<Product Area>

<Device>

API Specification for

System Service Layer

Draft Version 0.1

April 2015

# Table of Contents

[1 Introduction 5](#_Toc417918765)

[1.2 Purpose of the Document 5](#_Toc417918766)

[1.3 Scope 5](#_Toc417918767)

[1.4 Abbreviations 5](#_Toc417918768)

[1.5 References 6](#_Toc417918769)

[1.6 Assumptions 6](#_Toc417918770)

[2 Public Data Structure 6](#_Toc417918771)

[2.1 SSL\_STATUS 6](#_Toc417918772)

[2.2 SSL\_DBG\_LVL 7](#_Toc417918773)

[2.3 SSL\_WAIT\_TYPE 7](#_Toc417918774)

[3 API 7](#_Toc417918775)

[3.1 SSL\_STATUS SSL\_init(void) 7](#_Toc417918776)

[3.2 SSL\_STATUS SSL\_term(void) 8](#_Toc417918777)

[3.3 SSL\_STATUS SSL\_lock\_create(SSL\_LOCK\_HANDLE \*lock\_id, const uint8\_t \*name, void \*option) 8](#_Toc417918778)

[3.4 SSL\_STATUS SSL\_lock\_delete(SSL\_LOCK\_HANDLE lock\_id) 8](#_Toc417918779)

[3.5 SSL\_STATUS SSL\_lock(SSL\_LOCK\_HANDLE handle, SSL\_WAIT\_TYPE wait\_type) 9](#_Toc417918780)

[3.6 SSL\_STATUS SSL\_unlock(SSL\_LOCK\_HANDLE lock\_id) 9](#_Toc417918781)

[3.7 SSL\_STATUS SSL\_lock\_get\_id\_by\_name(const uint8\_t \*lock\_name, SSL\_LOCK\_HANDLE \*lock\_id) 10](#_Toc417918782)

[3.8 SSL\_STATUS SSL\_print\_set\_lvl(SSL\_DBG\_LVL level); 10](#_Toc417918783)

[3.9 void SSL\_print(SSL\_DBG\_LVL level, char \*msg,…); 11](#_Toc417918784)

[3.10 SSL\_STATUS SSL\_memset(void \*dst, int32\_t val,size\_t size) 11](#_Toc417918785)

[3.11 SSL\_STATUS SSL\_memcpy(void \*dst, void \*src, size\_t size) 11](#_Toc417918786)

[3.12 SSL\_STATUS SSL\_port\_open(SSL\_PORT\_HANDLE \*handle, void \*devcfg) 12](#_Toc417918787)

[3.13 SSL\_STATUS SSL\_port\_close(SSL\_PORT\_HANDLE handle) 13](#_Toc417918788)

[3.14 SSL\_STATUS SSL\_port\_read(SSL\_PORT\_HANDLE handle, void \*dst, uint32\_t \*n) 13](#_Toc417918789)

[3.15 SSL\_STATUS SSL\_port\_write(SSL\_PORT\_HANDLE handle, void \*src, uint32\_t \*n) 14](#_Toc417918790)

[3.16 SSL\_STATUS SSL\_delay(u32 t) 14](#_Toc417918791)

[3.17 SSL\_STATUS SSL\_port\_rw(SSL\_PORT\_HANDLE handle, SSL\_PORT\_ACCESS \*port) 14](#_Toc417918792)

# List of Tables

[Table 1 Abbreviations used in this document 5](#_Toc415653919)

# List of Figures

**No table of figures entries found.**

|  |  |
| --- | --- |
| **Document Owner** | Shally Verma |
| **Version** | Draft |
| **Date** | 1 April 2015 |

| Document Change History | | | |
| --- | --- | --- | --- |
| Date | Version | Changed by | Change Description |
| 27 April 2015 | 0.1 | Shally Verma | Prototype change to use void \* or tyedef data structure than u32. Also all references to u32 change to uint32\_t types. Removed reference of optional functions. Added one more function for combined read / write transaction on ssl port |
|  |  |  |  |
|  |  |  |  |

# Introduction

This document outlines the API set of System Service Layer for development of generic software modules for Microsemi Voice Processor devices.

This library has been leveraged and thin down version Open Source Software Project OSAL. Reason to consider OSAL as a reference for SSL specification is OSAL has already been evaluated on various real time operating systems and we have evaluation of its compatibility for linux kernel. This ensures the API prototyping we choose will be evaluated enough to be considered on various typically used OS and Non-OS based systems.

Though for the purpose of current work, SSL is supposed to be used only by HBI driver but API has been designed keep in mind more generic use of SSL driver. Thus, there are some parameters which may not be used by HBI but can be of use by some other layers or modules.



## Purpose of the Document

The purpose of the document is to provide ‘Application Programming Interface’ specification for System Service Layer which will provide an abstraction to user from base Operating System and Hardware layer. Intention is to provide platform independence to HBI driver and layer above so as to make them portable with minimal amount of time and effort.

## Scope

Scope of this document will only be limited to prototype declaration of APIs and expected return types. Definition or implementation is up to developer porting them on host environment.

Please note this document act as a reference to SSL developer with guidelines on minimal requirement expected to be met by each API.

## Abbreviations

| Table 1 Abbreviations used in this document | |
| --- | --- |
| Abbreviation | Explanation |
| SSL | System Service Layer |
| OSAL | Operating System Abstraction Layer |
| HAL | Hardware Abstraction Layer |

## References

[1] Operating System Abstraction Layer http://osal.sourceforge.net/

## Assumptions

This document assumes that user is aware to different operating system services and their usage.it will not explain and define any OS primitive and related context.

# Public Data Structure

## SSL\_STATUS

This section only gives Status description. Actual definition is up to SSL developer.

The status list enumerates most common listed error types. This list can be extended by user according to their requirement and platform specific error reporting.

|  |  |
| --- | --- |
| Status | Description |
| SSL\_STATUS\_NOT\_INIT | This indicates the SSL\_init() not called |
| SSL\_STATUS\_INTERNAL\_ERR | This is generic error code to indicate that an error has been returned from base host SDK function |
| SSL\_STATUS\_INVALID\_PARAM | This indicates some or all of the input parameters are invalid. Not allocated or out of range |
| SSL\_STATUS\_BAD\_HANDLE | This indicated ID passed to the call is invalid i.e. not in list or unallocated. |
| SSL\_STATUS\_RESOURCE\_ERR | This indicates if API fails to get any resource either memory or any other |
| SSL\_STATUS\_OK | This indicates call is successful |
| SSL\_STATUS\_TIMEOUT | This indicates SSL\_lock() timed out waiting to acquire a lock |
| SSL\_STATUS\_FAILED | Call failed |
| SSL\_STATUS\_OP\_INCOMPLETE | This indicates if port read/write transaction remain incomplete i.e. number of bytes actually read or written is less than requested |

## SSL\_DBG\_LVL

|  |  |
| --- | --- |
| Status | Description |
| SSL\_DBG\_LVL\_ALL | Print all of the messages regardless of level. |
| SSL\_DBG\_LVL\_INFO | Prints messages marked as informational |
| SSL\_DBG\_LVL\_FUNC | Prints Entry and Exit of function calls |
| SSL\_DBG\_LVL\_WARN | Prints messages marked with warning |
| SSL\_DBG\_LVL\_ERR | Prints messages marked as error |
| SSL\_DBG\_LVL\_NONE | No print message will be output to console |

## SSL\_WAIT\_TYPE

|  |  |
| --- | --- |
| Enum | Desciption |
| SSL\_WAIT\_NONE | Call returns immediately if failed to get lock |
| SSL\_WAIT\_FOREVER | Call would block until it get requested resource |

# API

A SSL will be providing an API for following feature set (Reference: Generic SW Architecture Document):

* Locking and unlocking (Mutex or Semaphore is dependent on user)
* Memory clear and copy operations
* Print APIs to print console messages on stdio
* Hardware port init, open, term and close and bytewide address read and write

## SSL\_STATUS SSL\_init(void)

**Description**:

This function should setup basic infrastructure necessary to communicate host to target device. For example, initialization of local data structure and resources as consumed by SSL. Hardware bus one time initialization is supposed to be done here.

If not called, then call to some of the SSL function should fail with error SSL\_STATUS\_NOT\_INIT

**Return Code**:

SSL\_STATUS\_INTERNAL\_ERR

SSL\_STATUS\_RESOURCE\_ERR

SSL\_STATUS\_OK

**Input**:

None

**Output**:

None

## SSL\_STATUS SSL\_term(void)

**Description**:

This function should release all of the resources as acquired or setup during call to SSL\_init() function. Basically it should reverse whatever been done during initialization call. After a call to this function, call to port access functions should return error SSL\_STATUS\_NOT\_INIT

**Return Code**:

SSL\_STATUS\_OK

SSL\_STATUS\_FAILED

SSL\_STATUS\_INTERNAL\_ERR

**Input**:

None

**Output**:

None

## SSL\_STATUS SSL\_lock\_create(SSL\_LOCK\_HANDLE \*lock\_id, const uint8\_t \*name, void \*option)

**Description**:

This function will create a lock for synchronization purpose. User can implement it as binary or counting semaphore or simple mutex. Any information required to create semaphore or mutex or critical section lock can be passed into a data structure pointed by ***void \****parameter.

**Return Code**:

SSL\_STATUS\_RESOURCE\_ERR

SSL\_STATUS\_INVALID\_PARAM

SSL\_STATUS\_INTERNAL\_ERR

SSL\_STATUS\_OK

**Input**:

|  |  |
| --- | --- |
| Parameter | Description |
| const u8 \*name | Name of lock.optional can be NULL |
| void \*options | This is pointer to specific data structure as may be relevant to base operating system or lock type being created by user. User can define it to anything as per their need thus it is passed as void \*. |
| u32 \* lock\_id | An unsigned long pointer to store lock id value |

**Output**:

|  |  |
| --- | --- |
| Parameter | Description |
| U32 pointer | This value at this pointer should be updated by create call at the successful execution. This variable is act as reference handle for subsequent call to other lock APIs. |

## SSL\_STATUS SSL\_lock\_delete(SSL\_LOCK\_HANDLE lock\_id)

**Description:**

This function should destroy a lock as created by call to SSL\_lock\_create(). Any further reference or usage of this lock should return SSL\_STATUS\_BAD\_HANDLE after the successful execution of this function.

**Return Code**:

SSL\_STATUS\_BAD\_HANDLE

SSL\_STATUS\_INTERNAL\_ERROR

SSL\_STATUS\_OK

**Input**:

|  |  |
| --- | --- |
| Parameter | Description |
| SSL\_LOCK\_HANDLE | Lock ID as returned by successful call to SSL\_lock\_create() |

**Output**:

None

## SSL\_STATUS SSL\_lock(SSL\_LOCK\_HANDLE handle, SSL\_WAIT\_TYPE wait\_type)

**Description**:

This function should acquire a lock whenever called. If already locked, this call would behave according to wait\_type parameter. If not equal to SSL\_WAIT\_NONE and SSL\_WAIT\_FOREVER , then call is assumed to be passed as timeout value in milliseconds. The implementation of wait\_type parameter is optional and dependent upon lock type being implemented.

**Return Codes:**

SSL\_STATUS\_OK

SSL\_STATUS\_INTERNAL\_ERR

SSL\_STATUS\_BAD\_HANDLE

SSL\_STATUS\_TIMEOUT

SSL\_STATUS\_FAILED

**Input**:

|  |  |
| --- | --- |
| Parameter | Description |
| SSL\_LOCK\_HANDLE | lock ID as returned by successful call to SSL\_lock\_create() |
| SSL\_WAIT\_TYPE | integer value indicating wait period (wait\_none, wait\_forever, timeout in usec) |

**Output**:

None

## SSL\_STATUS SSL\_unlock(SSL\_LOCK\_HANDLE lock\_id)

**Description**:

This function releases the lock acquired.

**Return Code:**

SSL\_STATUS\_OK

SSL\_STATUS\_INTERNAL\_ERR

SSL\_STATUS\_BAD\_HANDLE

**Input**:

|  |  |
| --- | --- |
| Parameter | Description |
| SSL\_LOCK\_HANDLE | lock ID as returned by successful call to SSL\_lock\_create() |

**Output**:  
None

## SSL\_STATUS SSL\_lock\_get\_id\_by\_name(const uint8\_t \*lock\_name, SSL\_LOCK\_HANDLE \*lock\_id)

**Description**:

This function should return lock id corresponding to given name.

If any lock isn’t created with respective name at SSL layer, then function will return SSL\_STATUS\_FAILED. If any of the input parameter is invalid or a NULL pointer, then function should return SSL\_STATUS\_INVALID\_PARAM. Function may return SSL\_STATUS\_INTERNAL\_ERR if using host SDK function which failed.

Implementation of this function is optional and purely on user descrition.

**Return Code:**

SSL\_STATUS\_INVALID\_PARAM

SSL\_STATUS\_INTERNAL\_ERR

SSL\_STATUS\_OK

SSL\_STATUS\_FAILED

**Input**:

|  |  |
| --- | --- |
| Parameter | Description |
| const uint8\_t \*lock\_name | Valid null terminated string pointer containing name of the lock |
| SSL\_LOCK\_HANDLE \*lock\_id | A valid allocated pointer |

**Output**:

|  |  |
| --- | --- |
| Parameter | Description |
| SSL\_LOCK\_HANDLE \* | An pointer as updated with respective lock id value by function on successful execution |

## SSL\_STATUS SSL\_print\_set\_lvl(SSL\_DBG\_LVL level);

**Description**:

This call would set debug level as described in SSL\_DBG\_LVL. It is expected messages marked with specified level will be printed.

**Input**:

|  |  |
| --- | --- |
| Parameter | Description |
| SSL\_DBG\_LVL | Enum indicating debug level |

**Output**:

None

**Return Code:**

SSL\_STATUS\_OK

SSL\_STATUS\_INVALID\_PARAM

## void SSL\_print(SSL\_DBG\_LVL level, char \*msg,…);

**Description**:

This will print the messages with given level on stdio. A level should be preset by making a call to SSL\_print\_set\_lvl().

**Return Code:**

None

**Input**:

|  |  |
| --- | --- |
| Parameter | Description |
| Level | Message Debug Level |
| uint8\_t \*msg | pointer to string to be printed |
| … | Ellipsis for variable arguments |

**Output:**

None

## SSL\_STATUS SSL\_memset(void \*dst, int32\_t val,size\_t size)

**Description**:

This function will set memory area beginning with “src” pointer upto length as indicated by “size” parameter to a specified value in “val” parameter.

This function doesn’t need initialization (subject to review)

**Return code:**

SSL\_STATUS\_OK

SSL\_STATUS\_INVALID\_PARAM

SSL\_STATUS\_INTERNAL\_ERR

**Input:**

|  |  |
| --- | --- |
| Parameter | Description |
| void \*dst | A destination pointer to be written to |
| Int32\_t value | Integer value to be written to |
| size\_t | Number of bytes to be written with specified value |

**Output:**

None

## SSL\_STATUS SSL\_memcpy(void \*dst, void \*src, size\_t size)

**Description:**

This function will copy data from one memory located as pointed by “src” parameter to another location as pointed by “dst” parameter of the size given by “size” parameter.

**Return Code:**

SSL\_STATUS\_OK

SSL\_STATUS\_INVALID\_PARAM

SSL\_STATUS\_INTERNAL\_ERR

**Input:**

|  |  |
| --- | --- |
| Parameter | Description |
| void \*src | A source pointer to copy data from |
| void \*dst | Destination pointer to copy data to |
| size | Number of bytes to copy |

**Output:**

None

## SSL\_STATUS SSL\_port\_open(SSL\_PORT\_HANDLE \*handle, void \*devcfg)

**Description:**

This function should open a communication port to device. It can be setting up a bus or gpio’s configuration or interfacing to host controller driver as needed according to host platform hardware and software level interface to target device (ex. timberwolf).

A call to this function should return a reference handle to caller which should further be used a reference handle to identify device / bus. This is particularly needed for a system with multiple devices and host controller.

**Return Code:**

SSL\_STATUS\_OK

SSL\_STATUS\_NOT\_INIT

SSL\_STATUS\_RESOURCE\_ERR

SSL\_STATUS\_INTERNAL\_ERR

SSL\_STATUS\_INVALID\_PARAM

**Input**

|  |  |
| --- | --- |
| Parameter | Description |
| SSL\_PORT\_HANDLE \*handle | An unsigned long pointer |
| void \*devcfg | Any user defined configuration that user may need to pass while opening a communication port to device. This is specific to user and optional. User may or may not implement. For our case, it will point to SSL\_DEV\_CFG |

**Output**:

|  |  |
| --- | --- |
| Parameter | Description |
| SSL\_PORT\_HANDLE \*handle | A pointer to be updated with reference handle to a hardware port |

## SSL\_STATUS SSL\_port\_close(SSL\_PORT\_HANDLE handle)

**Description**:

This call simply closes the communication port as opened by a call to SSL\_port\_open()

**Return Code:**

SSL\_STATUS\_OK

SSL\_STATUS\_INTERNAL\_ERR

SSL\_STATUS\_NOT\_INIT

SSL\_STATUS\_INVALID\_PARAM

**Input:**

|  |  |
| --- | --- |
| Parameter | Description |
| u32 handle | Handle to hardware port to be closed |

**Output:**

None

## SSL\_STATUS SSL\_port\_read(SSL\_PORT\_HANDLE handle, void \*dst, uint32\_t \*n)

**Description:**

This function will read number of bytes from port in to a user buffer.

Number of bytes actually been read should be updated back to pointer “n”.

**Return Code:**

SSL\_STATUS\_OK

SSL\_STATUS\_NOT\_INIT

SSL\_STATUS\_INVALID\_PARAM

SSL\_STATUS\_BAD\_HANDLE

SSL\_STATUS\_INTERNAL\_ERR

SSL\_STATUS\_RESOURCE\_ERR

SSL\_STATUS\_OP\_INCOMPLETE

Input:

|  |  |
| --- | --- |
| Parameter | Description |
| Handle | Handle to hardware port to be read |
| \*dst | Destination buffer to read data to |
| \*n | Pointer indicating Number of bytes to read from port |

Output

|  |  |
| --- | --- |
| Parameter | Description |
| n | Pointer updated with Number of bytes actually read from port |

## SSL\_STATUS SSL\_port\_write(SSL\_PORT\_HANDLE handle, void \*src, uint32\_t \*n)

**Description:**

This function should write number of bytes from user buffer to communication port.

**Return Code:**

SSL\_STATUS\_OK

SSL\_STATUS\_INTERNAL\_ERR

SSL\_STATUS\_NOT\_INIT

SSL\_STATUS\_INVALID\_PARAM

SSL\_STATUS\_BAD\_HANDLE

SSL\_STATUS\_RESOURCE\_ERR

SSL\_STATUS\_OP\_INCOMPLETE

**Input:**

|  |  |
| --- | --- |
| Parameter | Description |
| Handle | Handle to hardware port |
| \*src | source buffer to be written |
| \*n | Pointer indicating Number of bytes to write from buffer to port |

**Output:**

|  |  |
| --- | --- |
| Parameter | Description |
| \*n | Pointer updated with Number of bytes actually written to port |







## SSL\_STATUS SSL\_delay(u32 t)

**Description:**

This function delays the execution of the current call for the time period indicated by “t” in milliseconds.

**Return Code:**

SSL\_STATUS\_OK

## SSL\_STATUS SSL\_port\_rw(SSL\_PORT\_HANDLE handle, SSL\_PORT\_ACCESS \*port)

**Description:**

This function does a combined read write transaction over a port. It first writes data as given in src buffer and then reads back the response in destination buffer pointer. It is useful where user want to read a register or device memory in one go.

**Return Code:**

SSL\_STATUS\_OK

SSL\_STATUS\_INVALID\_PARAM

SSL\_STATUS\_INTERNAL\_ERR

SSL\_STATUS\_NOT\_INIT

SSL\_STATUS\_BAD\_HANDLE

SSL\_STATUS\_OP\_INCOMPLETE

Input

|  |  |
| --- | --- |
| Parameter | Description |
| Handle | Handle to hardware port |
| SSL\_PORT\_ACCESS \* | Pointer to a structure containing port access operation |

Output:

This function may opt to output a number of bytes actually read or written