeration column type, and is used to represent table types in the Table table. This type has valid values in the range of integers from 1-3. These integers have the following values:

- 1 = Object
- 2 = Array
- 3 = Byte

Format:

```
uinteger
```

In the messaging stream, these values will be represented as follows:

- 0x01 represents Object
- 0x02 represents Array
- 0x03 represents Byte

#### *2.1.2.1.16 table_sizes*

This abstract type defines a grouping of pairs of values that are table uidrefs and the size associated with that particular table. The grouping is a list of uidrefs and uintegers. The set of values are encapsulated by List value delimiters ("[", "]"). Inside the delimiters will be a series of one or more pairs of values. The first value in each pair is a uidref to a table descriptor object and the second value in each pair is a uinteger that describes the number of rows that may be additionally created for that table.

Format:

```
[ [ uidref {TableObjectUID}, uinteger ] ... ]
```

Pseudo-code example:

```
[ [ uidref1 uinteger1 ] [ uidref2 uinteger2 ] [ uidref3 uinteger3 ] ]
```

#### *2.1.2.1.17 uidref*

The uidref abstract type represents a uid of an object, table, or table row that is expressed using a bytes type with a size of 8, and corresponds to an object, table, or table row's UID column value.

In the pseudo-code method signatures, the uidref abstract type is often followed by curly brackets ("{", "}") that are used to define the limitation of a valid value for that uidref. These valid values are typically represented as requiring an object of a specific type. Limitations expressed with curly brackets have no effect on the appearance of the associated uid value as it appears in the message stream.

Because this abstract type describes the inclusion of a uid, it represents a bytes value that has an implicit size restriction, and that value shall always be 8 bytes long.

Format:

```
bytes
```

## 2.1.3 Method Signatures

Method signatures are presented in pseudo-code, which is used to describe types, method parameters, and snippets of code without having to use the byte encodings directly.

Methods are made up of two kinds of parameters: required and optional.

- Required parameters must come in the order given in the method signature, and must precede optional parameters. In the pseudo-code signature, required parameters are given expositional names for ease of reference. The right-hand portion of the parameter is the interface or abstract type that shall be used with that parameter.

  Required parameters are formatted as follows:

  - `Expositional-Name : Parameter-type`

- Optional parameters are not required to come in order, but each shall appear only once in a method invocation or the method shall fail and return a non-Success status. In the pseudo-code signature, optional parameters are given in the form of Named values except the right-hand portion of the parameter is the interface or abstract type that shall be used with that parameter. The Name (left-hand portion in the pseudo-code) shall be the name of the Named value type for that parameter when the method is invoked.

  Optional parameters are formatted as follows:

  - `Parameter-Name = Parameter-type`

The result portion of a method's signature are formatted similarly to the parameters, using the same conventions for results that are required to be returned for successful method invocations ("required results"), and results that may be returned only in certain situations ("optional results"). The result list of a failed method invocation should be empty.

Any appearance of "=" in a method's parameter list or result list (including in abstract type definitions) indicates the required use of an interface Named value, where the required Name is to the left of the "=" and the required value is to the right of the "=".

The components of an abstract typeOr alternative type used in method signature pseudo-code are always presented as Named value pairs. As such, each typeOr component will be represented on the interface as a Named value pair. Note that the typeOr itself may be an optional parameter or result, and as such this type could represent an instance of an embedded Named value pair (i.e. Name1 = Name2 = Value, where the value of Name2 is "Value" and the value of Name1 is "Name2 = Value").

Parameters typically have implicit size restrictions based on the table column that the particular parameter is modifying or to which it is referring.

Separating brackets ("[", "]") in method signatures are used to mark places in the stream where List values are used to encapsulate values. Commas (",") in the pseudo-code method signatures are used to separate items in a list. Ellipses in pseudo-code method signatures are used to indicate multiples of the immediately preceding type appears within the list (i.e. within the closest set of enclosing brackets).

In the pseudo-code, curly braces ("{", "}") are used to signify additional information regarding the type with which they are associated, but are not required to be checked as part of method parsing and do not affect the content or composition of the messaging stream.

### 2.1.3.1 Session Manager

#### *2.1.3.1.1 StartSession/SyncSession*

```
SMUID.StartSession [
```

```
            HostSessionID : uinteger,
            SPID : uidref {SPObjectUID},
            Write : boolean,
            HostChallenge = bytes,
            HostExchangeAuthority = uidref {AuthorityObjectUID},
            HostExchangeCert = bytes,
            HostSigningAuthority = uidref {AuthorityObjectUID},
            HostSigningCert = bytes,
            SessionTimeout = uinteger,
            TransTimeout = uinteger,
            InitialCredit = uinteger,
            SignedHash = bytes ]
      =>
      SMUID.SyncSession [
            HostSessionID : uinteger,
            SPSessionID : uinteger,
            SPChallenge = bytes,
            SPExchangeCert = bytes,
            SPSigningCert = bytes,
            TransTimeout = uinteger,
            InitialCredit = uinteger,
            SignedHash = bytes ]
```

#### *2.1.3.1.2 StartTrustedSession/SyncTrustedSession*

```
      SMUID.StartTrustedSession [
            HostSessionID : uinteger,
            SPSessionID : uinteger,
            HostResponse = bytes,
            HostEncryptSessionKey = bytes,
            HostIntegritySessionKey = bytes,
            SignedHash = bytes ]
      =>
      SMUID.SyncTrustedSession [
            HostSessionID : uinteger,
            SPSessionID : uinteger,
            SPResponse = bytes,
            SPEncryptSessionKey = bytes,
            SPIntegritySessionKey = bytes,
            SignedHash = bytes ]
```

#### *2.1.3.1.3 CloseSession*

```
SMUID.CloseSession [
        RemoteSessionNumber : uinteger,
        LocalSessionNumber : uinteger ]
```

### 2.1.3.2 Base Template

#### *2.1.3.2.1 DeleteSP*

```
SPUID.DeleteSP [ ]
=>
[ Result : boolean ]
```

#### *2.1.3.2.2 CreateTable*

```
SPUID.CreateTable [
        NewTableName : name,
        Kind : table_kind,
        GetSetACL : access_control_list,
        Columns : columns,
        MinSize : uinteger,
        MaxSize = uinteger,
        HintSize = uinteger,
        CommonName = name]
=>
[ UID : uid, Rows : uinteger ]
```

#### *2.1.3.2.3 Delete*

```
ObjectUID.Delete [ ]
=>
[ Result : boolean ]
```

#### *2.1.3.2.4 CreateRow*

```
TableUID.CreateRow [
        Row : row_data+ ]
=>
[ Result : typeOr { ArrayTable = list [ list [ ref, uidref ] ... ], ObjectTable =
list [ uidref ... ] } ]
```

#### *2.1.3.2.5 DeleteRow*

```
TableUID.DeleteRow [
        Where : row_address,
        Count = uinteger ]
=>
[ Result : boolean ]
```

#### *2.1.3.2.6 Get*

```
TableUID.Get [
```

```
ObjectUID.Get [
        Cellblock : cell_block ]
=>
[ Result : typeOr { Bytes = Bytes, RowValues = list [ list [ ColumnName = Value
... ] ... ] } ]
```

#### *2.1.3.2.7 Set*

```
TableUID.Set [
ObjectUID.Set [
        Where : cell_block,
        Values : typeOr { Bytes = bytes, RowValues = list [ list [ ColumnName =
Value ... ] ... ] ]
=>
[ Result : boolean ]
```

#### *2.1.3.2.8 Next*

```
TableUID.Next [
        Where = row_address,
        Count = uinteger ]
=>
[ Result : TypeOr { ArrayTable = list [ [ ref, uidref ] ... ], ObjectTable = list
[ uidref ... ] } ]
```

#### *2.1.3.2.9 GetFreeSpace*

```
SPUID.GetFreeSpace [ ]
=>
[ FreeSpace : uinteger, TableRows : table_sizes ]
```

#### *2.1.3.2.10 GetFreeRows*

```
TableObjectUID.GetFreeRows [ ]
=>
[ FreeRows : uinteger ]
```

#### *2.1.3.2.11 DeleteMethod*

```
MethodTableUID.DeleteMethod [
        InvokingID : uidref { SP/table/object },
        MethodID : uidref { MethodID } ]
=>
[ Result : boolean ]
```

#### *2.1.3.2.12 Authenticate*

```
SPUID.Authenticate [
```

```
            Authority : uidref { AuthorityObjectUID },
            Proof = bytes ]
      =>
      [ Result : typeOr { Success = boolean, Response = bytes } ]
```

#### 2.1.3.2.13 GetACL

```
      MethodTableUID.GetACL [
            InvokingID : uidref { SP/table/object },
            MethoID : uidref { MethodID } ]
      =>
      [ Result : access_control_list ]
```

#### 2.1.3.2.14 AddACE

```
      MethodTableUID.AddACE [
            InvokingID : uidref { SP/table/object },
            MethodID : uidref { MethodID },
            ACE : uidref { ACEObjectUID} ]
      =>
      [ Result : boolean ]
```

#### 2.1.3.2.15 RemoveACE

```
      MethodTableUID.RemoveACE [
            InvokingID : uidref { SP/table/object },
            MethodID : uidref { MethodID },
            ACE : uidref { ACEObjectUID} ]
      =>
      [ Result : boolean ]
```

#### 2.1.3.2.16 GenKey

```
      CredentialObjectUID.GenKey [
            PublicExponent = uinteger,
            PinLength = uinteger ]
      =>
      [ Result : boolean ]
```

### Admin Template

#### 2.1.3.3.1 IssueSP

```
      SPUID.IssueSP [
            SPName : name,
            Size : uinteger,
            Templates : list [ TemplateObjectUID ... ],
            AdminExch : bytes,
            Enabled : boolean ]
      =>
      [ UID : uid, Size : uinteger ]
```

### Clock Template

#### *2.1.3.4.1 GetClock*

```
ClockTimeTableUID.GetClock [ ]

=>

[ Kind : clock_kind, ExactTime : clock_time, LagTime : lag, MonotonicTime :
uinteger ]
```

#### *2.1.3.4.2 ResetClock*

```
ClockTimeTableUID.ResetClock [ ]

=>

[ Result : boolean ]
```

#### *2.1.3.4.3 SetClock*

```
ClockTimeTableUID.SetClockHigh [

      ExactTime : clock_time ]

=>

[ Result : boolean ]
```

#### *2.1.3.4.4 GetLagHigh*

```
ClockTimeTableUID.SetLagHigh [

      LagTime : lag ]

=>

[ Result : boolean, LowPreserved : boolean ]
```

#### *2.1.3.4.5 SetClockLow*

```
ClockTimeTableUID.SetClockLow [

      ExactTime : clock_time ]

=>

[ Result : boolean ]
```

#### *2.1.3.4.6 GetLagLow*

```
ClockTimeTableUID.SetLagHigh [

      LagTime : lag ]

=>

[ Result : boolean ]
```

#### *2.1.3.4.7 IncrementCounter*

```
ClockTimeUID.IncrementCounter [ ]

=>

[ MonotonicTime : uinteger ]
```

### 2.1.3.5 Crypto Template

#### *2.1.3.5.1 Random*

```
SPUID.Random[

      Count : uinteger,

      BufferOut = cell_block ]

=>

[ Result : bytes ]
```

#### 2.1.3.5.2 Stir

```
SPUID.Stir[
Value : typeOr { Input = integer, Internal = boolean } ]
=>
[ Result : boolean ]
```

#### 2.1.3.5.3 DecryptInit

```
CredentialObjectUID.DecryptInit [
IV = bytes ]
=>
[ Result : boolean]
```

#### 2.1.3.5.4 Decrypt

```
CredentialObjectUID.Decrypt [
Input : typeOr { Data = bytes, Buffer = cell_block },
BufferOut = cell_block ]
=>
[ Result : bytes ]
```

#### 2.1.3.5.5 DecryptFinalize

```
CredentialObjectUID.DecryptFinalize [ ]
=>
[ Result : bytes]
```

#### 2.1.3.5.6 EncryptInit

```
CredentialObjectUID.EncryptInit [
IV = bytes ]
=>
[ Result : boolean]
```

#### 2.1.3.5.7 Encrypt

```
CredentialObjectUID.Encrypt [
Input : typeOr { Data = bytes, Buffer = cell_block },
BufferOut = cell_block ]
=>
[ Result : bytes ]
```

#### 2.1.3.5.8 EncryptFinalize

```
CredentialObjectUID.EncryptFinalize [ ]
=>
[ Result : boolean]
```

#### 2.1.3.5.9 Sign

```
CredentialObjectUID.Sign
HashObjectUID.Sign[
Input : typeOr { Data = bytes, Buffer = cell_block },
BufferOut = cell_block ]
=>
[ Result : bytes ]
```

#### *2.1.3.5.10 Verify*

```
CredentialObjectUID.Verify

HashObjectUID.Verify[
        Input : typeOr { Data = bytes, Buffer = cell_block},
        Data : typeOr { Proof = bytes, ProofBuffer = cell_block } ]

=>

[ Result : boolean ]
```

#### *2.1.3.5.11 HashInit*

```
HashObjectUID.HashInit [
        BufferOut = cell_block ]

=>

[ Result : boolean ]
```

#### *2.1.3.5.12 Hash*

```
HashObjectUID.HashCalc [
        Input : typeOr { Data = bytes, BufferIn = cell_block } ]

=>

[ Result : bytes ]
```

#### *2.1.3.5.13 HashFinalize*

```
HashObjectUID.HMACFinalize [ ]

=>

[ Result : bytes ]
```

#### *2.1.3.5.14 HashCinit*

```
HashObjectUID.HMACInit [ ]

=>

[ Result : boolean ]
```

#### *2.1.3.5.15 HMAC*

```
HashObjectUID.HMACCalc [
        Input : typeOr { Data = bytes, Buffer = cell_block } ]

=>

[ Result : bytes ]
```

#### *2.1.3.5.16 HMACFinalize*

```
HashObjectUID.HMACFinalize [
        BufferOut = cell_block
        ]

=>

[ Result : bytes ]
```

#### *2.1.3.5.17 XOR*

```
SPUID.XOR[
        PatternInput : uidref {ByteTable},
        DeletePattern : boolean,
        Input : typeOr { Data = bytes, BufferIn = cell_block },
        BufferOut = cell_block
```

```
        ]
    =>
    [ Result : bytes ]
```

### 2.1.3.6 Locking Template

#### *2.1.3.6.1 GetPackage*

```
    CredentialObjectUID.GetPackage [
        Purpose : package_purpose,
        WrappingKey : uidref { AuthorityObjectUID },
        HashType : hash_protocol,
        Date = date,
        Log = bytes
    =>
    [ Result : package ]
```

#### *2.1.3.6.2 SetPackage*

```
    CredentialObjectUID.SetPackage [
        Value : package,
        WrappingKey : uidref { AuthorityObjectUID },
        HashType : hash_protocol ]
    =>
    [ Result : boolean ]
```

### 2.1.3.7 Log Template

#### *2.1.3.7.1 AddLog*

```
    LogTableUID.AddLog[
        LogEntryName : name,
        Data : bytes ]
    =>
    [ Result : boolean ]
```

#### *2.1.3.7.2 CreateLog*

```
    LogListUID.CreateLog[
        NewLogTableName : name,
        HighSecurity : boolean,
        MinSize : uinteger,
        MaxSize = uinteger,
        Hintsize = uinteger,
        CommonName = name ]
    =>
    [ LogListUID : uid, LogTableUID : uid, Rows : uinteger ]
```

#### *2.1.3.7.3 ClearLog*

```
    LogTableUID.ClearLog [ ]
    =>
```

```
[ Result : boolean ]
```

#### *2.1.3.7.4 FlushLog*

```
LogTableUID.FlushLog [ ]
=>
[ Result : boolean ]
```

## 2.1.4 Column Types in Messaging

Certain column types used in messaging as method parameters (particularly in the Set method) utilize the interface grouping mechanisms (Named and List values) to provide clarity regarding the scope of the transmitted values.

- Simple types – values of this type require no special handling in the messaging stream.
- Enumeration types – values of this type require no special handling in the messaging stream.
- Alternative types – values of this type require no special handling in the messaging stream. See 2.1.7 for more information on Alternative types.
- List type – the "List" column type is handled in the same way a parameter list is handled, by using the interface List value grouping tokens (F0 and F1 tokens, which represent "[" and "]" respectively) to enclose the values in the list.
- Restricted Reference types – values of this type require no special handling in the messaging stream.
- General Reference types – values of this type require no special handling in the messaging stream.
- Name value types – values of this type are handled in the same way a Named value in a parameter list is handled, by using the Named value grouping tokens (SN and EN tokens, which represent "StartName" and "EndName" respectively) to enclose the name-value pair.

  Note that Name value types are made up of two components – a name component (bytes), and a uid to a row in the Type table. So, the Format column for a Name value type holds a format identifier, a name, and a uid to a Type table row.
- Struct value types – Structs allow the creation of composite types by combining Name value types. Values of the struct type are made up of either Named value types. These optional types do not need to be included when sending values for a struct.

  The struct itself is delimited using the List value grouping tokens (F0 and F1 tokens, which represent "[" and "]" respectively) to enclose the values in the struct. The name values that make up the values stored in the struct are each grouped using the interface Named value-grouping tokens (SN and EN tokens, which represent "StartName" and "EndName" respectively) to enclose each name-value pair.
- Set value types – the "Set" column type is handled in the same way that the List type is handled, by using the interface List value grouping tokens (F0 and F1 tokens, which represent "[" and "]" respectively) to enclose the values in the Set.

## 2.1.5 Type Table

The Type table holds only those types required to identify Column types/sizes. Abstract types and Interface types are not represented in the Type table, though there may be types represented in the Type table that appear to be similarly constructed to Abstract and Interface types, or that have similar names. These type analogs are not to be confused with the actual Interface types or the pseudo-code Abstract types.

The contents of the Type table are presented below. The Default column defines the default value for the associated type. Rows with no Default column value in the descriptive table are types that shall have a value specified whenever a column of that type is used in a CreateRow (or other similar) method invocation, or that method invocation shall fail. Other rows, those with values, shall have a uidref in the Type table to a byte table that stores the default value for that type (without the “”).

The Description column in the table below is informative only, and is not intended to be part of the Type table implementation.

In the interest of efficiency, due to the removal of many types from this table, it may be necessary to re-assign uids to the default rows described below to utilize contiguous blocks of uids.

Note: * in the table below indicates SSC-dependent or implantation-dependent values.

**Table 01 Default Type Table Values**

| ID | Name | Format | Size | Default | Description |
|---|---|---|---|---|---|
| 00 00 00 05 00 00 00 01 | NULL | 0 | 0 | | Base installed type, used to represent a null value. The null value for a particular column is dependent on that column's type. In order to define a legal Null value for a particular type, it is necessary to construct an alternative type where Null is one of the options. |
| 00 00 00 05 00 00 00 02 | bytes | 0 | 0 | | Base installed type, used to represent a value made up of a fixed-size sequence of bytes. |
| 00 00 00 05 00 00 00 03 | max_bytes | 0 | 0 | | Base installed type, used to represent a bytes value that is equal to or less than the size specified for the type instance. |
| 00 00 00 05 00 00 00 04 | integer | 0 | 0 | | Base installed type, used to represent a signed integer. |
| 00 00 00 05 00 00 00 05 | uinteger | 0 | 0 | | Base installed type, used to represent an unsigned integer. |
| 00 00 00 05 00 00 02 01 | bytes_12 | 1 0000000500000002 12 | | | |
| 00 00 00 05 00 00 02 02 | bytes_16 | 1 0000000500000002 16 | | | |
| 00 00 00 05 00 00 02 03 | bytes_20_def_00 | 1 0000000500000002 20 | | "00s" | |
| 00 00 00 05 00 00 02 04 | bytes_32_def_00 | 1 0000000500000002 32 | | "00s" | |
| 00 00 00 05 00 00 02 05 | bytes_32 | 1 0000000500000002 32 | | | This bytes type is used for, among other things, the Key column of the C_HMAC_256 table. |
| 00 00 00 05 00 00 02 06 | version_bytes_4 | 1 0000000500000002 | | "00 00 00 01" | |

| ID | Name | Format | Size | Default | Description |
|---|---|---|---|---|---|
| | | 4 | | | |
| 00 00 00 05 00 00 02 07 | bytes_48_def_00 | 1 0000000500000002 48 | | "00s" | |
| 00 00 00 05 00 00 02 08 | bytes_64_def_00 | 1 0000000500000002 64 | | "00s" | |
| 00 00 00 05 00 00 02 09 | uid | 1 0000000500000002 8 | | | Used for UIDs |
| 00 00 00 05 00 00 02 0B | name | 1 0000000500000003 32 | | | Name that generically describes bytes{max=32}, which is used for name columns and method parameters. This type is also used in the Name_Value_Type format. |
| 00 00 00 05 00 00 02 0C | password | 1 0000000500000003 32 | | | Max {bytes = 32}, used for PINs |
| 00 00 00 05 00 00 02 0D | max_bytes_32 | 1 0000000500000003 32 | | | |
| 00 00 00 05 00 00 02 0E | max_bytes_64 | 1 0000000500000003 64 | | | Generic Max Bytes type, used for logging. |
| 00 00 00 05 00 00 02 0F | int_1_def_0 | 1 0000000500000004 1 | | "0" | integer_1 with default of 0 |
| 00 00 00 05 00 00 02 10 | integer_1 | 1 0000000500000004 1 | | | |
| 00 00 00 05 00 00 02 11 | uinteger_1 | 1 0000000500000005 1 | | | |
| 00 00 00 05 00 00 02 12 | uinteger_128 | 1 0000000500000005 128 | | | |
| 00 00 00 05 00 00 02 13 | uinteger_16 | 1 0000000500000005 16 | | | |
| 00 00 00 05 00 00 02 14 | feedback_size | 1 0000000500000005 2 | | | Feedback sizes for AES used in CFB or OFB mode. If AES Mode is CFB, this shall be between 1 and the block length. If AES Mode is OFB, this shall be the block size. |
| 00 00 00 05 00 00 02 15 | uinteger_2 | 1 0000000500000005 2 | | | |

| ID | Name | Format | Size | Default | Description |
|---|---|---|---|---|---|
| 00 00 00 05 00 00 02 16 | uinteger_20 | 1 0000000500000005 20 | | | |
| 00 00 00 05 00 00 02 17 | uinteger_21 | 1 0000000500000005 21 | | | |
| 00 00 00 05 00 00 02 18 | uinteger_24 | 1 0000000500000005 24 | | | |
| 00 00 00 05 00 00 02 19 | uinteger_256 | 1 0000000500000005 256 | | | |
| 00 00 00 05 00 00 02 1A | uinteger_28 | 1 0000000500000005 28 | | | |
| 00 00 00 05 00 00 02 1B | uinteger_30 | 1 0000000500000005 30 | | | |
| 00 00 00 05 00 00 02 1D | uinteger_32 | 1 0000000500000005 32 | | | |
| 00 00 00 05 00 00 02 1F | uinteger_36 | 1 0000000500000005 36 | | | |
| 00 00 00 05 00 00 02 20 | uinteger_4 | 1 0000000500000005 4 | | | |
| 00 00 00 05 00 00 02 21 | uint_4_def_0 | 1 0000000500000005 4 | | "0" | Uinteger_4 with default of 0 |
| 00 00 00 05 00 00 02 23 | uinteger_48 | 1 0000000500000005 48 | | | |
| 00 00 00 05 00 00 02 24 | uinteger_64 | 1 0000000500000005 64 | | | |
| 00 00 00 05 00 00 02 25 | uinteger_8 | 1 0000000500000005 8 | | | |
| 00 00 00 05 00 00 02 26 | common_name | 1 0000000500000003 32 | | "Host_Application" | This type is used for the CommonName column. Many tables have values defined for the CommonName column for rows created at issuance. This type defines the default value of rows for user-defined objects. |
| 00 00 00 05 00 00 02 27 | uinteger_66 | 1 0000000500000005 66 | | | |

| ID | Name | Format | Size | Default | Description |
|---|---|---|---|---|---|
| 00 00 00 05 00 00 02 36 | bytes_20 | 1 0000000500000002 20 | | | This bytes type is used for the Key column of the C_HMAC_160 table. |
| 00 00 00 05 00 00 02 37 | bytes_48 | 1 0000000500000002 48 | | | This bytes type is used for the Key column of the C_HMAC_384 table. |
| 00 00 00 05 00 00 02 38 | bytes_64 | 1 0000000500000002 64 | | | This bytes type is used for the Key column of the C_HMAC_512 table. |
| 00 00 00 05 00 00 04 01 | boolean | 2 0 1 | | | Derived type, used to represent True (1) or False (0). |
| 00 00 00 05 00 00 04 02 | boolean_def_false | 2 0 1 | | "0" | |
| 00 00 00 05 00 00 04 03 | boolean_def_true | 2 0 1 | | "1" | |
| 00 00 00 05 00 00 04 04 | messaging_type | 2 0 128 | | | This enumeration describes the options for selecting secure messaging. The options for this value are defined in the TCG Storage Architecture Core Specification 27-128 are reserved values. |
| 00 00 00 05 00 00 04 05 | life_cycle_state | 2 0 15 | | | Used to represent the current life cycle state. The valid values are: 0 = issued, 1 = issued-disabled, 2 = issued-frozen, 3 = issued-disabled-frozen, 4 = manufacturing, 5 = manufacturing-disabled, 6 = manufacturing-frozen, 7 = manufacturing-disabled-frozen, 8 = failed, 9-15 = reserved |
| 00 00 00 05 00 00 04 06 | padding_type | 2 0 15 | | | Defines the type of padding used with RSA encryption. '0' identifies the value as None or Null, '1' identifies the padding as that described in PKCS #1 v 1.5, and '2' identifies the padding as that described in PKCS #1 v 2.1. Values 3-15 are reserved for future use. |
| 00 00 00 05 00 00 04 08 | auth_method | 2 0 23 | | | This describes the enumeration used to represent authentications methods that may be used to authenticate authorities. The valid entries are: 0 = None 1 = Password, 2 = Exhange, 3 = Sign, 4 = SymK, 5 = HMAC, 6 = TPerSign, 7 = TPerExchange, 8-23 = reserved for future use |

| ID | Name | Format | Size | Default | Description |
|---|---|---|---|---|---|
| 00 00 00 05 00 00 04 09 | log_kind | 2 0 23 | | | Used to represent the predefined log messages used in the default Log table. The valid values are: 0 = available, 1 = methodFail,2 = methodSuccess, 3 = authenticateFail,4 = authenticateSuccess,5 = transactOpen,6 = transactCommit, 7 = transactAbort,8 = sessionEnd, 9 = user, 10 = system, 11-23 = reserved |
| 00 00 00 05 00 00 04 0A | symmetric_mode | 2 0 23 | | | Defines the mode to be used with this AES credential. The valid values are: 0 = ECB, 1 = CBC, 2 = CFB, 3 = OFB, 4 = GCM, 5 = CTR, 6 = CCM, 23 = MediaEncryption, 7-22 reserved for future use. |
| 00 00 00 05 00 00 04 0B | clock_kind | 2 0 3 | | | Defines the type of clock currently active. The valid values are: 0 = Timer, 1 = Low, 2 = High, 3 = LowAndHigh |
| 00 00 00 05 00 00 04 0C | log_select | 2 0 3 | | | Identifies the scope of the logging for an access control association or authority. The valid values are: 0 = None,1 = LogSuccess,2 = LogFail,3 = LogAlways |
| 00 00 00 05 00 00 04 0D | hash_protocol | 2 0 15 | | | Selects which hash algorithm should be used to create a digital signature. Options are: 0 = none, 1 = SHA 1, 2 = SHA 256, 3 = SHA 384, 4 = SHA 512, 5-15 = reserved |
| 00 00 00 05 00 00 04 0E | boolean_ACE | 2 0 7 | | | Used to identify "And" and "Or", where "And" is 0, "Or" is 1, and “Not” is 2, and 3-7 are reserved for future use - used to construct ACE Expression |
| 00 00 00 05 00 00 04 0F | adv_key_mode | 2 0 7 | | | This enumeration defines when the NextKey is moved to the ActiveKey. 0 = wait for ADVKey_Req, 1 = auto-advance keys |
| 00 00 00 05 00 00 04 10 | keys_avail_conds | 2 0 7 | | | This enumeration describes the conditions required to assert KeysAvailable in the Locking Template. 0 = None, 1 = Authentication of the authority with Set access to read/write locked columns for the LBA Range |

| ID | Name | Format | Size | Default | Description |
|---|---|---|---|---|---|
| 00 00 00 05 00 00 04 11 | last_reenc_stat | 2 0 7 | | | This enumeration identifies the last attempted re-encryption step. 0 = success, 1 = Read error, 2 = Write Error, 3 = Verify Error |
| 00 00 00 05 00 00 04 12 | verify_mode | 2 0 7 | | | This enumeration defines the verification operation to perform after a sector has been written with a new encryption key. 0 = no verify, 1 = verify enabled, 2-7 = reserved |
| 00 00 00 05 00 00 04 13 | reencrypt_request | 2 1 16 | | | This enumeration identifies a host re-encryption request value. |
| 00 00 00 05 00 00 04 14 | reencrypt_state | 2 1 16 | | | This enumeration identifies the present Re-encryption state for an LBA range. 1 = Idle, 2 = Pending, 3 = Active, 4= Completed, 5 = Paused, 6-16 = Reserved |
| 00 00 00 05 00 00 04 15 | table_kind | 2 1 3 | | | Defines the kind of table. The valid values are: 1 = Object, 2 = Array, 3 = Byte |
| 00 00 00 05 00 00 06 01 | ACE_expression | 3 2<br>1<br>0000000500000C04<br>2<br>000000050000040E | | | This is an alternative type where the options are either a uidref to an ACE object or one of the boolean_ACE options |
| 00 00 00 05 00 00 06 02 | row_selection | 3 2<br>1<br>0000000500000F01<br>2<br>0000000500001001 | | | This type is used to provide a selection between a uidref to an object table row or a ref to an array table row |
| 00 00 00 05 00 00 06 04 | uint_ref | 3 2<br>1<br>0000000500000211<br>2<br>0000000500000C02 | | | Alternative type with selections for a uinteger_1 or a uidref to an object in the Type table |
| 00 00 00 05 00 00 06 06 | table_object_ref | 3 2<br>1<br>0000000500001001<br>2<br>0000000500001201 | | | This type defines a reference to the uid of a table or the uid of some object. |
| 00 00 00 05 00 00 08 01 | AC_element | 4 *<br>0000000500000601 | | | An AC_Element is a list of ACE_Expressions forming a postfix Authority expression. For example: [ 32 24 0 8273 1 7728 0 ] is the list representing the infix ACE Expression:((32 AND 24) OR 8273) AND 7728 |
| 00 00 00 05 00 00 08 02 | ACL | 4 *<br>0000000500000801 | | | An ACL is represented as a list of uidrefs to ACE objects. The length of the list is SSC- |

| ID | Name | Format | Size | Default | Description |
|---|---|---|---|---|---|
| | | | | | dependant. |
| 00 00 00 05<br>00 00 08 03 | type_ref_list | 4 *<br>0000000500000C02 | | | A list of an SSC-dependent number of uidrefs to objects in the Type table |
| 00 00 00 05<br>00 00 08 04 | components_list | 4 *<br>0000000500001601 | | | A list of SSC-dependant number of struct_component types. |
| 00 00 00 05<br>00 00 08 06 | uint_ref_list | 4 2<br>0000000500000604 | | | List of the Alternative type that contains selections for a uinteger_1 or a uidref to an object in the Type table |
| 00 00 00 05<br>00 00 0A 01 | column_ref | 5<br>0000000400000000 | | | Reference to a row number that must exist in the Column table |
| 00 00 00 05<br>00 00 0C 01 | SPTemplates_ref | 6<br>0000000300000000 | | | |
| 00 00 00 05<br>00 00 0C 02 | Type_ref | 6<br>0000000500000000 | | | Reference to a uid that must exist in the Type table. |
| 00 00 00 05<br>00 00 0C 03 | MethodID_ref | 6<br>0000000500000000 | | | Reference to a uid that must exist in the MethodID table |
| 00 00 00 05<br>00 00 0C 04 | ACE_table_ref | 6<br>0000000800000000 | | | This is a Restricted_Reference_Type, which indicates that the uidref used in this type must be to a uid contained in the ACE table. |
| 00 00 00 05<br>00 00 0C 05 | Authority_ref | 6<br>0000000900000000 | | | Reference to a uid that must exist in the Authority table |
| 00 00 00 05<br>00 00 0C 06 | Certificates_ref | 6<br>0000000A00000000 | | | Reference to a uid that must exist in the Certificates table |
| 00 00 00 05<br>00 00 0C 08 | Template_ref | 6<br>0000020400000000 | | | Reference to a uid that must exist in the Admin SP's Template table. |
| 00 00 00 05<br>00 00 0F 01 | row_ref | 7 | | | |
| 00 00 00 05<br>00 00 0F 02 | log_row_ref | 7 | | | This is a reference type that shall be used specifically for rows in Log tables. When performing type checking, as part of that type checking the TPer shall validate that this is a ref to a row in a Log table. |
| 00 00 00 05<br>00 00 10 01 | row_uidref | 8 | | | |

| ID | Name | Format | Size | Default | Description |
|---|---|---|---|---|---|
| 00 00 00 05<br>00 00 10 02 | cred_object_uidref | 8 | | | This is a reference type that shall be used specifically for uidrefs to credential objects. When performing type checking, as part of that type checking the TPer shall validate that this uidref is to an object in a credential (C_*) table. |
| 00 00 00 05<br>00 00 12 01 | table_ref | 9 | | | This type is used to represent a uidref to a Table that is one of the set of all tables in the SP. |
| 00 00 00 05<br>00 00 12 02 | ref_def_00 | 9 | | "00 00 00 00 00 00 00 00" | This table ref is to a byte table that defines the default value for this column |
| 00 00 00 05<br>00 00 12 03 | byte_table_ref | 9 | | | This is a reference type that shall be used specifically for uidrefs to byte tables. When performing type checking, as part of that type checking the TPer shall validate that this uidref is to a table that is a byte table. |
| 00 00 00 05<br>00 00 14 01 | Year | 10<br>Year<br>0000000500000215 | | | Name-value pair that has a Name of "Year" and takes a uinteger_2 as the value. The value limitations associated with this type can be found in 2.1.2.1.4. |
| 00 00 00 05<br>00 00 14 02 | Month | 10<br>Month<br>0000000500000211 | | | Name-value pair that has a Name of "Month" and takes a uinteger_1 as the value. The value limitations associated with this type can be found in 2.1.2.1.4. |
| 00 00 00 05<br>00 00 14 03 | Day | 10<br>Day<br>0000000500000211 | | | Name-value pair that has a Name of "Day" and takes a uinteger_1 as the value. The value limitations associated with this type can be found in 2.1.2.1.4. |
| 00 00 00 05<br>00 00 14 04 | Hour | 10<br>Hour<br>0000000500000211 | | | Name-value pair that has a Name of "Hour" and takes a uinteger_1 as the value. The value limitations associated with this type can be found in 2.1.2.1.4. |
| 00 00 00 05<br>00 00 14 05 | Minute | 10<br>Minute<br>0000000500000211 | | | Name-value pair that has a Name of "Minte" and takes a uinteger_1 as the value. The value limitations associated with this type can be found in 2.1.2.1.4. |

| ID | Name | Format | Size | Default | Description |
|---|---|---|---|---|---|
| 00 00 00 05<br>00 00 14 06 | Seconds | 10 Seconds<br>0000000500000211 | | | Name-value pair that has a Name of "Seconds" and takes a uinteger_2 as the value. The value limitations associated with this type can be found in 2.1.2.1.4. |
| 00 00 00 05<br>00 00 14 07 | Fraction | 10 Fraction<br>0000000500000215 | | | Name-value pair that has a Name of "Fraction" and takes a uinteger_1 as the value. The value limitations associated with this type can be found in 2.1.2.1.4. |
| 00 00 00 05<br>00 00 14 08 | Format | 10<br>Format<br>0000000500000211 | | | Name-value pair that has a Name of "Format" and that takes a uinteger_1 value. |
| 00 00 00 05<br>00 00 14 09 | Size | 10<br>Size<br>0000000500000211 | | | Name-value pair that has a Name of "Size" and that takes a uinteger_1 value. |
| 00 00 00 05<br>00 00 14 0A | Name | 10<br>Name<br>000000050000020D | | | Name-value pair that has a Name of "Name" and that takes a max_bytes_32 value. |
| 00 00 00 05<br>00 00 14 0B | Components | 10<br>Components<br>0000000500000804 | | | Name-value pair that has a Name of "Components" and that has a components_list type as a value. |
| 00 00 00 05<br>00 00 14 0C | ID | 10<br>ID<br>000000050000020D | | | Name-value pair that has a Name of "ID" and that takes a max bytes 32 as the value. |
| 00 00 00 05<br>00 00 14 0D | Value | 10<br>Value<br>0000000500000C02 | | | Name-value pair that has a Name of "Value" and takes a uid that must exist in the Type table as the value. |
| 00 00 00 05<br>00 00 16 02 | lag | 11 2<br>0000000500001406<br>0000000500001407 | | | A struct made up of 2 uinteger_2 name-value types, used to define the lag when setting time. The 2 types represent seconds and fraction of seconds. The names required are "Seconds" for the first value and "Fraction" for the second. The "Fraction" value is a number of milliseconds. |
| 00 00 00 05<br>00 00 16 04 | date | 11 3<br>0000000500001401<br>0000000500001402<br>0000000500001403 | | | The date type represents the date portion of the time from the system clock. This is a set of name-value pairs, with the following names: "Year" (uinteger_2), "Month" (uinteger_1), and "Day" (uinteger_1) |

| ID | Name | Format | Size | Default | Description |
|---|---|---|---|---|---|
| 00 00 00 05<br>00 00 16 05 | clock_time | 11 3<br>0000000500001401<br>0000000500001402<br>0000000500001403<br>0000000500001404<br>0000000500001405<br>0000000500001406<br>0000000500001407 | | | Type made up of name-value pairs used to represent time. Any value not supplied is treated as 0. Time comes from the Clock SP. If the host has supplied a trusted time since powerup, that time is used; otherwise a monotonic counter is used. The Clock_time type can be used to represent times in either Generalized Time or UTC Time. Using this type to represent UTC Time requires 0's (zeroes) in fields where Generalized time requires a value but UTC Time does not (i.e. 2006 in UTC Time would be represented as 0006). The names for these name-value types are "Year", "Month", "Day", "Hour", "Minute", "Seconds", "Fraction" |
| 00 00 00 05<br>00 00 16 06 | type_def | 11 4<br>0000000500001408<br>0000000500001409<br>000000050000140A<br>000000050000140B | | | A struct made up of of the Format name-value type, the Size name-value type, the Name name-value type, and the Components name-value type. It is used to represent the Type table format column. The requirements for each of the name-value pairs in the type are based on the corresponding format (the value of the component Format type). |
| 00 00 00 05<br>00 00 16 01 | struct_components | 11 2<br>000000050000140C<br>000000050000140D | | | Struct composed of an ID type that takes a name value of Name=name and name value of Value=uid {Type} |
| 00 00 00 05<br>00 00 18 01 | reset_types | 12 0 31 | | | This Set type is used to identify TCG reset types that map to interface specific behaviors. The set values are: 0 = Power Cycle, 1 = Hardware, 2 = HotPlug, 3-15=reserved for TCG use, 16-31 reserved for vendor-specific reset behaviors. |
| 00 00 00 05<br>00 00 18 02 | gen_status | 12 0 63 | | | This set type is used to identify the general status of the re-encryption process. |

| ID | Name | Format | Size | Default | Description |
|---|---|---|---|---|---|
| 00 00 00 05 00 00 18 03 | enc_supported | 12 0 15 | | | This set describes the types of user data encryption supported by the TPer. 0 = None, 1 = Media Encryption, 2-15 are reserved. |

### 2.1.6 Type Checking

*Informative Content:*

Type checking when a method is received remains possible even without the use of type identifiers in the messaging stream.

It is reasonable to consider the parameter list of each method call as a struct with both required and optional member types. Since this is the case, whenever a particular method is received, the TPer may compare the values included as method parameters in the method's invocation to the types required for each of that method's parameters according to the method definition.

For methods that have less static parameter requirements (such as the Set and Get methods), it is necessary to consider the composition of the table upon which the method is operating.

Using the Set method as an example, the method parameters upon invocation of this method will include names of columns and the values to be assigned to each of those columns. Because the definition of a table is known and fixed, the TPer can treat each table as a struct (for the purposes of type checking), with components equal to the columns of that table.

With the knowledge of the columns that make up the table/object upon which the method is operating, as well as the type of each of those columns, the TPer is able to initially determine if the value sent is of the correct type for each column without having to perform close type checking on whether or not the value is valid for actual assignation to that object's column (i.e. the TPer can initially verify that a particular parameter is a uinteger without having to determine if its size is within bounds for the column).

### 2.1.7 Other Restrictions

In order to mitigate complexity required for type casting, and to limit messaging overhead to identify options in variable types, it is necessary for the TPer to enforce that, for alternative types, each component of the alternative type, when the type is created, shall have an arbitrary bytes identifier associated by the host that shall be unique for that component of that type.

For instance, when creating an alternative type that is made up of a uinteger_1 and a uinteger_2, the host includes in the data for the Format column for that type an identifier preceding the uid of component types.

When a value is to be set to a column that is an alternative type, the value includes both the identifier and the value. In the previous example, invoking the Set method to set a value in a column that is either a 1 followed by the uinteger_1 value or a 2 followed by the uinteger_2 value.

## 2.2 Examples

### 2.2.1 Get

This example demonstrates the invocation of the Get method on the TPerInfo table in order to retrieve the GUDID.

#### 2.2.1.1 Pseudo-Code Invocation

```
TPerInfo_TableUID.Get [ [ "startRow" = "Row" = 1, "endRow" = "Row" = 1,
"startColumn" = "GUDID", "endColumn" = "GUDID" ] ]
```

#### 2.2.1.2 Invocation Byte Representation

```
F8 A8 00 00 02 01 00 00 00 00 A8 00 00 00 06 00 00 00 0C F0 F0 F2 A8 73 74 61
72 74 52 6F 77 F2 A3 52 6F 77 01 F3 F3 F2 A6 65 6E 64 52 6F 77 F2 A3 52 6F 77
01 F3 F3 F2 AB 73 74 61 72 74 43 6F 6C 75 6D 6E A5 47 55 44 49 44 F3 F2 A9 65
6E 64 43 6F 6C 75 6D 6E A5 47 55 44 49 44 F3 F1 F1 F9 F0 00 00 00 F1
```

| Bytes | Purpose | Value | Notes |
|---|---|---|---|
| F8 | Call Token | | Begins method |
| A8 00 00 02 01 00 00 00 00 | Invoking UID | TPerInfo table UID | |
| A8 00 00 00 06 00 00 00 0C | Method UID | Get Method UID | |
| F0 | Start List Token | | Begins parameter list |
| F0 | Start List Token | | Begins cell block for Where parameter |
| F2 | Start Name Token | | name-value |
| A8 73 74 61 72 74 52 6F 77 | Name | "startRow" | |
| F2 | Start Name Token | | |
| A3 52 6F 77 | Name | "Row" | |
| 01 | Value | 1 | |
| F3 | End Name Token | | |
| F3 | End Name Token | | |
| F2 | Start Name Token | | name-value |
| A6 65 6E 64 52 6F 77 | name | "endRow" | |
| F2 | Start Name Token | | |
| A3 52 6F 77 | Name | "Row" | |
| 01 | Value | 1 | |
| F3 | End Name Token | | |
| F3 | End Name Token | | |
| F2 | Start Name Token | | name-value |
| AB 73 74 61 72 74 43 6F 6C 75 6D 6E | Name | "startColumn" | |
| A5 47 55 44 49 44 | Value | "GUDID" | |
| F3 | End Name Token | | |
| F2 | Start Name Token | | name-value |
| A9 65 6E 64 43 6F 6C 75 6D 6E | Name | "endColumn" | |
| A5 47 55 44 49 44 | Value | "GUDID" | |
| F3 | End Name Token | | |
| F1 | End List Token | | Ends cell block |
| F1 | End List Token | | Ends parameter list |
| F9 | End of Data Token | | Ends method |
| F0 00 00 00 F1 | Status List | | |

#### 2.2.1.3 Pseudo-Code Response

```
["RowValues" = [ [ "GUDID"="GUDID_VALUE" ] ] ]
```

#### 2.2.1.4 Response Byte Representation

```
F0 F2 A9 52 6F 77 56 61 6C 75 65 73 F0 F0 F2 A5 47 55 44 49 44 AB 47 55 44 49
44 5F 56 41 4C 55 45 F3 F1 F1 F3 F1
```

| Bytes | Purpose | Value | Notes |
|---|---|---|---|
| F0 | Start List token | | Start of results list |
| F2 | Start Name token | | |
| A9 52 6F 77 56 61 6C 75 65 73 | Name | "RowValues" | |
| F0 | Start List token | | |
| F0 | Start List token | | Start of first row of results |
| F2 | Start Name token | | name-value |
| A5 47 55 44 49 44 | | "GUDID" | |
| AB 47 55 44 49 44 5F 56 41 4C 55 45 | | "GUDID_VALUE" | |
| F3 | End Name token | | |
| F1 | End List token | | |
| F1 | End List token | | |
| F3 | End Name token | | |
| F1 | End List token | | End of results list |

## 2.2.2 CreateRow

This example demonstrates the invocation of the CreateRow method on the C_PIN table to create two rows within one CreateRow invocation.

### 2.2.2.1 Pseudo-Code Invocation

```
C_PIN_TableUID.CreateRow [ [ [ "Name" = "LockingOwner1", "CommonName" = "Pool",
"PIN" = "THIS_IS_A_32_BYTE_PASSWORD_VALUE", "TryLimit" = 0, "Tries" = 0,
"Persistence" = 0 ] [ "Name" = "LockingOwner2", "CommonName" = "Pool", "PIN" =
"THIS_IS_A_32_BYTE_PASSWORD_VALUE", "TryLimit" = 0, "Tries" = 0, "Persistence"
= 0 ] ] ]
```

### 2.2.2.2 Invocation Byte Representation

```
F8 A8 00 00 00 0B 00 00 00 00 A8 00 00 00 06 00 00 00 0A F0 F0 F0 F2 A4 4E 61
6D 65 AC 4C 6F 63 6B 69 6E 67 4F 77 6E 65 72 31 F3 F2 AA 43 6F 6D 6D 6F 6E 4E
61 6D 65 A4 50 6F 6F 6C F3 F2 A3 50 49 4E D0 20 54 48 49 53 5F 49 53 5F 41 5F
33 32 5F 42 59 54 45 5F 50 41 53 53 57 4F 52 44 5F 56 41 4C 55 45 F3 F2 A8 54
72 79 4C 69 6D 69 74 00 F3 F2 A5 54 72 69 65 73 00 F3 F2 AB 50 65 72 73 69 73
74 65 6E 63 65 00 F3 F1 F0 F2 A4 4E 61 6D 65 AC 4C 6F 63 6B 69 6E 67 4F 77 6E
65 72 32 F3 F2 AA 43 6F 6D 6D 6F 6E 4E 61 6D 65 A4 50 6F 6F 6C F3 F2 A3 50 49
4E D0 20 54 48 49 53 5F 49 53 5F 41 5F 33 32 5F 42 59 54 45 5F 50 41 53 53 57
4F 52 44 5F 56 41 4C 55 45 F3 F2 A8 54 72 79 4C 69 6D 69 74 00 F3 F2 A5 54 72
69 65 73 00 F3 F2 AB 50 65 72 73 69 73 74 65 6E 63 65 00 F3 F1 F1 F1 F9 F0 00
00 00 F1
```

| Bytes | Purpose | Value | Notes |
|---|---|---|---|
| F8 | Call Token | | Begins method |
| A8 00 00 00 0B 00 00 00 00 | Invoking UID | C_PIN Table UID | |
| A8 00 00 00 06 00 00 00 0A | Method UID | CreateRow Method UID | |

| | | | |
|---|---|---|---|
| F0 | Start List Token | | Begins parameter list |
| F0 | Start List Token | | |
| F0 | Start List Token | | |
| F2 | Start Name Token | | name-value |
| A4 4E 61 6D 65 | | "Name" | |
| AC 4C 6F 63 6B 69 6E 67 4F 77 6E 65 72 31 | | "LockingOwner1" | |
| F3 | End Name Token | | |
| F2 | Start Name Token | | name-value |
| AA 43 6F 6D 6D 6F 6E 4E 61 6D 65 | | "CommonName" | |
| A4 50 6F 6F 6C | | "Pool" | |
| F3 | End Name Token | | |
| F2 | Start Name Token | | name-value |
| A3 50 49 4E | | "PIN" | |
| D0 20 54 48 49 53 5F 49 53 5F 41 5F 33 32 5F 42 59 54 45 5F 50 41 53 53 57 4F 52 44 5F 56 41 4C 55 45 | | "THIS_IS_A_32_BYTE_PASSWORD_VALUE" | |
| F3 | End Name Token | | |
| F2 | Start Name Token | | name-value |
| A8 54 72 79 4C 69 6D 69 74 | | "TryLimit" | |
| 00 | | 0 | |
| F3 | End Name Token | | |
| F2 | Start Name Token | | name-value |
| A5 54 72 69 65 73 | | "Tries" | |
| 00 | | 0 | |
| F3 | End Name Token | | |
| F2 | Start Name Token | | name-value |
| AB 50 65 72 73 69 73 74 65 6E 63 65 | | "Persistence" | |
| 00 | | 0 | |

| | | | |
|---|---|---|---|
| F3 | End Name Token | | |
| F1 | End List Token | | |
| F0 | Start List Token | | |
| F2 | Start Name Token | | name-value |
| A4 4E 61 6D 65 | | "Name" | |
| AC 4C 6F 63 6B 69 6E 67 4F 77 6E 65 72 32 | | "LockingOwner2" | |
| F3 | End Name Token | | |
| F2 | Start Name Token | | name-value |
| AA 43 6F 6D 6D 6F 6E 4E 61 6D 65 | | "CommonName" | |
| A4 50 6F 6F 6C | | "Pool" | |
| F3 | End Name Token | | |
| F2 | Start Name Token | | name-value |
| A3 50 49 4E | | "PIN" | |
| D0 20 54 48 49 53 5F 49 53 5F 41 5F 33 32 5F 42 59 54 45 5F 50 41 53 53 57 4F 52 44 5F 56 41 4C 55 45 | | "THIS_IS_A_32_BYTE_PASSWORD_VALUE" | |
| F3 | End Name Token | | |
| F2 | Start Name Token | | name-value |
| A8 54 72 79 4C 69 6D 69 74 | | "TryLimit" | |
| 00 | | 0 | |
| F3 | End Name Token | | |
| F2 | Start Name Token | | name-value |
| A5 54 72 69 65 73 | | "Tries" | |
| 00 | | 0 | |
| F3 | End Name Token | | |
| F2 | Start Name Token | | name-value |
| AB 50 65 72 73 69 73 74 65 6E 63 65 | | "Persistence" | |
| 00 | | 0 | |
| F3 | End Name Token | | |
| F1 | End List Token | | |

| | | | |
|---|---|---|---|
| F1 | End List Token | | |
| F1 | End List Token | | Ends parameter list |
| F9 | End of Data Token | | Ends method |
| F0 00 00 00 F1 | Status List | | |

#### 2.2.2.3 Pseudo-Code Response

```
[ "ObjectTable" = [ 0000000B00000002 0000000B00000003 ] ]
```

#### 2.2.2.4 Response Byte Representation

```
F0 F2 AA 4F 62 6A 65 63 74 54 61 62 6C 65 F0 A8 00 00 00 0B 00 00 00 02 00 00
00 0B 00 00 00 03 F1 F3 F1 F9 F0 00 00 00 F1
```

| Bytes | Purpose | Value | Notes |
|---|---|---|---|
| F0 | Start List Token | | Begins results list |
| F2 | Start Name Token | | |
| AA 4F 62 6A 65 63 74 54 61 62 6C 65 | Name | "ObjectTable" | |
| F0 | Start List Token | | |
| A8 00 00 00 0B 00 00 00 02 | New Row UID | | |
| A8 00 00 00 0B 00 00 00 03 | New Row UID | | |
| F1 | End List Token | | |
| F3 | End Name Token | | |
| F1 | End List Token | | Ends results list |
| F9 | End of Data Token | | Ends method response |
| F0 00 00 00 F1 | Status List | | |

## 2.2.3 CreateTable

#### 2.2.3.1 Pseudo-Code Invocation

```
SPUID.CreateTable [ "SpecialStore", 1, 0000000000000000, [ "IsColumn" = [
"GUDID" = 0000000500000201, "PasswordUID" = 0000000500000209, "Name" =
000000050000020B, "CommonName" = 000000050000020B, "Password =
"000000050000020C ] ] , 100 ]
```

#### 2.2.3.2 Invocation Byte Representation

```
F8 A8 00 00 00 00 00 00 00 01 A8 00 00 00 06 00 00 00 08 F0 AC 53 70 65 63 69
61 6C 53 74 6F 72 65 01 A8 00 00 00 00 00 00 00 00 F0 F2 A8 49 73 43 6F 6C 75
6D 6E F0 F2 A5 47 55 44 49 44 A8 00 00 00 05 00 00 02 01 F3 F2 AB 50 61 73 73
77 6F 72 64 55 49 44 A8 00 00 00 05 00 00 02 09 F3 F2 A4 4E 61 6D 65 A8 00 00
00 05 00 00 02 0B F3 F2 AA 43 6F 6D 6D 6F 6E 4E 61 6D 65 A8 00 00 00 05 00 00
02 0B F3 F2 A8 50 61 73 73 77 6F 72 64 A8 00 00 00 05 00 00 02 0C F3 F1 F1 81
64 F1 F9 F0 00 00 00 F1
```

| Bytes | Purpose | Value | Notes |
|---|---|---|---|

| | | | |
|---|---|---|---|
| F8 | Call Token | | Begins method |
| A8 00 00 00 00 00 00 01 | Invoking UID | ThisSP Reserved UID | |
| A8 00 00 00 06 00 00 08 | Method UID | CreateTable Method UID | |
| F0 | Start List Token | | Begins parameter list |
| AC 53 70 65 63 69 61 6C 53 74 6F 72 65 | Required Parameter: Name | "SpecialStore" | |
| 01 | Required Parameter: Kind | Object table | |
| A8 00 00 00 00 00 00 00 00 | Required Parameter: GetSetACL | "0000000000000000" | |
| F0 | Start List Token | | |
| F2 | Start Name Token | | |
| A8 49 73 43 6F 6C 75 6D 6E | name | "IsColumn" | |
| F0 | Start List Token | | Required Parameter: Columns |
| F2 | Start Name Token | | name-value |
| A5 47 55 44 49 44 | | "GUDID" | |
| A8 00 00 00 05 00 00 02 01 | | bytes_12 | |
| F3 | End Name Token | | |
| F2 | Start Name Token | | name-value |
| AB 50 61 73 73 77 6F 72 64 55 49 44 | | "PasswordUID" | |
| A8 00 00 00 05 00 00 02 09 | | uid | |
| F3 | End Name Token | | |
| F2 | Start Name Token | | name-value |
| A4 4E 61 6D 65 | | "Name" | |
| A8 00 00 00 05 00 00 02 0B | | name | |
| F3 | End Name Token | | |
| F2 | Start Name Token | | name-value |
| AA 43 6F 6D 6D 6F 6E 4E 61 6D 65 | | "CommonName" | |
| A8 00 00 00 05 00 00 02 0B | | name | |
| F3 | End Name Token | | |
| F2 | Start Name Token | | name-value |
| A8 50 61 73 73 77 6F 72 64 | | "Password" | |
| A8 00 00 00 05 00 00 02 0C | | password | |
| F3 | End Name Token | | |
| F1 | End List Token | | |
| F1 | End List Token | | |

| | | | |
|---|---|---|---|
| 81 64 | Required Parameter: MinSize | 100 | |
| F1 | End List Token | | End of parameters |
| F9 | End of Data Token | | End of method |
| F0 00 00 00 F1 | Status List | | |

#### 2.2.3.3 Pseudo-Code Response

```
[ 0000090100000000, 100 ]
```

#### 2.2.3.4 Response Byte Representation

```
A8 00 00 09 01 00 00 09 01 81 64 F1 F9 F0 00 00 00 F1
```

| Bytes | Purpose | Value | Notes |
|---|---|---|---|
| F0 | Start List Token | | Begins results list |
| A8 00 00 09 01 00 00 09 01 | New Table UID | | |
| 81 64 | Number of rows assigned | 100 | |
| F1 | End List Token | | Ends results list |
| F9 | End of Data Token | | Ends method response |
| F0 00 00 00 F1 | Status List | | |

## 2.2.4 Set (Locking Object)

This example demonstrates the usage of the Set method. This particular example utilizes the Set method on a Locking object.

#### 2.2.4.1 Pseudo-Code Invocation

```
LockingTable_GlobalRangeUID.Set [ [ ], "RowValues" = [ [ "ReadLockEnabled" = 1,
"WriteLockEnabled" = 1, "ReadLocked" = 1, "WriteLocked" = 1, "LockOnReset" = [
0 ] ] ] ]
```

#### 2.2.4.2 Invocation Byte Representation

```
F8 A8 00 00 08 02 00 00 00 01 A8 00 00 00 06 00 00 00 0D F0 F0 F1 F2 A9 52 6F
77 56 61 6C 75 65 73 F0 F0 F2 AF 52 65 61 64 4C 6F 63 6B 45 6E 61 62 6C 65 64
01 F3 F2 D0 10 57 72 69 74 65 4C 6F 63 6B 45 6E 61 62 6C 65 64 01 F3 F2 AA 52
65 61 64 4C 6F 63 6B 65 64 01 F3 F2 AB 57 72 69 74 65 4C 6F 63 6B 65 64 01 F3
F2 AB 4C 6F 63 6B 4F 6E 52 65 73 65 74 F0 00 F1 F3 F1 F1 F1 F3 F1 F9 F0 00 00
00 F1
```

| Bytes | Purpose | Value | Notes |
|---|---|---|---|
| F8 | Call Token | | Begins method |
| A8 00 00 08 02 00 00 00 01 | Invoking UID | Locking Table Global Range Object UID | |
| A8 00 00 00 06 00 00 00 0D | Method UID | Set Methd UID | |
| F0 | Start List Token | | Begins parameter list |
| F0 | Start List Token | | |
| F1 | End List Token | | |

| | | | |
|---|---|---|---|
| F2 | Start Name Token | | |
| A9 52 6F 77 56 61 6C 75 65 73 | Name | "RowValues" | |
| F0 | Start List Token | | |
| F0 | Start List Token | | |
| F2 | Start Name Token | | name-value |
| AF 52 65 61 64 4C 6F 63 6B 45 6E 61 62 6C 65 64 | | "ReadLockEnabled" | |
| 01 | | True | |
| F3 | End Name Token | | |
| F2 | Start Name Token | | name-value |
| D0 10 57 72 69 74 65 4C 6F 63 6B 45 6E 61 62 6C 65 64 | | "WriteLockEnabled" | |
| 01 | | True | |
| F3 | End Name Token | | |
| F2 | Start Name Token | | name-value |
| AA 52 65 61 64 4C 6F 63 6B 65 64 | | "ReadLocked" | |
| 01 | | True | |
| F3 | End Name Token | | |
| F2 | Start Name Token | | name-value |
| AB 57 72 69 74 65 4C 6F 63 6B 65 64 | | "WriteLocked" | |
| 01 | | True | |
| F3 | End Name Token | | |
| F2 | Start Name Token | | name-value |
| AB 4C 6F 63 6B 4F 6E 52 65 73 65 74 | | "LockOnReset" | |
| F0 | Start List Token | | |
| 00 | | 0 | |
| F1 | End List Token | | |
| F3 | End Name Token | | |
| F1 | End List Token | | |
| F1 | End List Token | | |
| F1 | End List Token | | |

| | | | |
|---|---|---|---|
| F3 | End Name Token | | |
| F1 | End List Token | | Ends parameter list |
| F9 | End of Data Token | | Ends method |
| F0 00 00 00 F1 | Status List | | |

#### 2.2.3.3 Pseudo-Code Response

```
[ 0000090100000000, 100 ]
```

#### 2.2.4.3 Response Byte Representation

```
[ 1 ]
```

#### 2.2.4.4 Response Byte Representation

```
F0 01 F1 F9 F0 00 00 00 F1
```

| Bytes | Purpose | Value | Notes |
|---|---|---|---|
| F0 | Start List Token | | Begins results list |
| 01 | | True | |
| F1 | End List Token | | Ends results list |
| F9 | End of Data Token | | Ends method response |
| F0 00 00 00 F1 | Status List | | |

## 2.2.5 Set (ACE Object)

This example demonstrates the usage of the Set method. This particular example utilizes the Set method on an ACE object.

#### 2.2.5.1 Pseudo-Code Invocation

```
Locking_2_ACEObjectUID.Set [ [ ], "RowValues" = [ [ "BoolExpr" = [
0000000900000002, 0000000900FFFF01, 0 ] ] ] ]
```

#### 2.2.5.2 Invocation Byte Representation

```
F8 A8 00 00 00 08 00 00 08 03 A8 00 00 00 06 00 00 00 0D F0 F0 F1 F2 A9 52 6F
77 56 61 6C 75 65 73 F0 F0 F2 A8 42 6F 6F 6C 45 78 70 72 F0 A8 00 00 00 09 00
00 00 02 A8 00 00 00 09 00 FF FF 01 00 F1 F3 F1 F1 F1 F3 F1 F9 F0 00 00 00 F1
```

| Bytes | Purpose | Value | Notes |
|---|---|---|---|
| F8 | Call Token | | Begins method |
| A8 00 00 00 08 00 00 08 03 | Invoking UID | Locking_2 ACE Object UID | |
| A8 00 00 00 06 00 00 00 0D | Method UID | Set Methd UID | |
| F0 | Start List Token | | Begins parameter list |
| F0 | Start List Token | | |
| F1 | End List Token | | |
| F2 | Start Name Token | | |

| | | | |
|---|---|---|---|
| A9 52 6F 77 56 61 6C 75 65 73 | Name | "RowValues" | |
| F0 | Start List Token | | |
| F0 | Start List Token | | |
| F2 | Start Name Token | | name-value |
| A8 42 6F 6F 6C 45 78 70 72 | | "BoolExpr" | |
| F0 | Start List Token | | |
| A8 00 00 00 09 00 00 00 02 | | 0000000900000002 | Admins Class Authority UID |
| A8 00 00 00 09 00 FF FF 01 | | 0000000900FFFF01 | LockingOwner Authority UID |
| 00 | | And | |
| F1 | End List Token | | |
| F3 | End Name Token | | |
| F1 | End List Token | | |
| F1 | End List Token | | |
| F1 | End List Token | | |
| F3 | End Name Token | | |
| F1 | End List Token | | Ends parameter list |
| F9 | End of Data Token | | Ends method |
| F0 00 00 00 F1 | Status List | | |

#### 2.2.5.3 Response Byte Representation

[1]

#### 2.2.5.4 Response Byte Representation

```
F0 01 F1 F9 F0 00 00 00 F1
```

| Bytes | Purpose | Value | Notes |
|---|---|---|---|
| F0 | Start List Token | | Begins results list |
| 01 | | True | |
| F1 | End List Token | | Ends results list |
| F9 | End of Data Token | | Ends method response |
| F0 00 00 00 F1 | Status List | | |

*End Proposal*

### 10.2.3 Authenticate Method, Authority Parameter

*Start Proposal*

# Authenticate Method Parameter Requirement

Author: Jason Cox

Revision:

| | | |
|---|---|---|
| 0.1 | 07.31.07 | Start of draft (Jason Cox) |
| 0.2 | 08.03.07 | fixed title |

# 1 Goal

The goal of this proposal is:

- To suggest modification of the Authenticate method parameter requirements as described in Core Spec section 5.3.4.1.12.

# 2 Proposal

Section 5.3.4.1.12 of the Core Specification indicates that, when an authority that is being authenticated by the host through the use of the Authenticate method requires challenge-response, the host must invoke the Authenticate method twice. The first Authenticate shall be empty, and the result should be a challenge from the TPer.

This behavior is incorrect, and must be modified. The first Authenticate method in the challenge/response pair must be invoked with an Authority object UID as the Authority parameter, so that the TPer can determine the length of the challenge to be returned as the result of that method invocation.

The second Authenticate method invocation shall also be invoked with an Authority object UID as the Authority parameter, which shall be the same as that used in the first invocation. In addition, as specified, the second Authenticate method invocation shall contain the response to the TPer's challenge as its Challenge parameter.

*End Proposal*

### 10.2.4 Rename Method Table Name

*Start Proposal*

# Rename Method Tables

Author: Jason Cox

Revision:

| | | |
|---|---|---|
| 0.1 | 08.01.07 | Start of draft (Jason Cox) |
| 1.0 | 08.10.07 | Final draft, approved for inclusion in Core Spec (Jason Cox) |

# 1 Goal

The goal of this proposal is:

- To suggest a modification in table naming to make the Base Template Method table have a more descriptive name.

# 2 Proposal

In the TCG Store Core Architecture Specification, the Method table is a table that contains access control associations between methods and entities (objects/tables/SP). This document proposes to change the name of this table to "AccessControl".

The "Method" table becomes the "AccessControl" table.

*End Proposal*

10.2.5 Anybody Authority Authentication

*Start Proposal*

# Anybody Authority Authentication

Author: Jason Cox

Revision:

| | | |
|---|---|---|
| 0.1 | 08.07.2007 | Start of draft |

# 1 Goal

The goal of this proposal is:

- To clarify authentication methods and usages associated with the Anybody Authority.

# 2 Proposal

The utilization of the Anybody authority in certain situations requires clarification in the TCG Storage Architecture Core Specification.

## 2.1 Concepts

### 2.1.1 Session Startup

The Anybody authority has an Operation column value of None. Per the TCG Storage Architecture Core Specification, an authority with an Operation value of None may be parameterized during session startup as a signing authority (HostSigningAuthority in the StartSession method, or SPSigningAuthority – the ResponseSign column value of the Control Authority).

Also per the Core Spec, parameterization of an authority with an Operation value of None as an exchange authority (HostExchangeAuthority in the StartSession method, or SPExchangeAuthority – the ResponseExch column value of the Control Authority) shall result in method failure.

Thus, during session startup, it is not an error for the Anybody authority to be parameterized as the HostSigningAuthority in the StartSession method. If the Anybody authority is submitted in this manner, any value submitted in the HostChallenge parameter shall be ignored and disregarded.

It is also not an error for a referenced HostSigningAuthority to have the Anybody authority as its ResponseSign column value.

### 2.1.2 Authenticate Method

The Anybody authority can be submitted as a parameter of the Authenticate method, either with or without a value in the Authenticate method's Challenge parameter. Any value submitted in the Challenge parameter when the Authority parameter is the Anybody authority shall be ignored and disregarded. Assuming all other Authenticate method syntax is correct, the method shall return a Success status and a True result.

### 2.1.3 Authentication Limits

The Anybody authority counts against the maximum number of authenticated authorities permitted per session (as reported in the Properties method response MaxAuthentications field). Thus, if the maximum number of authorities that may be authenticated within a session is 8, the Anybody authority counts as one of these, and the host may authenticate up to 7 additional authorities during session startup or using the Authenticate method.

*End Proposal*

10.2.6 Symmetric Key ChallengeResponse

*Start Proposal*

# Symmetric Key Challenge/Response

Author: Jason Cox

Revision:

| | | |
|---|---|---|
| 0.1 | 08.09.2007 | Start of draft |

# 1 Goal

The goal of this proposal is:

- To clarify the operation of the SymK (symmetric key challenge/response) authentication type.

# 2 Proposal

This proposal identifies the requirements for the use of SymK (symmetric key challenge/response) authentication during session startup and upon invocation of the Authenticate method using an authority that requires such authentication.

This proposal describes the challenge size, response size, and encryption mode used with this authentication type.

## 2.1 Concepts

### 2.1.1 SymK Authorities

Authorities that require symmetric key challenge/response authenticate are identified by the value of their Operation column. Authorities that use SymK shall have an Operation column value of 4. Such an authority may be authenticated through the use of the Authenticate method, or via session startup. This authority may be used as the HostSigningAuthority in the StartSession method call, or it may be referenced from the Control Authority's ResponseSign column value as the SPSigningAuthority.

In addition to having an Operation column value of 4 ("SymK"), a SymK authority shall also have a Credential column value that is a valid uidref to a valid symmetric key credential object (for instance, a C_AES_128 or C_AES_256 object). That credential shall have a Mode column value of 0 ("ECB").

*End Proposal*

10.2.7 Global Locking Range Identification

*Start Proposal*

# Global Locking Range Identification

Author: Doug Philips, Jason Cox

Revision:

| | | |
|---|---|---|
| 0.1 | 09.19.2007 | Start of draft |

# 1 Goal

The goal of this proposal is:

- To clarify and simplify the identification of the Global Locking Range.

# 2 Proposal

This proposal clarifies the properties of the Global Locking Range by which it can be identified by the host.

# 3 Concepts

This item proposes removing text indicating that the Global Range is the "first row" in the table. The Locking table is an object table, and "row ordering is not defined for object tables.

Additionally, the implied restriction that only the Global Range may have RangeLength and RangeStart both = 0 should also be lifted.

Since the Core Spec assigns the UID of the Global Range, and that row is not deletable from the Locking table, the assigned UID is sufficient to identify the Global Range in that table and the other restrictions are superfluous.

This proposal lifts the restriction that the RangeLength and RangeStart columns of Locking objects other than the Global Range cannot be changed after the row has been created instead leaving that to the usual ACL control mechanism. Changes to the RangeLength and/or RangeStart columns are to be subjected to the same constraints and checks that are defined for those columns when rows of the locking table are created.

*End Proposal*

10.2.8 Fixed Location Optional Parameters

*Start Proposal*

# Fixed Location Optional Parameters

Author: Jason Cox

Revision:

| | | |
|---|---|---|
| 0.1 | 09.19.2007 | Start of draft |

# 1 Goal

The goal of this proposal is:

- To simplify method parameterization

# 2 Proposal

This proposal suggests requiring that Optional method parameters be submitted in a fixed order.

# 3 Concepts

Section 3.2.2.1 of the Core Specification 0.9 indicates:

> "Optional parameters are not required to be in order, and are not required to be included in a method invocation."

This proposal suggests that the above text in the Core Specification be replaced by the following:

"Optional parameters are not required to be included in a method invocation. If any optional parameters of a method are supplied to an invocation of that method, the supplied optional parameters shall be provided in the order specified in the Core Specification for that method. If any optional parameter is supplied out of order, the method invocation shall fail and return a non-success status code."

*End Proposal*

### 10.2.9 Authentication Within Transactions

*Start Proposal*

# Authentication Within Transactions

Author: Jason Cox

Revision:

| | | |
|---|---|---|
| 0.1 | 08.07.2007 | Start of draft |
| 0.2 | 10.04.2007 | Modified per SWG discussion |

# 1 Goal

The goal of this proposal is:

- To clarify the behavior of the Authenticate method when used within transactions.

# 2 Proposal

The utilization of the Authenticate method within transactions causes some specific behavior that differs from most other methods and should be defined in the TCG Storage Architecture Core Specification.

## 2.1 Concepts

### 2.1.1 Authenticate Method in-Transaction Invocations

Successful invocations of the Authenticate method occur outside of transactional control, such that even in the event that a transaction in which a successful Authenticate method occurs is aborted, the authority authenticated by that method invocation continues to be authenticated.

If a successful Authenticate method invocation is made at any time within a session (either inside or outside of a transaction), the authority is considered authenticated for the rest of the session and any subsequent method invocations that depend on that authentication will be authorized. This applies even to a successful Authenticate method invocation that occurs in a transaction that is subsequently aborted.

*End Proposal*

### 10.2.10 Zero-Length Locking Range Handling

*Start Proposal*

# Zero-length Locking Range Handling

Author: Doug Philips, Jason Cox

Revision:

| | | |
|---|---|---|
| 0.1 | 09.19.2007 | Start of draft |
| 0.2 | 10.02.2007 | Updated with Re-encryption interactions |
| 0.3 | 10.04.2007 | Added FAIL status code |

# 1 Goal

The goal of this proposal is:

- To permit the existence of zero length ranges in the Locking Table.
- To define the behavior of modification of Locking objects undergoing re-encryption.

# 2 Proposal

This proposal clarifies handling of Locking ranges with RangeLength column value of zero.

# 3 Concepts

## 3.1 Zero-Length Range Handling

This proposal depends on the Global Range Identification proposal's passage. Assuming that, this proposal enhances the definition of overlapping ranges such that they apply only to Locking Table rows that have a RangeLength > 0.

Locking objects whose RangeLength column == 0 do not have any LBAs under their control and thus do not overlap any other row, even if their RangeStart values match. Any Set method invocation that results in a Locking Table row's RangeLength column being non-zero, or that does not change a non-zero RangeLength column but does change a RangeStart column, is subject to the same overlapping range restrictions as already described in the Core Spec.

Locking objects that have a RangeLength column of zero interact with re-encryption requests as identified in the Core Spec and section 3.2 of this document.

## 3.2 Re-Encryption and Modification of Locking Objects

Attempts to modify the RangeStart and RangeLength columns of a Locking object that is undergoing re-encryption (the Locking object's ReEncryptState column value is not IDLE) shall fail and return a non-success status (FAIL) for the invoked method.

Attempts to delete a Locking object that has a ReEncryptState column value of ACTIVE shall fail and return a non-success status (FAIL) for the invoked method.

When the Global Locking Range is undergoing re-encryption (the Global Range's ReEncryptState column value is not IDLE):

- Attempts to modify the RangeStart and RangeLength columns of any Locking object shall fail and return a non-success status (FAIL) for the invoked method.
- Attempts to delete any Locking object shall fail and return a non-success status (FAIL) for the invoked method.
- Attempts to create a new Locking object shall fail and return a non-success status (FAIL) for the invoked method.

## 3.3 FAIL Status Code

In addition to the method status codes defined in the Core Spec, a new status code is suggested.

| Name | Value |
|---|---|
| FAIL | 0x3F |

This status is returned when a method fails in a manner for which none of the other failure statuses apply.

*End Proposal*

10.2.11 Next Method

*Start Proposal*

# Next Method Modification

Author: Scott Marks

Revision:

| | | |
|---|---|---|
| 1 | 11.08.2007 | Start of draft |

# 1 Goal

The goal of this proposal is:

- To clarify the semantics of the Next method.

# 2 Proposal

## 2.1 Concepts

The Core Specification defines the Next[] method to be:

```
TableUID.Next [
        Where : row_selection,
        Count: uinteger_4 ]
        =>
[ Result : next_result ]
```

The purpose of this method is to allow iteration over an object table despite the lack of ordering of the rows of such tables. This method is intentionally fragile with respect to modifications of the underlying table.

In addition, the semantics of "iteration", if that is taken to mean a sequence of Next method calls, is not completely described. What happens if one calls Next with the same parameters twice – when is it reasonable to expect the same result? What constitutes "modification of the underlying table"? If once calls Next without a Where parameter, without any intervening "modification", is it reasonable to expect the same result?

Fortunately, the "state" of the iteration can be considered to be completed specified in the Result returned by one step and subsequently fed back in as the Where in a next step. The very act of iteration implies, at least for the duration of that iteration, an "current" order to the rows, although that order is determined by the TPer, not in particular by the UIDs of the rows or the order in which rows were added or deleted in bringing the table to its current state.

This proposal suggests changing the focus of the next method from implementing iteration, with its possible implication of a hidden dynamic state, to discovery of a "current" order of rows of an object table. This order will still be arbitrarily determined by the TPer and subject to change when the underlying table is modified. In the proposed replacement explanatory text below, the first paragraph, addressing using the method on array tables, is unchanged.

### 2.1.1 Proposed replacement explanatory text following pseudocode

When successfully invoked on an array table, the Next method returns zero or more row number/uidref pairs currently in use in the table following the specified **Where** row, iterating sequentially (by RowNumber column value) through the table rows. If Where is not specified, the first row of the table is the first row number returned. If **Count** is not specified, it defaults to 1. If there are fewer than **Count** rows defined after the indicated starting row, only the defined row numbers are returned.

The Next Method may be used to discover an ordering of rows in an object table. Since the ordering of object tables is unspecified, the ordering that is discovered by successful invocation(s) of this method on an object table will be some undefined ordering, the "current" ordering.

When successfully invoked on an object table, the Next method returns a list of zero or more uidrefs "following" the specified **Where** row in the current ordering. If a value for the **Where** parameter is not specified in the method invocation, the first element, if any, of the list of uidrefs, will denote the "beginning" of the ordering, i.e. the row which has no predecessor in the current ordering.

The implementation is required to discover a consistent ordering of all rows of an object table only if the object table is not modified between calls to Next. Actions which cause modifications to the object table

that would result in a new ordering shall be specified in each SSC, and shall include at least method calls adding or deleting rows if those are permitted by the SSC.

*End Proposal*

10.2.12 Synchronous Communications

*Start Proposal*

# Synchronous Communications

Author: Jason Cox

Revision:

| | | |
|---|---|---|
| 1 | 08.28.2007 | Start of draft |
| 2 | 09.19.2007 | Removed some extraneous restrictions, mods per SJ FtF |
| 3 | 10.15.2007 | Removed more extraneous restrictions, added state diagram |
| 4 | 11.15.2007 | Modified OutstandingData and MinTransfer field values per discussion |
| 5 | 12.03.2007 | Modified to add method, protocol restriction; renumbered |

# 1 Goal

The goal of this proposal is:

- To describe the usage of the communications protocol stack described in the TCG Storage Architecture Core Specification to perform synchronous command exchanges.

# 2 Proposal

The Core Specification defines how to enable fully asynchronous messaging between host applications and SPs. The communications protocol can be adjusted also to enable synchronous communications to occur between host applications and SPs.

This proposal describes the operation of that protocol, including requirements on exchange of interface commands and associated limitations on packets, subpackets, and flow control.

## 2.1 Concepts

### 2.1.1 Introduction

*Begin Informative Text*

The communications protocol stack as described in the Core Specification enables a fully asynchronous exchange of data between host and TPer. Using the communications stack in this manner is a matter of arbitrarily interleaving IF-SEND commands with IF-RECV commands.

Asynchronous communications allows the host to transmit methods and data to the TPer without having to retrieve the results of those methods before sending additional methods; and enables the TPer to return method results, upon request, at arbitrary boundaries. Flow control provides a mechanism for buffer management to occur as data is successfully transmitted and received.

However, for some hosts and devices, these mechanisms are more complex and require more processing capability and code space than may be realistically available. For these situations, the communications protocol stack may be tailored to better meet the capabilities of the TPer.

For instance, fixed or semi-fixed sized commands simplifies message creation and parsing; and fixed buffer sizes along with restrictions on the relationship between IF-SEND and IF-RECV negates the need for the communications to require flow control for buffer management.

*End Informative Text*

## 2.1.2 Interface Commands

### 2.1.2.1 Restrictions

This section defines the restrictions imposed on the exchange of IF-SEND and IF-RECV commands.

1. Any number of non IF-SEND/IF-RECV commands may be interleaved with IF-SEND/IF-RECV commands.
2. The normal communications state of an Associated ComID shall be to await receipt of an IF-SEND command for that ComID.
    a. While awaiting receipt of an IF-SEND interface command, any received IF-SEND command with a valid ComID shall be accepted.
    b. Once the entire command payload has been received, the TPer shall return an interface status to the host.
    c. Any IF-RECV command received for the Associated ComID awaiting receipt of an IF-SEND command shall return to the host a ComPacket with a Length field value of zero, an OutstandingData field value of zero, and a MinTransfer field value of zero. This signals to the host that there is no pending response data to retrieve.
3. After an IF-SEND command has been received, a command completion without error has been returned, and the payload has been decoded without an error, the TPer shall not accept another IF-SEND command for that ComID until the host has retrieved the entire response via IF-RECV(s).
    a. Any subsequently received IF-SEND commands for the specified ComID shall be aborted at the interface level. The interface status for this action shall be specified in the SWG SIIF Interface Specification.
    b. If the TPer has not sufficiently processed the command payload and prepared a response, any IF-RECV command for that ComID shall receive a ComPacket with a Length field value of zero (no payload), an OutstandingData field value of 0x01, and a MinTransfer field value of zero.
    c. If the TPer has sufficiently processed the command payload and prepared a response, an IF-RECV command that requests a transfer length less than the amount of response data the TPer has prepared shall reply with a ComPacket with a Length field value of zero (no payload) and OutstandingData value of total bytes currently available, and MinTransfer field value of zero or the minimum request required to transfer a packet.
    d. In the case of TPers based on an SSC that permits multiple method invocations per IF-SEND command, the SSC may additionally require that each method response shall be retrieved separately (along with Control Tokens as determined by the TPer), via multiple IF-RECV commands. For these SSCs:
        i. If all responses have not been retrieved, and additional responses are available, the TPer shall respond to an IF-RECV command with OutstandingData value of total bytes currently available, and MinTransfer field value of zero or the minimum request required to transfer a packet.
        ii. If all responses have not been retrieved, and no additional responses are prepared but more are to come, the TPer shall respond to an IF-RECV command with OutstandingData field value of 0x01 and MinTransfer field value of Zero.

**Table 01 IF-RECV Field Values**

| IF-RECV | Length Field Value | OutstandingData Field Value | MinTransfer Field Value |
|---|---|---|---|

| IF-RECV | Length Field Value | OutstandingData Field Value | MinTransfer Field Value |
|---|---|---|---|
| Response(s) to come, no Response(s) available | Zero | 0x01* | Zero |
| Response ready, insufficient transfer length request | Zero | Total bytes currently available | Zero, or the minimum request required to transfer a packet |
| Response, additional Response(s) available | Data Length | Total bytes currently available | Zero, or the minimum request required to transfer a packet |
| Response, additional Response(s) to come, no Response(s) available | Data Length | 0x01* | Zero |
| Response, all Response(s) returned – no further data | Data Length | Zero | Zero |
| All Response(s) returned – no further data | Zero | Zero | Zero |

*Indicates a required change to the Core Spec – an OutstandingData field value of 0x01 denotes that the TPer is processing response(s). This provides insight to the host that there are responses still to come, but that are not ready yet. This is applicable to both the synchronous and asynchronous exchange of messages.

#### 2.1.2.2 Error Handling

This section defines the manner in which violations of the restrictions on Interface Commands shall be handled by the TPer.

1. If a restriction violation occurs such that the TPer is unable to resolve a valid Session ID in an IF-SEND command, or if the restriction violation occurs due to violations of packet requirements, the TPer shall ignore the entire payload and shall immediately transition to the state of awaiting an IF-SEND command.
2. If a restriction violation occurs such that the TPer is able to resolve the Session ID, the TPer shall close that session and shall prepare for transmission the CloseSession method for retrieval by the host.
3. The device shall abort at the interface level any IF-SEND command whose transfer length is greater than the reported MaxComPacketSize for the corresponding ComID. The interface status for this action shall be specified in the SWG SIIF Interface Specification.
4. For SSCs that require that entire method responses be retrieved, if data generated in response to any single method in an IF-SEND command (together with required communications overhead) does not fit entirely within the TPer's response buffer, the device shall not return any part of that method response and shall instead return an empty response list with a status code of RESPONSE_OVERFLOW in the response status list. Additionally, the TPer shall continue processing methods and control tokens that had been sent in that command payload (if any).

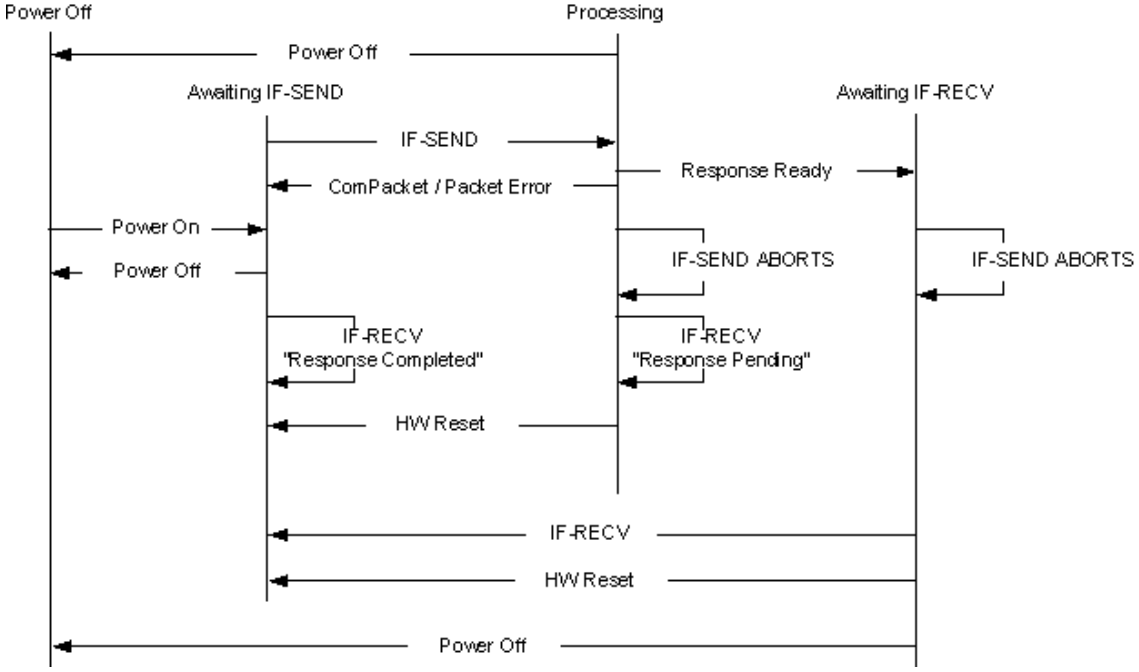
### 2.1.3 Other Restrictions

There are two other restrictions necessitated by this restriction on the exchange of interface commands:

1. Methods shall not span ComPackets. In the case where an incomplete method is submitted, if the TPer can identify the associated session, then that session shall be aborted and a CloseSession may be prepared for delivery on Session 0/Session Manager Layer.
2. The synchronous exchange of interface commands shall only apply to IF-SEND/IF-RECV commands exchanged on Protocol ID 1.

### 2.1.4 State Diagram

**Figure 1 State Diagram**




*End Proposal*

10.2.13 TypeOr Name Removal

*Start Proposal*

# TypeOr Name Removal

Author: Jason Cox

Revision:

| | | |
|---|---|---|
| 1 | 12.03.2007 | Start of draft |
| 2 | 1.02.2007 | Fixed Set method signature |

# 1 Goal

The goal of this proposal is:

- To remove TypeOr tag names in method parameters.

# 2 Proposal

This proposal suggests removing the TypeOr tag names in method parameters, and defines how TypeOr values are to be transmitted over the interface.

## 2.1 Scope

The scope of this proposal is limited to addressing TypeOr tag names in method parameters. This proposal does not address naming in Optional Parameter naming, and is not related to the Alternative column type.

## 2.2 Concepts

### 2.2.1 TypeOr Values

Based on the descriptions in the Core Specification, and updated in the Stream Type Removal Proposal, the TypeOr type used in method parameterization is defined as follows:

> "The components of an abstract typeOr alternative type used in method signature pseudo-code are always presented as Named value pairs. As such, each typeOr component will be represented on the interface as a Named value pair. Note that the typeOr itself may be an optional parameter or result, and as such this type could represent an instance of an embedded Named value pair (i.e. Name1 = Name2 = Value, where the value of Name2 is "Value" and the value of Name1 is "Name2 = Value")."

An example of a method signature that contains one of these types is the Next method:

```
TableUID.Next [
       Where = row_address,
       Count = uinteger ]
=>
[ Result : TypeOr { ArrayTable = list [ [ ref, uidref ] ... ],
ObjectTable = list [ uidref ... ] } ]
```

The result of this method contains a TypeOr type, and will either be represented as:

```
ArrayTable = [ [ ] ] or ObjectTable = [ ]
```

### 2.2.2 Proposed Functional Modifications

Review of the methods that contain TypeOr types and tags in the Core Specification indicate that an implementation can, based on the context of the method, properly identify and interpret which of the alternatives in that type is being presented as the method parameter or result.

Based on this review, this proposal suggests removing of the tagging in the TypeOr types on the in methods transmitted on the interface. Thus, the value transmitted for a TypeOr will not be a Named value type. It will only be the value, encoded as only the value (not as a Named value).

Additionally, for clarity and readability, this proposal suggests that the TypeOr names remain in the pseudo-code method definitions and abstract type definitions, but that those signatures change so that the TypeOr identifiers use ":" instead of "=". The identifier to the left of the ":" in the method signature would NOT be transmitted as part of the method invocation, and is used only for documentation and identification purposes.

#### 2.2.2.1 Documentation Example

For example, the Next method signature of Next is currently:

```
TableUID.Next [
       Where = row_address,
       Count = uinteger ]
=>
[ Result : TypeOr { ArrayTable = list [ [ ref, uidref ] ... ],
ObjectTable = list [ uidref ... ] } ]
```

When considered with the "abstract type" row_address factored into the method signature, the Next signature is actually:

```
TableUID.Next [
       Where = typeOr { RowAddress = ref, UIDAddress = uidref },
       Count = uinteger ]
=>
[ Result : TypeOr { ArrayTable = list [ [ ref, uidref ] ... ],
ObjectTable = list [ uidref ... ] } ]
```

This signature would change to:

```
TableUID.Next [
       Where = row_address,
       Count = uinteger ]
=>
[ Result : TypeOr { ArrayTable : list [ [ ref, uidref ] ... ], ObjectTable : list [
uidref ... ] } ]
```

The abstract type row_address is thus modified to be:

```
typeOr { RowAddress : ref, UIDAddress : uidref }
```

#### 2.2.2.2 Encoding Example

The encoding of the Next method, as defined by the Core Spec and the Stream Typing Removal Proposal, is as follows:

```
ArrayTableUID.Next F0 F2 Where F2 ArrayTable F0 F0 ref uidref F1 F1 F3 F3 F2 Count
uinteger F3 F1 F9 F0 0 0 0 F1
=>
F0 F2 ArrayTable F0 F0 ref uidref F1 F0 ref uidref F1 F3 F1 F0 0 0 0 F1
```

Based on this proposal, the above encoding would be:

```
ArrayTableUID.Next F0 F2 Where F0 F0 ref uidref F1 F1 F3 F2 Count uinteger F3 F1 F9 F0
0 0 0 F1
=>
F0 F0 F0 ref uidref F1 F0 ref uidref F1 F1 F1 F0 0 0 0 F1
```

## 2.2.3 Updated Abstract Types

This section identifies the specific changes required of relevant abstract types. Not all abstract types are presented – only those that are affected by this proposal.

#### 2.2.3.1 cell_block

This type represents a grouping of Named values that are used to identify a portion of a table. In messaging, this grouping is enclosed by List value delimiters, and each component is enclosed by Named value delimiters.

Because this is a group of Named values, its separate components are optional. However, there are default requirements if components are omitted. These requirements are as follows:

- Table – this Named value has the Name "Table" and a value that is a uid to a table.

  - If the value with Name "Table" is omitted, then the operation defaults to the table upon which the method was invoked.
  - Table shall be omitted if the method was invoked to operate on an object.

- startRow – this Named value has the Name "startRow". This Named value type can be assigned one of two values – either a uid of an object or a RowNumber that corresponds to the RowNumber value of an Array table row. Only one of these two values will appear in the messaging stream. The "typeOr" identifier, accompanying curly brackets ("{", "}"), and identifier ("'name' :") have no effect on the values as represented in the message.
  - If the value with Name "startRow" is omitted and the method is invoked to operate on a table, then the operation defaults to the first row of the table.
  - The value with Name "startRow" may be omitted if the method is invoked to operate on an object. If it is not omitted, it shall be the uid of the object on which the method is to operate, and shall be the same as the value assigned to endRow.
  - If both the value with Name "startRow" and the value with Name "endRow" are included in the type parameterization, then the value with Name "startRow" shall have the same type (uid or uinteger) as the value with Name "endRow".
- endRow – this Named value has the Name "endRow". This Named value type can be assigned one of two values – either a uid of an object or a RowNumber that corresponds to the RowNumber value of an Array table row. Only one of these two values will appear in the messaging stream. The "typeOr" identifier, accompanying curly brackets ("{", "}"), and identifier ("'name' :") have no effect on the values as represented in the message.
  - If the value with Name "endRow" is omitted and the method is invoked to operate on a table, then the operation defaults to the last row of the table.
  - The value with Name "endRow" shall be omitted if the method is invoked to operate on an object. If it is not omitted, it shall be the uid of the object on which the method is to operate, and shall be the same as the value assigned to startRow.
  - If both the value with Name "startRow" and the value with Name "endRow" are included in the type parameterization, then the value with Name "endRow" shall have the same type (uid or uinteger) as the value with Name "startRow".
- startColumn – this Named value has the Name "startColumn". This Named value type has a max bytes value that is represented by here using the name abstract type.
  - If the value with Name "startColumn" is omitted, then the operation defaults to the first column of the table or object.
- endColumn – this Named value has the Name "endColumn". This Named value type has a max bytes value that is represented by here using the name abstract type.
  - if the value with Name "endColumn" is omitted, then the operation defaults to the last column of the table or object.

Format:

```
[ Table = uid, startRow = typeOr { UID : uid, Row : RowNumber }, endRow = typeOr { UID :
uid, Row : RowNumber }, startColumn = name, endColumn = name ]
```

#### 2.2.3.2 row_address

This abstract type is used to describe a parameter that can be either a ref or a uidref. It is similar to the alternative column type. For additional information on the component types (ref and uidref), see their respective entries in this section.

Only one of these two values will appear in the messaging stream. The "typeOr" identifier, accompanying curly brackets ("{", "}"), and identifier ("'name' :") have no effect on the values as represented in the message.

Format

```
typeOr { RowAddress : ref, UIDAddress : uidref }
```

In the message stream itself, the value will one of the following:

- ref
- uidref

## 2.2.4 Updated Method Signatures

This section identifies the specific changes required of relevant method signatures. Not all method signatures are presented – only those that are affected by this proposal.

### 2.2.4.1 CreateRow

```
TableUID.CreateRow [
        Row : row_data+ ]
=>
[ Result : typeOr { ArrayTable : list [ list [ ref, uidref ] ... ], ObjectTable : list [ uidref ... ] } ]
```

### 2.2.4.2 DeleteRow

```
TableUID.DeleteRow [
        Where : row_address,
        Count = uinteger ]
=>
[ Result : boolean ]
```

### 2.2.4.3 Get

```
TableUID.Get [
ObjectUID.Get [
        Cellblock : cell_block ]
=>
[ Result : typeOr { Bytes : Bytes, RowValues : list [ list [ ColumnName = Value ... ] ... ] } ]
```

### 2.2.4.4 Set

(This method signature is based on the approved Set Parameters proposal)

```
TableUID.Set [
ObjectUID.Set [
        Where = typeOr { UID : UID, Row : RowNumber },
        Values = typeOr { Bytes : bytes, RowValues : list [ list [ ColumnName = Value ... ] ... ] ]
        =>
```

[ ]

#### 2.2.4.5 Next

```
TableUID.Next [
        Where = row_address,
        Count = uinteger ]
=>
[ Result : TypeOr { ArrayTable : list [ [ ref, uidref ] ... ], ObjectTable : list
[ uidref ... ] } ]
```

#### 2.2.4.6 Authenticate

```
SPUID.Authenticate [
        Authority : uidref { AuthorityObjectUID },
        Proof = bytes ]
=>
[ Result : typeOr { Success : boolean, Response : bytes } ]
```

#### 2.2.4.7 Stir

```
SPUID.Stir[
        Value : typeOr { Input : integer, Internal : boolean } ]
=>
[ Result : boolean ]
```

#### 2.2.4.8 Decrypt

```
CredentialObjectUID.Decrypt [
        Input : typeOr { Data : bytes, Buffer : cell_block },
        BufferOut = cell_block ]
=>
[ Result : bytes ]
```

#### 2.2.4.9 Encrypt

```
CredentialObjectUID.Encrypt [
        Input : typeOr { Data : bytes, Buffer : cell_block },
        BufferOut = cell_block ]
=>
[ Result : bytes ]
```

#### 2.2.4.10 Sign

```
CredentialObjectUID.Sign
HashObjectUID.Sign[
        Input : typeOr { Data : bytes, Buffer : cell_block },
        BufferOut = cell_block ]
```

```
=>
[ Result : bytes ]
```

#### 2.2.4.11 Verify

```
CredentialObjectUID.Verify
HashObjectUID.Verify[
        Input : typeOr { Data : bytes, Buffer : cell_block},
        Data : typeOr { Proof : bytes, ProofBuffer : cell_block } ]
=>
[ Result : boolean ]
```

#### 2.2.4.13 Hash

```
HashObjectUID.HashCalc [
        Input : typeOr { Data : bytes, BufferIn : cell_block } ]
=>
[ Result : bytes ]
```

#### 2.2.4.14 HMAC

```
HashObjectUID.HMACCalc [
        Input : typeOr { Data : bytes, Buffer : cell_block } ]
=>
[ Result : bytes ]
```

#### 2.2.4.15 XOR

```
SPUID.XOR[
        PatternInput : uidref {ByteTable},
        DeletePattern : boolean,
        Input : typeOr { Data : bytes, BufferIn : cell_block },
        BufferOut = cell_block
        ]
=>
[ Result : bytes ]
```

## 2.2.5 Updated Text

In the Stream Typing Removal Proposal (section 2.1.3 in that proposal), TypeOr method parameters/abstract types are defined in this way:

> "The components of an abstract typeOr alternative type used in method signature pseudo-code are always presented as Named value pairs. As such, each typeOr component will be represented on the interface as a Named value pair. Note that the typeOr itself may be an optional parameter or result, and as such this type could represent an instance of an embedded Named value pair (i.e. Name1 = Name2 = Value, where the value of Name2 is "Value" and the value of Name1 is "Name2 = Value")."

This following text is intended to update that description (and to be subsequently adapted as necessary for inclusion in the Core Specification):

> "The components of an abstract typeOr alternative type used in method signature pseudo-code are presented as "'name' : value". Each typeOr component will be represented on the interface as only

the value – the colon (":") indicates that the name identifier is for informational purposes and is ignored."

Additionally, the following paragraph is intended to update the section (and to be subsequently adapted as necessary for inclusion in the Core Specification):

"In typeOr types, when represented in abstract types or method signatures, a colon (":") indicates that the string to the left of the colon is only a pseudo-code identifier associated with the value to the right of the colon, and the value to the right of the colon is the actual value to be transmitted on the interface."

*End Proposal*

10.2.14 Level 0 Capabilities Discovery

*Start Proposal*

# Level 0 Capabilities Discovery

Authors: Jim Hatfield, Seagate Technology
Danny Ybarra, Western Digital
Revision:

| | | |
|---|---|---|
| 0.1 | 10.25.07 | Start of draft [Hines] |
| 0.2 | 11.05.07 | Revision presented to SIIF F2F meeting [Hatfield] |
| 0.3 | 11.07.07 | Addressed review comments, put in SWG proposal format [Hatfield] |
| 0.4 | 01.10.08 | Added generic SSC discovery feature, renamed the function to "Level 0 discovery [Hatfield/Ybarra/Hines] |
| 0.5 | 01.15.08 | Feedback during reviews [Hatfield/Ybarra/Hines] |
| 0.6 | 01.30.08 | Feedback during 1/25/08 review [Hatfield/Ybarra] |
| 0.7 | 03.10.08 | Feedback during several reviews [Hatfield/Ybarra] |
| 0.71 | 03.12.08 | Feedback during F2F reviews [Hatfield/Ybarra] |
| 0.72 | 03.13.08 | Feedback during F2F reviews [Hatfield/Ybarra] |
| 0.73 | 03.18.08 | Feedback during reviews [Hatfield/Ybarra] |
| 0.74 | 03.21.08 | ***Final draft: as voted on and accepted*** [Hatfield/Ybarra] |

# 3 Proposal – Core Spec Changes

(Place this section in the Core spec. as new section 3.3.6 in the Interface Communications
section.)

## 3.3.6 Level 0 Discovery

The LEVEL 0 DISCOVERY command provides a host with some basic information about TPer
capabilities; both current and potential. More detailed information is obtainable through SP
operations.

### 3.3.6.1 IF-SEND Command

IF-SEND command, with
Security Protocol = 01h
Security Protocol Specific = 0001h
Transfer Length= n/a

There is no IF-SEND command defined for Level 0 Discovery.
The TPer SHALL transfer all of the data from the host, SHALL discard it, and return 'good' status
to the host.

#### 3.3.6.2 IF-RECV Command

IF-RECV command, with
Security Protocol = 01h
Security Protocol Specific = 0001h
Allocation Length = maximum length of the LEVEL 0 DISCOVERY response data that the host elects to receive.
This IF-RECV command may be processed at any time, without regard to sessions or prior
authentication.

If the Allocation Length is less than the size of the LEVEL 0 DISCOVERY response data that is
available, the TPer SHALL return the requested amount of data, even if it is truncated.

If the Allocation Length is greater than the size of the LEVEL 0 DISCOVERY response data:
a) An ATA device shall pad with zeros to the Allocation Length requested.
b) A SCSI target with INC_512 set to one shall pad with zeroes to the next 512-byte boundary. If INC_512 is set to zero, the target shall only transfer the available number of bytes.

The LEVEL 0 DISCOVERY response data (see Table 1) consists of a header field and zero or
more variable length feature descriptors. A TPer SHALL not include feature descriptors for
features that it does not implement. The data does not contain any ComPackets, and is not
contained within a ComPacket.

**Table 1 — LEVEL 0 DISCOVERY response data format**

| Byte \ Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| **0 – 47** | Level 0 Discovery header (see (see Table 2 | | | | | | | |
| **48 – n** | Feature Descriptor(s) (see 3.3.6.3) | | | | | | | |

**Table 2 — LEVEL 0 Discovery header**

| Byte \ Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| 0 | (MSB) | | | | | | | |
| 1 | | | | Length of Parameter Data | | | | |
| 2 | | | | | | | | |
| 3 | | | | | | | | (LSB) |
| 4 | (MSB) | | | Data structure revision | | | | |

| | | | |
|---|---|---|---|
| 5 | | | |
| 6 | | | |
| 7 | | | (LSB) |
| 8 | (MSB) | Reserved | |
| … | | | |
| 15 | | | (LSB) |
| 16 | (MSB) | Vendor Specific | |
| … | | | |
| 47 | | | (LSB) |

#### *3.3.6.2.1 Length of parameter data*

Indicates the total number of bytes that are valid in the level 0 discovery header and all of the
feature descriptors returned, not including this field.

#### *3.3.6.2.2 Data structure version number*

This version number describes the format of the level 0 discovery header returned. The value
SHALL be 00000001h

#### *3.3.6.2.3 Vendor Specific*

These bytes are vendor specific.

### 3.3.6.3 Features - Overview

A feature is a set of capabilities that may be implemented in a TPer. A Host may discover the
capabilities and properties of a TPer by examining its feature descriptors. Features that are
implemented by a TPer SHALL be indicated by the presence of a feature descriptor.
The feature descriptors SHALL be returned in the LEVEL 0 DISCOVERY response data in order
of increasing feature code values. Features that are not implemented SHALL NOT be returned.
Table 3 contains the list of defined feature codes.

**Table 3 –Feature Codes**

| Feature Code | Feature Name | Description |
|---|---|---|
| 0000h | Reserved | |
| 0001h | TPer feature | See 3.3.6.4 |
| 0002h | Locking feature | See 3.3.6.5 |
| 0003h – 00FFh | Reserved | |
| 0100h – 01FFh | Enterprise SSC | See the Enterprise_A SSC specification |
| | | |
| 0200h – 02FFh | Opal SSC | See the Opal SSC specification. |
| | | |
| 0300h - 03FFh | Optical SSC | See the Optical SSC specification |
| 0400h - BFFFh | Reserved | |
| C000h - FFFFh | Vendor Unique | Vendor specific features |

All feature descriptors SHALL conform to the general format defined in Table 4.

**Table 4 — Feature Descriptor template format**

| Byte \ Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| 0 | (MSB) | | | Feature Code | | | | |
| 1 | | | | | | | | (LSB) |
| 2 | Version | | | | Reserved | | | |
| 3 | Length | | | | | | | |
| 4 – n | Feature Dependent Data | | | | | | | |

#### *3.3.6.3.1.1 Feature Code*

The Feature Code field SHALL identify a feature (see Table 3) implemented by the TPer.

#### *3.3.6.3.1.2 Version*

The Version field describes the format of the data returned. Future versions of a feature SHOULD be backward compatible; incompatible changes SHOULD be included in a different feature.

#### *3.3.6.3.1.3 Length*

The Length field indicates the length of the Feature Dependent Data (in bytes) that follow this header. This field SHALL be an integral multiple of 4.

### 3.3.6.4 TPer feature (0001h)

This information reports support for various TPer parameters. This mandatory feature SHALL always be returned in the Level 0 Discovery response.

These parameters indicate whether the TPer supports a variety of features. Having a given "support' flag true does not imply that the feature is required or enabled. Actually enabling a feature may require personalization of the TPer.

**Table 5 — TPer feature**

| Byte \ Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| 0 | (MSB) | | | Feature Code | | | | |
| 1 | | | | | | | | (LSB) |
| 2 | Version | | | | Reserved | | | |
| 3 | Length | | | | | | | |
| 4 | Reserved | ComID Mgmt Supported | Reserved | Streaming Supported | Buffer Mgmt Supported | ACK/NAK Supported | Asynch Supported | Sync Supported |
| 5 - 15 | Reserved | | | | | | | |

The Feature Code field SHALL be set to 0001h.
The Version field SHALL be set to 1h.
The Length field SHALL be set to 0Ch.

#### *3.3.6.4.1 SyncSupported*

SyncSupported SHALL be set to one if the TPer supports the Synchronous Protocol, otherwise SyncSupported SHALL be cleared to zero.

#### *3.3.6.4.2 AsynchSupported*

AsynchSupported SHALL be set to one if the TPer supports the Asynchronous Protocol, otherwise AsynchSupported SHALL be cleared to zero.

#### *3.3.6.4.3 ACK/NAKSupported*

ACK/NAKSupported SHALL be set to one if the TPer supports transmission ACK/NAK flow control for communications, otherwise ACK/NAKSupported SHALL be cleared to zero.

#### *3.3.6.4.4 BufferMgmtSupported*

BufferMgmtSupported SHALL be set to one if the TPer supports buffer management flow control for communications, otherwise BufferMgmtSupported SHALL be cleared to zero.

#### *3.3.6.4.5 StreamingSupported*

StreamingSupported SHALL be set to one if the TPer supports the streaming protocol, otherwise StreamingSupported SHALL be cleared to zero.

#### *3.3.6.4.6 ComID Management Supported*

Shall be set to one if the TPer supports ComID management using Protocol ID 02h, otherwise SHALL be cleared to zero.

### 3.3.6.5 Locking Feature (0002h)

This information indicates support for an issued Locking template. This mandatory feature SHALL always be returned in the Level 0 Discovery response.

**Table 23 — Locking feature descriptor**

| Byte \ Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| 0 | (MSB) | | | Feature Code | | | | |
| 1 | | | | | | | | (LSB) |
| 2 | Version | | | | Reserved | | | |
| 3 | Length | | | | | | | |
| 4 | Reserved | | MBR Done | MBR Enabled | Media Encryption | Locked | Locking Enabled | Locking Supported |
| 5 - 15 | Reserved | | | | | | | |

The Feature Code field SHALL be set to 0002h.
The Version field SHALL be set to 1h.
The Length field SHALL be set to 0Ch.

#### *3.3.6.5.1 LockingSupported*

LockingSupported SHALL be set to one if the TPer supports the Locking template; otherwise
LockingSupported SHALL be set to zero.

#### *3.3.6.5.2 LockingEnabled*

LockingEnabled SHALL be set to one if an SP that incorporates the Locking template is in any

state other than nonexistent or manufactured-inactive; otherwise LockingEnabled SHALL be set
to zero.

#### *3.3.6.5.3 Locked*

Locked SHALL be set to one if LockingEnabled is set to one, and one or more LBA ranges in
the Locking table have either (ReadLockEnabled=True and ReadLocked=True) or
(WriteLockEnabled=True and WriteLocked=True); otherwise Locked SHALL be set to zero.

#### *3.3.6.5.4 MediaEncryption*

MediaEncryption SHALL be set to one if the TPer supports media encryption; otherwise
MediaEncryption SHALL be set to zero.

#### *3.3.6.5.5 MBREnabled*

MBREnabled SHALL be set to one if LockingEnabled is set to one, and the MBRControl and
MBR tables are implemented, and that the MBRControl table's Enabled column has a value of
"True"; otherwise MBREnabled SHALL be set to zero.

#### *3.3.6.5.6 MBRDone*

MBRDone SHALL be set to one if MBREnabled is set to one, and the MBRControl table's Done
column has a value of "True"; otherwise MBRDone SHALL be set to zero.

### 3.3.6.6 Common SSC feature information

This information is supplied as part of every reported SSC feature.

**Table 7 — Common SSC Information**

| Byte \ Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| 0 | (MSB) | | | Base ComID | | | | |
| 1 | | | | | | | | (LSB) |
| 2 | (MSB) | | | Number of ComIDs | | | | |
| 3 | | | | | | | | (LSB) |
| 4 - 15 | Reserved for future common SSC parameters | | | | | | | |

#### *3.3.6.6.1 Base ComID*

This is the lowest static, pre-assigned ComID that the SSC supports for Protocol ID=01h
sessions.

#### *3.3.6.6.2 Number of ComIDs*

This specifies the number of static, pre-assigned ComIDs that the SSC supports for Protocol
ID=01h sessions, starting at the Base ComID.

# 4 Proposal – Enterprise_A SSC Spec Changes

(Place the following section in the Enterprise_A SSC specification, in the Level 0
Discovery section.)

## 4.3.6 Enterprise_A SSC Feature (0100h)

This feature SHALL be returned in the Level 0 Discovery response if the Enterprise_A
SSC is implemented.

**Table 8 – Enterprise SSC Descriptor Format**

| Byte \ Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| 0 | (MSB) | | | Feature Code | | | | |
| 1 | | | | | | | | (LSB) |
| 2 | Version | | | | Reserved | | | |
| 3 | Length | | | | | | | |
| 4 | (MSB) | | | Base ComID | | | | |
| 5 | | | | | | | | (LSB) |
| 6 | (MSB) | | | Number of ComIDs | | | | |
| 7 | | | | | | | | (LSB) |
| 8 - 19 | Reserved for future common SSC parameters | | | | | | | |
| 20 | (MSB) | | | Reserved for Enterprise_A SSC | | | | |
| 21 | | | | | | | | |
| 22 | | | | | | | | |
| N | | | | | | | | (LSB) |

The Feature Code field SHALL be set to 0100h.
The Version field SHALL be set to 1h.
The Length field SHALL be set to 0x10.
The Base ComID SHALL be set to 07FEh.
The Number of ComIDs SHALL be set to 0002h.

*End Proposal*

10.2.15 ComPacket/Packet/Subpacket Header Alignment Proposal

*Start Proposal*

# ComPacket/Packet/Subpacket Header Alignment Proposal

Author: Darren Lasko (Fujitsu)

Revision

v0.1: 10/30/2007 start of the document

v0.2: 10/31/2007 Corrected the size of the Pad field in the Data Subpacket Payload

v0.3: 11/7/2007 Modified the Subpacket headers to group the Reserved fields into a single field of type uinteger_6; removed the Reserved field from the Credit Control Subpacket payload and changed the type of the Credit field to uinteger_4; added a section to modify the type for InitialCredit in StartSession/SyncSession to uinteger_4.

# 1 Goal

The goal of this proposal is to align all ComPacket, Packet, and Subpacket header fields to be 32-bit (4 byte) aligned, and to require that all Subpacket payloads contain a multiple of 4 bytes of data. This will ensure that all Compacket/Packet/Subpacket header and payload boundaries are 32-bit aligned (even for multiple Packets per ComPacket and multiple Subpackets per Packet). In addition to enabling easier processing of the headers, the alignment will also cause the memory locations for cryptographic processing of secure messages (encryption/decryption/MAC) to be 32-bit aligned.

# 2 Motivation

Currently, the **AckType** field in the Packet header is of type **uinteger_2**, which causes the **Acknowledgment** and **Length** fields to not be 32-bit aligned. It also causes the location in memory where SecureData encryption/decryption begins to not be 32-bit aligned.
Requiring all header fields and header/payload boundaries be 32-bit aligned will help simplify the header processing and the encryption/decryption/MAC of secure messages.

# 3 Proposal

## 3.1 ComPacket Header Modifications

No changes are necessary to the ComPacket header. However, any future modifications to the ComPacket header shall maintain the 32-bit alignment of all fields in the ComPacket header.
The value of the **Length** field in the ComPacket header will automatically be a multiple of 4 by virtue of the modifications defined in section 3.2 through 3.6 of this proposal.

## 3.2 Packet Header Modifications

Add a field of "**Reserved: uinteger_2**" between **SeqNumber** and **AckType** fields.
The value of the **Length** field in the Packet header will automatically be a multiple of 4 by virtue of the modifications defined in section 3.3 through 3.6 of this proposal.

## 3.3 Subpacket Header Modifications

Move the **Reserved** field to come before the **Kind** field, and change the **Reserved** field to be of type **uinteger_6** in each of the defined Subpacket headers.

## 3.4 Credit Control Subpacket Modifications

The **Length** field in the Subpacket header shall contain a value of 0x00000004.
In the Payload, the **Credit** field shall be changed to type **uinteger_4**.

## 3.5 Data Subpacket Modifications

In the payload, add a field after the **Data** field of "**Pad: bytes{(4 – (Length mod 4)) mod 4}**". The value of the Pad bytes shall be 0x00.
*Informative Note:* The receiver of a Subpacket can unambiguously know how many bytes of real data there are by examining the **Length** field in the Subpacket header. The receiver can also unambiguously know how many bytes of pad there are by calculating ((4 – (**Length** modulo 4)) modulo 4).

## 3.6 New Subpacket Kinds

Any new Subpacket kind that is added to a future revision of the SWG Core specification shall maintain the 32-bit alignment of boundaries between headers and payloads.

## 3.7 StartSession/SyncSession Parameters

The **InitialCredit** parameter for both **StartSession** and **SyncSession** shall be changed to type **uinteger_4** in order to maintain consistency with the type change to the **Credit** field defined in section 3.4.

*End Proposal*

10.2.16 Session Manager Session Number

*Start Proposal*

# Session Manager Session Number

Author: Jason Cox, Doug Philips

Revision:
0.1 08.03.07 Start of draft

# 1 Goal

The goal of this proposal is:

- To indicate the changes required to the TCG Storage Architecture Core Specification to indicate that Packets that are transmitted on the Session Manager layers, and contain Session Manager methods, shall have a Packet.SessionNumber value of 0.

# 2 Proposal

Currently, the Core Spec indicates that the SessionNumber field of packets transmitted on the Session Manager communications layer has a value of 0's for the TPer Session Number (TSN) portion, and the Host Session Number (HSN), which in this instance does not identify the session but is used as a "routing aid". This addressing is inconsistent and provides an artificial barrier to simultaneous startup of multiple sessions that can be eliminated by setting the entire SessionNumber field to 0's.

## 2.1 Concepts

By requiring all Session Manager traffic to utilize a Packet.SessionNumber value of 0's,

- The Properties method usage is not inconsistently associated with a phantom session.
- Multiple session startup methods may be sent in a single Packet.

### 2.1.1 Session Manager Method Usage

#### 2.1.1.1 Properties

The Properties method is a Session Manager layer method, but is never associated with a specific Packet.SessionNumber value. Requiring all Session Manager layer methods, such as the Properties method, to utilize Packet.SessionNumber values of 0 removes any inconsistencies with between the way this method is transmitted and the way other Session Manager layer methods are transmitted.

#### 2.1.1.2 Session Startup with SessionNumber=0's

Requiring the use of a non-0's Packet.SessionNumber with a session startup method provides an arbitrary barrier to any Host Session Manager that attempts to start multiple sessions simultaneously (transmitting multiple StartSession or StartTrustedSession methods in the same packet). The same barrier occurs in conjunction with aggregation by the TPer of corresponding multiple SyncSession or SyncTrustedSession methods for transmission back to the host. Associating Session Manager layer methods with a Packet.SessionNumber of 0's removes artificial limitations on requiring only one StartSession per packet.
The Host Session Number is identified in the StartSession method. Once the session has successfully started, this value becomes part of the Packet.SessionNumber. Until that occurs, the Host Session Manager and TPer Session Manager simply track the session startup methods transmitted on the single session with SessionNumber of 0's.

### 2.1.2 Core Specification Changes

#### 2.1.2.1 Session Manager Layer Method SessionNumber Requirements

The Control Sessions section of the Core Specification, section 3.3.4.2 shall be modified in a manner that reflects the changes suggested in this section.
All Session Manager Layer Methods shall be transmitted in packets where Packet.SessionNumber = 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00.
Session Manager layer methods are:

- Properties

- StartSession
- SyncSession
- StartTrustedSession
- SyncTrustedSession
- CloseSession

Once a session has started (the session startup protocol has completed successfully), data may be transmitted for that newly started session. The Packet.SessionNumber for that session shall be the concatenation of the TSN and HSN, as described in the Core Specification, where HSN is initially transmitted in the StartSession method and TSN is initially transmitted in the SyncSession method.

*End Proposal*

### 10.2.17 Next Method Behavior Modification

*Start Proposal*

# Next Method Behavior Modification

Author: Jason Cox
Revision:
1 03/25/2008 Start of draft
2 03/28/2008 Updated starting row description per discussion



# 1 Goal

The goal of this proposal is:

- To change the behavior of the Next method when the Count optional parameter is omitted.

# 2 Proposal

## 2.1 Concepts

Currently, the Core Spec indicates that the behavior of the Next method when the Count parameter is such that the value defaults to 1. So, if Next is invoked with an omitted Count parameter, the scope of the Next method's return value is a single row.
This proposal suggests that when the Next method is invoked and the Count parameter is omitted, the following behavior be exhibited:

- If both the Where parameter and the Count parameter are omitted, the scope of the Next method's return value is the entire table.
- If the Where parameter is included in the invocation and the Count parameter is omitted, the scope of the Next method's return value begins at the starting point in the table's ordering indicated by the row following that identified by the Where parameter, and ends at the end of the table's row ordering.

*End Proposal*

### 10.2.18 K_AES media encryption key tables

*Start Proposal*

## K_AES media encryption key tables

Author: Cyril Guyot
Revision: 0.2

Changes:
0.1 – first draft
0.2 – added usage text for the Key column of the K_AES_128 and K_AES_256 tables



# 1 Goals

C_AES credentials, as defined in the TCG SWG Core Specification, can be used in at least two fundamentally different processes: as media encryption keys and as authentication credentials. One consequence of this design is a fair amount of complexity in the access control set-up to ensure that a user is not allowed to use for authentication keys meant for media encryption - and vice-versa -.

Moreover, access control rules for Set and Get methods on media encryption keys might typically be fairly different from the ones associated with authentication. For instance one could envision a TPer that might disallow entirely Set and Get methods on media encryption keys, but for which allowing Get or Set on authentication credentials would be required to bootstrap a symmetric key authentication mechanism.

This proposal solves those issues by adding two new K_AES_128 and K_AES_256 key tables dedicated to media encryption keys.

# 2 Proposal

This proposal adds a new group of tables to the Locking template: the Media Key Tables of which
two new tables K_AES_128 and K_AES_256 are defined. Those tables contain the media encryption keys which are pointed to by the Locking Table cells in the Active Key and Next
Key columns.

Active Key and Next Key column types are modified to only allow pointing to K_AES_128

and K_AES_256 tables.

Finally, and as a consequence of the newly defined type, the Credential column of the
Authority tables does not allow pointing to K_AES_128 or K_AES_256 key tables.

**Media Encryption Key Table Group - K_AES_128 (Object Table)**

**Table 01 K_AES_128 Table Description**

| Column | IsIndex | Type | Description |
|---|---|---|---|
| UID | | uid | This is the unique identifier for this object. (Read-only) |
| Name | Yes | name | This is the name of this object. (Read-only for pre-personalization objects) |
| CommonName | Yes | common_name | A name that may be shared among multiple K_AES_128 objects (Read-only for prepersonalization objects) |
| Key | | bytes{max=32} | Key |
| Mode | | symmetric_mode | Defines the mode with which this key shall be used. |

The Mode column defines the encryption mode with which this key shall be used.
Valid values are ECB, CBC, CFB, OFB, GCM, CCM, CTR and MediaEncryption.
MediaEncryption mode permits a vendor-specific encryption mode. Any byte
beyond the first 16 in the Key column shall be ignored for the ECB, CBC, CFB,
OFB, GCM, CCM, CTR modes. For MediaEncryption mode, the content of the
Key column may be vendor-specific.

**Media Encryption Key Table Group - K_AES_256 (Object Table)**

**Table 02 K_AES_256 Table Description**

| Column | IsIndex | Type | Description |
|---|---|---|---|
| UID | | uid | This is the unique identifier for this object. (Read-only) |
| Name | Yes | name | This is the name of this |

| Column | IsIndex | Type | Description |
|---|---|---|---|
| | | | object. (Read-only for pre-personalization objects) |
| CommonName | Yes | common_name | A name that may be shared among multiple K_AES_256 objects (Read-only for prepersonalization objects) |
| Key | | bytes{max=64} | Key |
| Mode | | symmetric_mode | Defines the mode with which this key shall be used. |

The Mode column defines the encryption mode with which this key shall be used.
Valid values are ECB, CBC, CFB, OFB, GCM, CCM, CTR and MediaEncryption.
MediaEncryption mode permits a vendor-specific encryption mode. Any byte
beyond the first 32 in the Key column shall be ignored for the ECB, CBC, CFB,
OFB, GCM, CCM, CTR modes. For MediaEncryption mode, the content of the
Key column may be vendor-specific.

**Table 03 Default Type Table Values**

| ID | Name | Format | Size | Default | Description |
|---|---|---|---|---|---|
| …. | …. | …. | …. | …. | …. |
| 00 00 00 05 00 00 10 02 | cred_object_uidref | 8 | | | This is a reference type that shall be used specifically for uidrefs to credential objects. When performing type checking, as part of that type checking the TPer shall validate that |

| ID | Name | Format | Size | Default | Description |
|---|---|---|---|---|---|
| | | | | | this uidref is to an object in a credential table. |
| **00 00 00 05 00 00 10 03** | **mediakey_object_uidref** | **8** | | | **This is a reference type that shall be used specifically for uidrefs to media encryption key objects. When performing type checking, as part of that type checking the TPer shall validate that this uidref is to an object in a media encryption key table** |
| 00 00 00 05 00 00 12 01 | table_ref | 9 | | | This type is used to represent a uidref to a Table that is one of the set of all tables in the SP. |
| …. | …. | …. | …. | …. | …. |

**Table 04 Locking Table**

| Column | Type | Description |
|---|---|---|
| …. | …. | …. |
| LockOnReset | reset_types | Identifies the LBA range's storage-related |

| Column | Type | Description |
|---|---|---|
| | | locking behavior, *dependent on reset type*.<br>Note that both Read and Write Locking behavior on reset are controlled by this value.<br>An empty set means locking does not occur on any reset. |
| ActiveKey | **mediakey_object_uidref** | Points to the present encryption key for this LBA range |
| NextKey | **mediakey_object_uidref** | Points to the next encryption key for this LBA range |
| ReEncryptState | reencrypt_state | This is the present Re-encryption State for this LBA range.<br>ReEncryptState reports the TPer's response to re-encrypt requests.<br>(Read-only) |
| …. | …. | …. |

**NextKey:** This column identifies the LBA range's next media encryption key. This value and the referenced **media encryption key** object shall be writable when the value of the ReEncryptState column is IDLE only. Otherwise, attempts to invoke an of the Set, Delete, or DeleteRow methods on the associated credential object shall return an error.

User Data shall be returned to clear text when the key value stored at NextKey is 00s AND Re-encryption has been requested

**Table 05 Table UIDs**

| UID of Table Object | UID of Table | Table Name | Template |
|---|---|---|---|
| 00 00 00 01 00 00 00 01 | 00 00 00 01 00 00 00 00 | Table | Base |

| UID of Table Object | UID of Table | Table Name | Template |
|---|---|---|---|
| 00 00 00 01 00 00 00 02 | 00 00 00 02 00 00 00 00 | SPInfo | Base |
| 00 00 00 01 00 00 00 03 | 00 00 00 03 00 00 00 00 | SPTemplates | Base |
| 00 00 00 01 00 00 00 04 | 00 00 00 04 00 00 00 00 | Column | Base |
| 00 00 00 01 00 00 00 05 | 00 00 00 05 00 00 00 00 | Type | Base |
| 00 00 00 01 00 00 00 06 | 00 00 00 06 00 00 00 00 | MethodID | Base |
| 00 00 00 01 00 00 00 07 | 00 00 00 07 00 00 00 00 | Method | Base |
| 00 00 00 01 00 00 00 08 | 00 00 00 08 00 00 00 00 | ACE | Base |
| 00 00 00 01 00 00 00 09 | 00 00 00 09 00 00 00 00 | Authority | Base |
| 00 00 00 01 00 00 00 0A | 00 00 00 0A 00 00 00 00 | Certificates | Base |
| 00 00 00 01 00 00 00 0B | 00 00 00 0B 00 00 00 00 | C_PIN | Base |
| 00 00 00 01 00 00 00 0C | 00 00 00 0C 00 00 00 00 | C_RSA_1024 | Base |
| 00 00 00 01 00 00 00 0D | 00 00 00 0D 00 00 00 00 | C_RSA_2048 | Base |
| 00 00 00 01 00 00 00 0E | 00 00 00 0E 00 00 00 00 | C_AES_128 | Base |
| 00 00 00 01 00 00 00 0F | 00 00 00 0F 00 00 00 00 | C_AES_256 | Base |
| 00 00 00 01 00 00 00 10 | 00 00 00 10 00 00 00 00 | C_EC_160 | Base |
| 00 00 00 01 00 00 00 11 | 00 00 00 11 00 00 00 00 | C_EC_192 | Base |
| 00 00 00 01 00 00 00 12 | 00 00 00 12 00 00 00 00 | C_EC_224 | Base |
| 00 00 00 01 00 00 00 13 | 00 00 00 13 00 00 00 00 | C_EC_256 | Base |
| 00 00 00 01 00 00 00 14 | 00 00 00 14 00 00 00 00 | C_EC_384 | Base |
| 00 00 00 01 00 00 00 15 | 00 00 00 15 00 00 00 00 | C_EC_521 | Base |

| UID of Table Object | UID of Table | Table Name | Template |
|---|---|---|---|
| 00 00 00 01 00 00 00 16 | 00 00 00 16 00 00 00 00 | C_EC_163 | Base |
| 00 00 00 01 00 00 00 17 | 00 00 00 17 00 00 00 00 | C_EC_233 | Base |
| 00 00 00 01 00 00 00 18 | 00 00 00 18 00 00 00 00 | C_EC_283 | Base |
| 00 00 00 01 00 00 02 01 | 00 00 02 01 00 00 00 00 | TPerInfo | Admin |
| 00 00 00 01 00 00 02 02 | 00 00 02 02 00 00 00 00 | Properties | Admin |
| 00 00 00 01 00 00 02 03 | 00 00 02 03 00 00 00 00 | CryptoSuite | Admin |
| 00 00 00 01 00 00 02 04 | 00 00 02 04 00 00 00 00 | Template | Admin |
| 00 00 00 01 00 00 02 05 | 00 00 02 05 00 00 00 00 | SP | Admin |
| 00 00 00 01 00 00 04 01 | 00 00 04 01 00 00 00 00 | ClockTime | Clock |
| 00 00 00 01 00 00 06 01 | 00 00 06 01 00 00 00 00 | H_SHA_1 | Crypto |
| 00 00 00 01 00 00 06 02 | 00 00 06 02 00 00 00 00 | H_SHA_256 | Crypto |
| 00 00 00 01 00 00 06 03 | 00 00 06 03 00 00 00 00 | H_SHA_384 | Crypto |
| 00 00 00 01 00 00 06 04 | 00 00 06 04 00 00 00 00 | H_SHA_512 | Crypto |
| 00 00 00 01 00 00 0A 01 | 00 00 0A 01 00 00 00 00 | Log | Log |
| 00 00 00 01 00 00 0A 02 | 00 00 0A 02 00 00 00 00 | LogList | Log |
| 00 00 00 01 00 00 08 01 | 00 00 08 01 00 00 00 00 | Locking_Info | Locking |
| 00 00 00 01 00 00 08 02 | 00 00 08 02 00 00 00 00 | Locking | Locking |
| 00 00 00 01 00 00 08 03 | 00 00 08 03 00 00 00 00 | MBR_Control | Locking |
| 00 00 00 01 00 00 08 04 | 00 00 08 04 00 00 00 00 | MBR | Locking |
| 00 00 00 01 00 00 08 05 | 00 00 08 05 00 00 00 00 | K_AES_128 | Locking |

| UID of Table Object | UID of Table | Table Name | Template |
|---|---|---|---|
| 00 00 00 01 00 00 08 06 | 00 00 08 06 00 00 00 00 | K_AES_256 | Locking |

*End Proposal*

### 10.2.19 Protocol Stack Reset Command

*Start Proposal*

# Protocol Stack Reset Command

Author: Jason Cox, Manuel Offenberg

Revision: 1 [Initial Draft]
2 [Clarifications]
3 [Added error conditions]
4 [Clarifications]
5 [Removed one condition for buffer clearing, changed ComID error]
6 [Changed “No Response Available” and command processing during reset]
7 [Changed “Pending” payload]
8 [Updated text per SWG Telecon]
9 [mod'ed available data length notation to allow for blocks or bytes]
10 [reordered fields]
11 [clarification of request/response fields, buffer reset conditions]
12 [added clarification for response upon receipt of a 2nd request]
13 [final version, clarified operation to indicate only based on ComID requested]

# 1 References

[1] TCG Storage Workgroup: Core Specification 1.0, draft 0.9
[2] TCG Storage Workgroup: Storage Interface Interactions Specification 1.0, draft 0.3

# 2 Goal

The goal of this proposal is:

- Propose a mechanism to reset the Security Protocol stack for a given ComID. This allows the host to re-synch with the TPer in case one or both are in an undetermined state while communicating with each other.

# 3 Proposal

The Core Spec provides a facility for maintenance and management of ComIDs. A specific version of the IF-SEND command, called a HANDLE_COMID_REQUEST, is used to inquire about or manage a particular

ComID. The result of the command is retrieved via a specific IF-RECV command, called a GET_COMID_RESPONSE.
This proposal suggests adding a new Request code to the HANDLE_COMID_REQUEST command as defined in the Communication Layer Protocol (see [1]). The proposed Request code is STACK_RESET (02h) and its command block payload is defined as:

| | |
|---|---|
| Bytes 0 to 3 : | Extended ComID value |
| Bytes 4 to 7 : | STACK_RESET (00 00 00 02h) |
| Bytes 8 to TRNSFLEN – 1: | Reserved (0s) |

TRNSFLEN is defined as number of bytes transferred via the interface.

The device SHALL return an “Invalid Transfer Length parameter on IF-SEND” TPer Error [2] if less than 8 bytes or more than 512 bytes are sent to the device.

Depending on the SSC, the device MAY return:

an “Invalid ComID parameter on IF-SEND” Error, or

an “Other Invalid CDB parameter” Error [2] if the ComID value in the IF-SEND for the HANDLE_COMID_REQUEST command represents a non Active ComID (refer to [1] for Active ComIDs).

Once received, the TPer SHALL reset the Security Protocol stack for the ComID value defined in bytes 0-3 of the command block payload. While resetting the stack, the Tper SHALL NOT process any command for that ComID received via an IF-SEND on Protocol ID 01h. A Security Protocol stack reset results in:

1. All open sessions for that ComID SHALL be aborted;
2. All uncommitted transactions SHALL be aborted. CloseSession methods SHALL NOT be prepared by the TPer;
3. All pending session startup activities occurring on that ComID SHALL be aborted;
4. All TCG command and response buffers SHALL be invalidated for that ComID;
5. All related method processing occurring on that ComID SHALL be aborted;
6. The protocol stack SHALL reset to its initial state for that ComID only;
7. All communications properties (set via Properties method) and ComID associated properties for that ComID SHALL be reset to their default values;
8. No notification of these events SHALL be sent to the host.

The response SHALL be returned via the GET_COMID_RESPONSE (IF-RECV) command. The response payload is defined as:

| | |
|---|---|
| Bytes 0 to 3: | Extended ComID |
| Bytes 4 to 7: | STACK_RESET (00 00 00 02h) |
| Bytes 8 to 9: | Reserved (00 00h) |
| Bytes 10 to 11: | Available Data Length in bytes (00 04h) |
| Bytes 12 to 15: | Success (00 00 00 00h) / Failure (00 00 00 01h) |
| Bytes 16 to TRNSFLEN - 1: | Reserved (0s) |

Success (00h): the protocol stack has been reset for the specified ComID;
Failure (01h): the protocol stack has not been reset for the specified ComID;

A “Pending” payload is defined as:

| | |
|---|---|
| Bytes 0 to 3: | Extended ComID |
| Bytes 4 to 7: | STACK_RESET (00 00 00 02h) |
| Bytes 8 to 9: | Reserved (0s) |
| Bytes 10 to 11: | Available Data Length in bytes (00 00h) |
| Bytes 12 to TRNSFLEN - 1: | Reserved (0s) |

A “No Response Available” payload is defined as:

| | |
|---|---|
| Bytes 0 to 3: | Extended ComID |
| Bytes 4 to 7: | ZERO (00 00 00 00h) |
| Bytes 8 to 9: | Reserved (0s) |

| | |
|---|---|
| Bytes 10 to 11: | Available Data Length in bytes (00 00h) |
| Bytes 12 to TRNSFLEN - 1: | Reserved (0s) |

The response SHALL be cleared from the response buffer if one of the following conditions is true:

1. The host retrieves the entire response via the GET_COMID_RESPONSE command;
2. The device is hard-reset or power-cycled.
3. Another HANDLE_COMID_REQUEST is made for that ComID.

If the STACK_RESET is still processing and another HANDLE_COMID_REQUEST is received, the STACK_RESET SHALL complete but no response for that STACK_RESET command will be available.

Reserved bytes SHOULD be set to zero and SHALL be ignored by both host and device.
The device SHALL return “No Response Available” if:

1. No HANDLE_COMID_REQUEST command preceded the GET_COMID_RESPONSE command;
2. An error is detected in the HANDLE_COMID_REQUEST command payload.

The device SHALL return “Pending” if:

1. The host retrieves the command result via the GET_COMID_RESPONSE command while the stack reset is in progress for that specific ComID.

Note: Changes suggested to the ordering or the meaning of the bytes in the request (specifically, the re-ordering of the Reserved and Available Data Length fields) are to be migrated into the Core Specification for all request codes defined in the Core Specification for the HANDLE_COMID_REQUEST and GET_COMID_RESPONSE commands.

# 4 Informative

The host is not required to retrieve the status via GET_COMID_RESPONSE, i.e. successful retrieval of the STACK_RESET response by the host does not have an effect on the execution of the command itself.

The TCG Core Specification defines HANDLE_COMID_REQUEST as:

| | |
|---|---|
| Command: | IF-SEND |
| Protocol ID: | 02 |
| Transfer Length: | nn nn |
| ComID: | Allocated ComID |

The TCG Core Specification defines GET_COMID_RESPONSE as:

| | |
|---|---|
| Command: | IF-RECV |
| Protocol ID: | 02 |
| Transfer Length: | nn nn |
| ComID: | Allocated ComID |

*End Proposal*

### 10.2.20 Properties Clarification

*Start Proposal*

# Properties Clarification

Author: Jason Cox, Doug Philips
Revision: 1 Feb 21, 2008
Revision: 2 June 6, 2008

# 1 Goal

The goal of this proposal is:

- To remove obsolete values from the Properties method response.
- To introduce additional values required to be reported in the Properties response.
- To clarify the intent/purpose/scope of the Properties method.

# 2 Proposal

## 2.1 Clarification of Purpose

The purpose of the Properties method and response is to permit the Host and TPer to exchange information needed for setting up sessions, without having to set up a session first.

## 2.2 Properties Removed

- SessionVersion
- RealTimeClock

## 2.3 Properties Added

- MaxPackets
- MaxSubpackets
- MaxMethods

## 2.4 Core Spec – Properties Method

### 2.4.1 TPer Properties Method

#### 2.4.1.1 Properties (Method)

The Properties method pertains to the exchange of session-related metadata and settings between the host and the TPer prior to session start-up. The purpose of the Properties method is to permit the host and the

TPer to exchange the information required for session startup and maintenance, without the need to first start a session.

```
SMUID.Properties[ HostProperties = list [ name = value ... ] ]
=>
SMUID.Properties[ Properties : list [ name = value ... ], HostProperties = list [ name =
value ... ] ]
```

This Session Manager layer method is used by the host to provide its communication properties to the TPer, and to retrieve the communication properties of the TPer.

A list of name/value pairs may be provided as the optional HostProperties argument when invoking the Properties method.

If the method is successfully invoked the response is a list of property names and values from the TPer.

The TPer shall return all property name/value pairs for capabilities that it supports. For capabilities not supported by the TPer (for instance, Read-Only sessions), the associated property name/value pair (in this case, MaxReadSessions) shall be omitted from the TPer's response.

The TPer may also respond with additional name/value pairs other than those specified in this document.

The order of the name/value pairs returned by the TPer is not specified.

For the name/value pairs returned by the TPer, the TPer shall return values for the associated names as described in Table 01 or in the associated SSC (the values in the SSC have precedence). The values returned shall apply to all sessions started with the currently associated ComID.

All properties that the TPer returns are stored in the Properties table in the Admin SP. The format of this table is defined in the Admin Template section of this specification (section **Error! Reference source not found.**). All of the property name/value pairs, both specified and implementation-defined, shall be stored in the Properties table. The purpose of the Properties table is not discovery – it is a mechanism to facilitate SP backup.

If the method is invoked with the optional HostProperties parameter, the list of name/value pairs that the TPer shall recognize is:

- **MaxSubpackets**
- **MaxPacketSize**
- **MaxPackets**
- **MaxComPacketSize**
- **MaxResponseComPacketSize**
- **MaxIndTokenSize**
- **MaxAggTokenSize**

These parameters are used to describe the communications capabilities that the host possesses, and apply to any sessions started using the ComID associated with this Properties method invocation once the TPer has processed the request

These values may be submitted in any order by the host. Not all values are required to be submitted. Subsequent submission of these values (in a subsequent invocation of the Properties method) shall supersede values submitted to previous invocations of the Properties method for that ComID. Submitted values, if applicable, shall only apply to sessions started after the submission of those values, and not to sessions that are already open on that ComID.

The TPer may use these host properties when it is constructing responses to be transmitted to the host. The host may omit properties as necessary, depending on the host's communications capabilities. If the host omits a property, or specifies a value for a property that does not meet the minimum requirement as

defined in Table 01, then the TPer shall use the minimum value defined in Table 01 in place of the value supplied by the host.

If the host includes the HostProperties parameter to the Properties method invocation, then the TPer's response shall include all communication property value settings, including those it will use during any subsequently started sessions (both for its communications and the host's). These values reflect the cumulative modifications of all processed Properties methods for the associated ComID.

If a host includes property parameters to the Properties method invocation that the TPer does not recognize or comprehend, the TPer shall ignore those parameters, and shall not return them in its response.

Because of the session-less nature of the Session Manager protocol layer, and the possible different ordering of responses to Session Manager layer methods, the response to this method is formatted as a Properties method invocation so as to be identifiable as the response to the Properties method.

Informative Note: It is the host's responsibility to insure that Properties method invocations have processed prior to invocation of any session startup methods that rely on those invocations. Values for HostProperties at session startup rely on the Properties method invocations that have been processed by the TPer.

**Table 01 Properties Method Response**

| Property | Type | Description |
|---|---|---|
| **MaxMethods** | uinteger | Identifies the maximum number of methods the TPer shall accept in a single subpacket. A value of 0 indicates no limit. |
| **MaxSubpackets** | uinteger | Identifies the maximum number of subpackets that the communicator shall accept in a single Packet. A value of 0 indicates no limit (both a TPer Property and a Host Property). |
| **MaxPacketSize** | uinteger | The maximum size of a packet (including both data and header), in bytes, that the communicator is able to receive. This value shall be at least 512-ComPacketHeader overhead. A value of 0 indicates no limit (both a TPer Property and a Host Property). |
| **MaxPackets** | uinteger | Identifies the maximum number of packets that the communicator shall accept in a single ComPacket. A value of 0 indicates no limit (both a TPer Property and a Host Property). |
| **MaxComPacketSize** | uinteger | The maximum size of an IF Command payload (includes both the ComPacket header and payload) that the communicator is able to receive. This value shall be at least 512. A value of 0 indicates no limit (both a TPer Property and a Host Property). |
| **MaxResponseComPacketSize** | uinteger | The maximum length of an IF Command payload that the communicator is able to generate. A value of 0 indicates no limit (both a TPer Property and a Host Property). |
| **MaxSessions** | uinteger | The maximum number of simultaneous sessions supported by the TPer. A value of 0 indicates no limit. |

| | | |
|---|---|---|
| **MaxReadSessions** | uinteger | The maximum number of simultaneous Read-Only sessions to any one SP supported by the TPer. A value of 0 indicates no limit. |
| **MaxIndTokenSize** | uinteger | The maximum size of a token (in bytes) in a single subpacket that the communicator is able to accept. Token size refers to both the token header and data. This value shall be at least 256. A value of 0 indicates no limit (both a TPer Property and a Host Property). |
| **MaxAggTokenSize** | uinteger | The maximum aggregate size of a continued token after all individual parts of that token are combined that the communicator is able to accept. Token size refers to both the token header and data. This value shall be at least 256. A value of 0 indicates no limit (both a TPer Property and a Host Property). |
| **MaxAuthentications** | uinteger | The maximum number of simultaneously authenticated individual authorities per session that the TPer is able to support. A value of 0 indicates no limit. |
| **MaxTransactionLimit** | uinteger | The maximum number of concurrently open transactions that the TPer is able to support in a single session. A value of 0 indicates no limit. |
| **DefSessionTimeout** | uinteger | The session timeout length (in milliseconds) used by the TPer by default. A value of 0 indicates no limit. |
| **MaxSessionTimeout** | uinteger | The longest supported session timeout length (in milliseconds) supported by the TPer. A value of 0 indicates no limit. |
| **MinSessionTimeout** | uinteger | The shortest supported session timeout length (in milliseconds) supported by the TPer. A value of 0 indicates no limit. |
| **DefTransTimeout** | uinteger | The transmission timeout length (in milliseconds) used by the TPer by default. A value of 0 indicates no limit. |
| **MaxTransTimeout** | uinteger | The longest transmission timeout length (in milliseconds) permitted by the TPer. A value of 0 indicates no limit. |
| **MinTransTimeout** | uinteger | The shortest transmission timeout length (in milliseconds) permitted by the TPer. A value of 0 indicates no limit. |

| | | |
|---|---|---|
| **MaxComIDTime** | uinteger | The timeout length (in milliseconds) used by the TPer after it has assigned a ComID. A session using the associated ComID shall be started within this interval or the ComID shall transition from Issued to Inactive. A value of 0 indicates no limit. |
| **MaxComIDCMD** | uinteger | SSC-dependent limit on the number of interface commands that may be issued using a specific ComID, including both IF-SEND and IF-RECV commands. A value of 0 indicates no limit. |

*End Proposal*

10.2.21 Inactive or Unsupported ComID in CDB Proposal

*Start Proposal*

# Inactive or Unsupported ComID in CDB Proposal

Authors: Darren Lasko (Fujitsu)
Revision
v0.1: 04/11/2008 start of the document
v0.2: 05/01/2008 incorporated feedback:

- Reconsidered how to handle “reserved” ComIDs
- Added reserved values for the 2 least significant bytes of Extended ComIDs

v0.3: 05/08/2008 incorporated feedback from the 05/02/2008 SWG teleconference:

- Separated the conditions for TPers that support dynamic ComID allocation and those that do not
- Reference the “Other Invalid CDB parameter” in SIIF instead of spelling out the behavior in this proposal
- Indicated the bit positions of the ExtendedComID field instead of “least significant two bytes”
- Only reserve the value of FFFFh for bits 15 through 0 of the ExtendedComID field instead of FF00h - FFFFh

v0.4: 05/15/2008 clarified “unsupported, reserved ComID”

# 1 Goal

The goal of this proposal is to specify a mechanism for the TPer to report to the host that the host is using an inactive or unsupported ComID.

# 2 Background

The Core Specification currently does not specify TPer behavior when the host issues an IF-SEND or IF-RECV command with an invactive or unsupported ComID in the CDB (SP_Specific field). This proposal specifies the behavior.

# 3 Proposal

## 3.1 IF-SEND to inactive or unsupported reserved ComID

If the host sends an IF-SEND command to the TPer with a ComID value in the non-reserved range (1000h – FFFFh), and the ComID is in the Inactive state:

- If the TPer supports dynamic ComID allocation, the TPer SHALL:
    - Accept all data in the payload of the IF-SEND command and complete the command normally with good status (provided there are no other errors which would cause the command to abort at the interface level)
    - Ignore and discard the entire payload of the IF-SEND command.
- If the TPer does not support dynamic ComID allocation, the TPer SHALL:
    - Report “Other Invalid CDB parameter” (as specified in the SIIF document)

    **OR**

    - Perform the action described above for TPers that support dynamic ComID allocation

If the host sends an IF-SEND command to the TPer with a ComID value in the reserved range (0000h – 0FFFh), and the ComID is not supported by the TPer, the TPer SHALL:

- Report “Other Invalid CDB parameter” (as specified in the SIIF document)

## 3.2 IF-RECV to inactive or unsupported reserved ComID

If the host sends an IF-RECV command to the TPer with a ComID value in the non-reserved range (1000h – FFFFh), and the ComID is in the Inactive state:

- If the TPer supports dynamic ComID allocation, the TPer SHALL:
    - Respond to the IF-RECV with a zero-length ComPacket (a ComPacket header only) in the IF-RECV payload. The fields in the ComPacket header SHALL contain:
        - **ExtendedComID =** {<ComID from SP_Specific field of CDB>, FFFFh}
            - Note: The value of FFFFh in bits 15 through 0 of the **ExtendedComID** field is an indication to the host that the ComID it is attempting to use is inactive, and that it should not expect to receive any data on that ComID.

- **OutstandingData =** 00000000h
- **MinTransfer =** 00000000h
- **Length =** 00000000h

- Complete the command normally with good status (provided there are no other errors which would cause the command to abort at the interface level)

• If the TPer does not support dynamic ComID allocation, the TPer SHALL:

- Report “Other Invalid CDB parameter” (as specified in the SIIF document)

**OR**

- Perform the action described above for TPers that support dynamic ComID allocation

If the host sends an IF-RECV command to the TPer with a ComID value in the reserved range (0000h – 0FFFh), and the ComID is not supported by the TPer, the TPer SHALL:

• Report “Other Invalid CDB parameter” (as specified in the SIIF document)

## 3.3 Reserved values for Extended ComIDs (to be added to section 3.3.3.1 and/or section 3.3.5.3.1 of the Core Spec)

The value of FFFFh in bits 15 through 0 of the ExtendedComID field is reserved to indicate that the host has attempted to communicate using an inactive ComID.

When assigning Extended ComIDs via the **GET_COMID** command on Protocol ID 02h, the TPer SHALL NOT assign the value of FFFFh in bits 15 through 0 of the ExtendedComID field.

*End Proposal*

# 11 Appendix –ParamCheck examples - Informative

## 11.1 Set Method Example

Using the LRC to protect a Set operation on a PIN

```
MSID_UID.Set[ Values = [ [("PIN", "ThisIsMyPin")] ], ParamCheck = 0x2851]
```

Where ParamCheck is obtained by:

List of Values = [ "ThisIsMyPin" ]
ParamCheck = LRC( Pad("ThisIsMyPin") )
= LRC (0x00 0x54 0x68 0x69 0x73 0x49 0x73 0x4D 0x79 0x50 0x69 0x6E)
= 0x2851

## 11.2 Get Method Example

Using the LRC when retrieving a PIN

```
MSID_UID.Get[[("startCol","PIN"), ("endCol","PIN")], ParamCheck = 1]
   =>
[
   [ [("PIN", "ThisIsMyPin")] ],
   (ParamCheck, 0x2851)
]
```

Where the LRC value is calculated using the same procedure as in "Using the LRC to protect a Set operation on a PIN".