Common Security Services Manager

Data Storage Library Interface (DLI) Specification

Release 1.2
March 1997

Subject to Change Without Notice

Specification Disclaimer and Limited Use License

This specification is for release version 1.2, March 1997.

Table of Contents

List of Figures

1. Introduction

1.1 CDSA Overview

The Common Data Security Architecture (CDSA) defines the infrastructure for a complete set of security services. CDSA is an extensible architecture that provides mechanisms to manage add-in security modules, which use cryptography as a computational base to build secure protocols and secure systems. Figure 1 shows the four basic layers of the Common Data Security Architecture: Applications, System Security Services, the Common Security Services Manager, and Security Add-in Modules. The Common Security Services Manager (CSSM) is the core of CDSA. It enables applications to directly access security services through the CSSM security API, or to indirectly access them via layered security services and tools implemented over the CSSM API. CSSM manages the add-in security modules and directs application calls through the CSSM API to the selected add-in module servicing the request. Add-in modules perform various aspects of security services, including:

- Cryptographic Services
- Trust Policy Services
- Certificate Library Services
- Data Storage Library Services

Cryptographic Service Providers (CSPs) are add-in modules which perform cryptographic operations including encryption, decryption, digital signaturing, key pair generation, random number generation, and key exchange. Trust Policy (TP) modules implement policies defined by authorities and institutions, such as VeriSign* (as a certificate authority) or MasterCard* (as an institution). Each trust policy module embodies the semantics of a trust model based on using digital certificates as credentials. Applications may use a digital certificate as an identity credential and/or an authorization credential. Certificate Library (CL) modules provide format-specific, syntactic manipulation of memory-resident digital certificates and certificate revocation lists. Data Storage library (DL) modules provide persistent storage for certificates and certificate revocation lists.

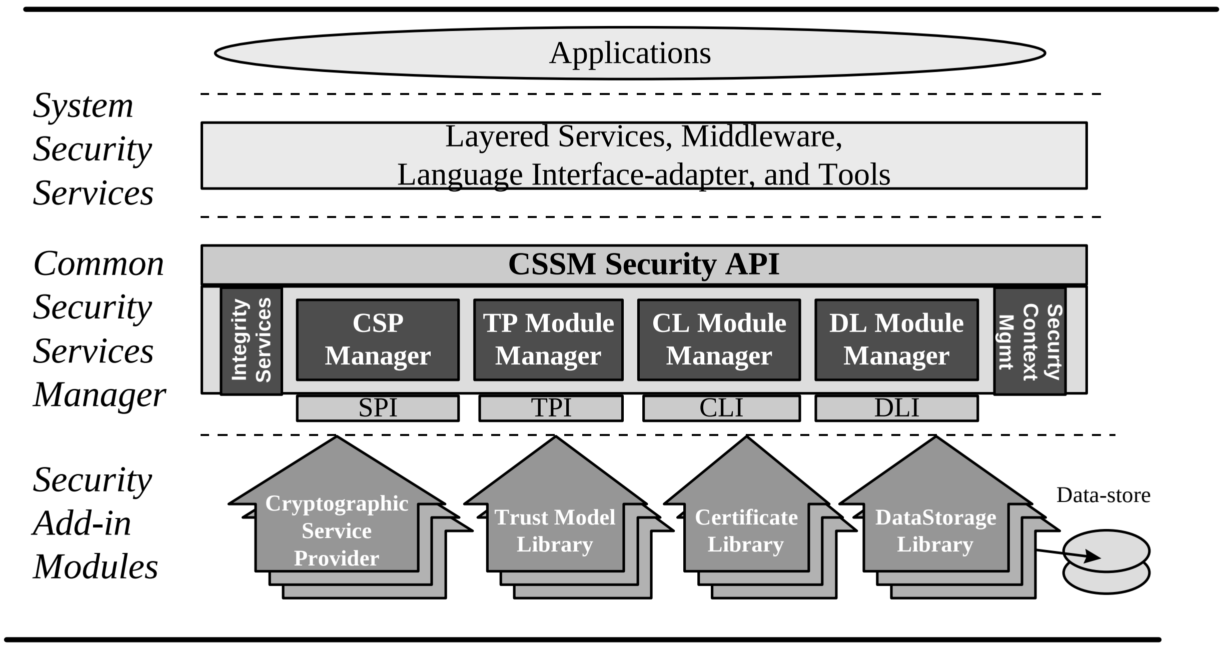

Figure 1. The Common Data Security Architecture for all platforms

Applications directly or indirectly select the modules used to provide security services to the application. These add-in modules are provided by independent software and hardware vendors. The functionality of the add-in module may be extended beyond the services defined by the CSSM API by exporting additional services to applications via the CSSM pass through mechanism.

The API calls defined for add-in modules are categorized as service operations, module management operations, and module-specific operations. Service operations include functions that perform a security operation such as encrypting data, inserting a certificate revocation list into a data store, or verifying that a certificate is trusted. Module management functions support module installation, registration of module features and attributes, and queries to retrieve information on module availability and features. Module-specific operations are enabled in the API through pass-through functions whose behavior and use is defined by the add-in module developer.

CSSM also provides integrity services and security context management. CSSM applies the integrity check facility to itself to ensure that the currently-executing instance of CSSM code has not been tampered with.

Security context management provides secured runtime caching of user-specific state information and secrets. The manager focuses on caching state information and parameters for performing cryptographic operations. Examples of secrets that must be cached during application execution include the application’s private key and the application’s digital certificate.

In summary, the CSSM provides these services through its API calls:

- Certificate-based services and operations
- Comprehensive, extensible service provider interfaces (SPI) for cryptographic service provider modules, trust policy modules, certificate library modules, and data storage modules
- Registration and management of available cryptographic service provider modules, trust policy modules, certificate library modules, and data storage modules
- Caching of keys and secrets required as part of the runtime context of a user application
- Call-back functions for disk, screen, and keyboard I/O supported by the operating system
- A test-and-check function to ensure CSSM integrity
- Management of concurrent security operations

1.2 Data Storage Library Overview

A data storage library (DL) module provides persistent storage for certificates, certificate revocation lists (CRLs) and generic objects. Stable storage can be provided by a

- commercially-available database management system product
- native file system
- custom hardware-based storage device

Each DL module may choose to implement only those operations required to provide persistent storage for certificates and CRLs under its selected model of service.

The implementation of DL operations should be semantically free. For example, a DL operation involving X.509 objects should not be responsible for checking whether the certificate values or CRL values conform to the X.509 format. Semantic interpretation of such values should be implemented in TP modules, layered services, and applications.

A pass-through function is defined in the DL API. This mechanism allows each DL to provide additional functions to store and retrieve certificates, CRLs and generic objects. Pass-through functions may be used to increase functionality or enhance performance.

1.2.1 Application Interaction

When a new DL is installed on a system, information specific to a DL is stored in the CSSM registry. An application uses this information to find an appropriate DL and to request that CSSM attach to the DL. When CSSM attaches to a DL, it returns a DL handle to the application that uniquely identifies the pairing of the application thread to a DL module instance. The application uses this handle to identify the DL in future function calls. The DL also uses the handle to identify the calling application.

CSSM passes function calls from an application to a data storage library by making use of a DL’s function table. The function table consists of pointers to the data storage functions from the CSSM APIs, which are supported by a DL. When an application causes CSSM to attach a DL, the data storage library registers its function table with CSSM using *CSSM_DL_RegisterServices*. During future DL function calls from the application, CSSM will use these function pointers to direct calls to the DL appropriately.

The calling application is responsible for the allocation and de-allocation of all memory that is passed into or out of the data storage library module. The application must register memory allocation and de-allocation upcalls with CSSM when it requests a DL attach. These upcalls and the handle identifying the application/DL pairing are passed to a DL when CSSM calls its *DL_Initialize* function. These functions must be used whenever a data storage library allocates or de-allocates memory, which is passed out of or into a DL module.

1.2.2 DL Structure and Use

A data storage library is composed of functions, which are invoked when a DL is attached and detached and ones that mirror the CSSM API for data storage operations. When a DL is attached, it registers its function table with the CSSM. During the attach process, CSSM also calls the *DL_Initialize* function and provides a DL with a set of upcalls for memory management. When a DL is detached, any necessary cleanup actions are performed. The remainder of the data storage library functions perform basic storage and retrieval operations on generic data objects, certificates and certificate revocation lists (CRLs). Extended services may be provided via the DL_PassThrough function. This function passes a DL module-defined operation identifier and parameters to a DL.

Data storage libraries may use other CSSM add-in modules to implement their functionality. For example, a data storage library that builds an index to support fast retrieval may use the capabilities of a CL add-in module library to obtain the individual fields of a certificate or CRL to build the index. More information about CL modules can be found in the *Common Data Security Architecture Specification* and in the *CSSM Certificate Library Interface Specification.*

Similarly, data storage libraries may be used by other CSSM add-in modules to implement their functionality. For example, Trust Policy modules may call the data storage library functions to store certificates and CRLs. More information about the TP module can be found in the *Common Data Security Architecture Specification* and the *CSSM Trust Policy Interface Specification.*

1.3 CSSM Data Storage Library Interface Specification

1.3.1 Intended Audience

This document should be used by Independent Software Vendors (ISVs) who want to develop their own data storage library modules to store and retrieve certificates and/or CRLs. These ISVs should be highly experienced software and security architects, advanced programmers, and sophisticated users. They are familiar with data storage systems, high-end cryptography, and digital certificates. It is assumed that this audience is familiar with the basic capabilities and features of the protocols they are considering.

1.3.2 Document Organization

This document is divided into the following sections.
Section 1, Introduction, gives the CDSA architecture overview and the data storage library module overview.

Section 2, Data Storage Library Interface, describes the functions that a data storage library makes available to applications via the CSSM.

Section 3, Data Storage Library Structure and Management, describes important considerations in developing a data storage library. It also gives examples of how several data storage library functions might be implemented.

1.4 References

PKCS*	*The Public-Key Cryptography Standards*, RSA Laboratories, Redwood City, CA: RSA Data Security, Inc
X.509	*CCITT. Recommendation X.509: The Directory – Authentication Framework*, 1988. CCITT stands for Comite Consultatif Internationale Telegraphique et Telephonique (International Telegraph and Telephone Consultative Committee)
SPKI	*Simple Public Key Infrastructure*
SDSI	*SDSI - A Simple Distributed Security Infrastructure*, R. Rivest and B. Lampson, 1996
CDSA	*Common Data Security Architecture Specification*, Intel Architecture Labs, 1996
CSSM API	*CSSM Application Programming Interface*, Intel Architecture Labs, 1996
CSSM SPI	*CSSM Cryptographic Service Provider Interface Specification*, Intel Architecture Labs, 1996
CSSM TPI	*CSSM Trust Policy Interface Specification*, Intel Architecture Labs, 1996
CSSM CLI	*CSSM Certificate Library Interface Specification*, Intel Architecture Labs, 1996
CSSM Java	*CSSM Java Application Programming Interface (API) Specification*, Intel Architecture Labs, 1996

2. Data Storage Library Interface

2.1 Overview

The primary purpose of a data storage library (DL) module is to provide persistent storage of security-related objects including certificates, certificate revocation lists (CRLs), keys, and policy objects. A DL module is responsible for the creation and accessibility of one or more data stores. A single DL module can be tightly tied to a CL and/or TP module, or can be independent of all other module types. A single data store can contain a single object type in one format, a single object type in multiple formats, or multiple object types. The persistent repository can be local or remote (For example, a DL provides client access to a remote directory/storage service). CSSM stores and manages meta-information about a DL in the CSSM registry. This information describes the storage and retrieval capabilities of a DL. Applications can query the CSSM registry to obtain information about the available DLs and attach to a DL that provides the needed services. Each DL should store meta-information about each of the data stores it manages.

The DL APIs define a data storage model that can be implemented using a custom storage device, a traditional local or remote file system service, a database management system package, or a complete information management system. The abstract data model defined by the DL APIs partitions all attributes stored in a data record into two general categories: immutable attributes and mutable attributes.

The DL APIs also support the notion that a DL may not be able to parse/interpret all attributes of a data object. For example, a DL that stores certificates may not be able to examine fields within those certificates when performing retrieval operations. In such cases, the APIs allow the caller to specify the handle of a certificate library module to be invoked to parse field values from the certificate.

The DL APIs defined in CSSM Release 1.0 included operations that were specific to certificates and CRLs. These functions are made obsolete by new CSSM Release 1.2 APIs to store and retrieve an extensible set of security-related data objects. However the obsolete APIs have been retained for backward compatibility. Developers are encouraged to use the new data object APIs defined in this specification exclusively.

The data storage library services API defines three categories of operations:

- Data-source management functions
- Persistent operations on generic data objects
- Persistence operations on certificates and certificate revocation lists (provided for backward compatibility with CSSM1.0)

The Data-source management functions operate on a data store as a single unit. These operations include:

- Opening and closing data stores. A DL module manages the mapping of logical data store names to the storage mechanisms it uses to provide persistence. The caller uses logical names to reference persistent data stores. The open operation prepares an existing data store for future access by the caller. The close operation terminates current access to the data store by the caller.
- Creating and deleting data stores. A DL module creates a new, empty data store and opens it for future access by the caller. An existing data store may be deleted. Deletion discards all data contained in the data store.

- Importing and exporting data stores. Occasionally a data store must be moved from one system to another or from one storage medium to another. The import and export operations support the transfer of an entire data store. The export operation prepares a snapshot of a data store. (Export does not delete the data store it snapshots.)

  The import operation accepts a snapshot (generated by the export operation) and includes it as a new data store managed by a DL module. The imported data store is assigned a new logical name, which is then known by the local DL module.

A data store may contain certificate objects, certificate revocation objects, policy objects, key objects or other generic data objects.

The persistence operations on data stores include:

- Adding new Data objects. A DL module adds a persistent copy of data object to an open data store. This operation may or may not include the creation of index entries. The mechanisms used to store and retrieve persistent data objects are private to the implementation of a DL module.
- Deleting data objects. A DL module removes single data object from the data store.
- Retrieving data objects. Applications and add-in security modules need to search data stores for data objects. A DL module must provide a search mechanism for selectively retrieving a copy of these persistent objects. Selection is based on a selection criterion.

The following APIs are provided for backward compatibility with CSSM Release 1.0 APIs.

- Adding new certificates and new certificate revocation records. The DL module adds a persistent copy of a certificate or a certificate revocation record to an open data store. This operation may or may not include the creation of index entries. The mechanisms used to store and retrieve persistent certificates and certificate revocation records are private to the implementation of a DL module.
- Deleting certificates and certificate revocation records. A DL module removes a single certificate or a single certificate revocation record from the data store.
- Retrieving certificates and certificate revocation records. Applications and add-in security modules need to search data stores for certificates and certificate revocation records. A DL module must provide a search mechanism for selectively retrieving a copy of these persistent objects. Selection is based on a selection criterion.

Module management functions include:

- Check module version compatibility. A data storage library module must provide support for the DL_Initialize operation. CSSM uses this operation to verify that the attached DL module version is compatible with a DL module version requested by the calling application. It is called as part of *CSSM_DL_Attach*, immediately following the data storage library's registration of its function table. If the versions are incompatible, CSSM will detach a DL and the *CSSM_DL_Attach* operation will fail.

Data store extensibility operations include:

- Pass-through for unique, module-specific operations. A pass-through function is included in the data storage library Interface to allow data store libraries to expose additional services beyond what is currently defined in the CSSM API. CSSM passes an operation identifier and input parameters from the application to the appropriate data storage library. Within the DL_PassThrough function in the data storage library, the input parameters are interpreted and the appropriate operation performed. The data storage library developer is responsible for making known to the application the identity and parameters of the supported pass-through operations.

2.1.1 Data source Operations

CSSM_DB_HANDLE CSSMDLI DL_DbOpen ( ) opens a data store with the specified logical name. Returns a handle to the data store.

CSSM_RETURN CSSMDLI DL_DbClose ( ) closes a data store.

CSSM_ DB_HANDLE CSSMDLI DL_DbCreate ( ) creates a new, empty data store with the specified logical name, and the specified attribute schema.

CSSM_RETURN CSSMDLI DL_DbDelete ( ) deletes all records from the specified data store and removes current state information associated with that data store.

CSSM_RETURN CSSMDLI DL_DbImport ( ) accepts as input a filename and a logical name for a data store. The file is an exported copy of an existing data store. The data records contained in the file must be in the native format of a DL module. A DL module imports all security objects in the file (such as certificates and CRLs), creating a new data record for each. The specified logical name is assigned to the new data store. Note: This mechanism can be used to copy data stores among systems or to restore a persistent data store from a backup copy. It could also be used to import data stores that were created and managed by other DLs, but this is not the typical implementation and use of this interface.

CSSM_RETURN CSSMDLI DL_DbExport ( ) accepts as input the logical name of a data store and the name of a target output file. The specified data store contains persistent data records. A representation of the schema for the data store being exported is written to the file along with a copy of each data record in the data store. Note: This mechanism can be used to copy data stores among systems or to create a backup of persistent data stores.

CSSM_DBINFO_PTR CSSMDLI DL_DbGetInfo ( ) CSSM returns a CSSM_DBINFO structure describing the Datasource meta data like DbType, Dbrecord type etc.

CSSM_RETURN CSSMDLI DL_DbSetInfo ( ) stores the Data source-specific meta data information.

CSSM_RETURN CSSMDLI DL_FreeDbInfo ( ) frees the memory structure returned by the CSSM_DL_DbGetInfo function.

CSSM_NAME_LIST_PTR CSSMDLI DL_GetDbHandleToName() retrieves the data source corresponding to an opened database handle. A DL module is responsible for allocating the memory required for the list.

2.1.2 Generic Data Storage Operations

CSSM_RETURN CSSMDLI DL_DataInsert ( ) accepts as input a data object, the record type of the object, a handle to a data store, and an optional handle to a module that can parse un-crackable attribute fields of .object using GetFirstField operations. The data object is made persistent in the specified data store. This may or may not include the creation of index entries, etc. The mechanisms used to store and retrieve persistent certificates is private to the implementation of the Data Storage Library.

CSSM_RETURN CSSMDLI DL_DataDelete ( ) accepts as input a data object, the record type of the object, and a handle to a data store. The object is removed from the data store. If the object is not found in the specified data store, the operation fails.

CSSM_DATA_PTR CSSMDLI DL_DataGetFirst ( ) accepts as input a selection predicate, the type of data record to be retrieved, and a handle to a data store. The specified data store is searched for data objects of the specified type that match the selection criteria. Selection predicates are represented in one or two forms: a predicate string (For example, a string that is appropriate as a *where* clause in an SQL statement), or as a set of (name, value, relational operator) triples that are connected by a conjunctive operator. The conjunctive operators are Boolean-and and Boolean-or. The relational operators include greater-than, less-than, equal-to, and not-equal-to. This function returns a count of the total number of data objects matching the selection criteria, the first data object matching the criteria, and a selection handle that may be used to retrieve the subsequent objects matched in the search. A data storage library may limit the number of concurrently managed selection handles to exactly one. The library developer must document all such restrictions and application developers should proceed accordingly.

CSSM_DATA_PTR CSSMDLI DL_DataGetNext ( ) accepts as input the type of the record to be returned and a selection results handle that was returned by an invocation of the function CSSM_DL_DataGetFirst ( ). In response, a DL module returns the next data record from the set specified by the selection results handle. If all data records have already been returned from the set specified by the selection handle, then the function returns a NULL data record. A data storage library may limit the number of concurrently-managed selection result handles to exactly one. The library developer must document such restrictions, and application developers should proceed accordingly.

CSSM_RETURN CSSMDLI DL_DataAbortQuery ( ) cancels the query initiated by CSSM_DL_DataGetFirst function and resets the selection results handle.

2.1.3 Certificate Storage Operations - included for backward compatibility with CSSM 1.0

CSSM_RETURN CSSMDLI DL_CertRevoke ( ) accepts as input a certificate to be revoked and a handle to a data store. The knowledge that the certificate has been revoked is made persistent. The representation of this information and the mechanism for creating and managing the *representation of revocation* is private to the implementation of the Data Storage Library.

CSSM_RETURN CSSMDLI DL_CertInsert ( ) accepts as input a certificate, and optional handle to a certificate library module (which may be used to parse the certificate), and a handle to a data store. The certificate is made persistent in the specified data store. This may or may not include the creation of index entries, etc. The mechanisms used to store and retrieve persistent certificates is private to the implementation of the Data Storage Library.

CSSM_RETURN CSSMDLI DL_CertDelete ( ) accepts as input a certificate and a handle to a data store. The certificate is removed from the data store. If the certificate is not found in the specified data store, the operation fails.

CSSM_DATA_PTR CSSMDLI DL_CertGetFirst ( ) accepts as input a set of relational expressions, a single conjunctive operator, and a handle to a data store. The specified data store is searched for certificates that match the selection criteria. The selection criteria is the expression formed by connecting all of the relational expressions using the one conjunctive operator. The conjunctive operators are Boolean-and and Boolean-or. The relational operators include greater-than, less-than, equal-to, and not-equal-to. This function returns a count of the total number of certificates matching the selection criteria, the first certificate matching the criteria, and a selection handle that may be used to retrieve the subsequent certificates matching the selection criteria. A data storage library may limit the number of concurrently-managed selection handles to exactly one. The library developer must document such restrictions and application developers should be aware of such restrictions.

CSSM_DATA_PTR CSSMDLI DL_CertGetNext ( ) accepts as input a selection handle that was returned by an invocation of the function CSSM_DL_CertGetFirst ( ). In response, a DL module returns the next certificate from the set specified by the selection handle. If all certificates have already been returned from the set specified by selection handle, then the function returns a NULL certificate. A data storage library may limit the number of concurrently-managed selection handles to exactly one. The library developer must document such restrictions, and application developers should be aware of such restrictions.

CSSM_RETURN CSSMDLI DL_CertAbortQuery ( ) cancels the query initiated by CSSM_DL_CertGetFirst function and resets the selection handle.

2.1.4 CRL Storage Operations - included for backward compatibility with CSSM 1.0

CSSM_RETURN CSSMDLI DL_CrlInsert ( ) accepts as input a CRL record and the handle of a data store of CRL records. The new record is made persistent in the data store of CRL records. This may or may not include the creation of index entries, etc. The mechanisms used to store and retrieve persistent CRL records is private to the implementation of the data storage library.

CSSM_RETURN CSSMDLI DL_CrlDelete ( ) accepts as input a CRL record and the handle of a data store of CRL records. The single record is removed from the data store. If the record is not found in the specified data store, the operation fails.

CSSM_DATA_PTR CSSMDLI DL_CrlGetFirst ( ) accepts as input a set of relational expressions, a single conjunctive operator, and a handle to a data store. The specified data store is searched for CRL records that match the selection criteria. The selection criteria is the expression formed by connecting all of the relational expressions using the one conjunctive operator. The conjunctive operators are Boolean-and and Boolean-or. The relational operators include greater-than, less-than, equal-to, and not-equal-to. This function returns a count of the total number of CRL records matching the selection criteria, the first CRL record matching the criteria, and a selection handle that may be used to retrieve the subsequent CRL records matching the selection criteria. A data storage library may limit the number of concurrently-managed selection handles to exactly one. The library developer must document such restrictions and application developers should be aware of such restrictions.

CSSM_DATA_PTR CSSMDLI DL_CrlGetNext ( ) accepts as input a selection handle that was returned by an invocation of the function CSSM_DL_CrlGetFirst ( ). In response, a DL module returns the next CRL record from the set specified by the selection handle. If all CRL records have already been returned from the set specified by selection handle, then the function returns a NULL CRL pointer. A data storage library may limit the number of concurrently-managed selection handles to exactly one. The library developer must document such restrictions and application developers should be aware of such restrictions.

CSSM_RETURN CSSMDLI DL_CrlAbortQuery ( ) cancels the query initiated by the DL_CrlGetFirst function and resets the selection handle.

2.1.5 Module Management Functions

CSSM_RETURN CSSMDLI DL_Install ( ) accepts as input the name and GUID of a DL module, selected attributes describing the module, and information required by CSSM to dynamically load the module, if its use is requested by some application. The storage capabilities of some storage devices and implementations can not be fully determined until execution time. DLs using such devices must register all known, static capability information at install time. Incremental capability information can be added to the registry when a caller attaches to the module or at any time during the use of the module at install time. CSSM adds a DL module name and attributes to the registry of DL modules.

CSSM_RETURN CSSMDLI DL_Uninstall ( ) CSSM removes a specified DL module from the registry.

CSSM_DLINFO_PTR CSSMDLI DL_GetInfo ( ) CSSM returns a structure describing the identity and storage capabilities of a DL, as it is currently recorded in a CSSM registry. Multiple capability structures are returned if the DL manages multiple storage mediums (such as two custom hardware storage devices). The storage capabilities of certain types of devices are not known at install time. The caller can specify if such incomplete capability descriptions should or should not be included in the returned information.

CSSM_RETURN CSSMDLI DL_FreeInfo ( ) frees the memory structure CSSM allocated to hold the module-descriptive information returned by the CSSM_DL_GetInfo function. Multiple structures can be freed if multiple structures were returned by the DL_GetInfo function.

CSSM_LIST_PTR CSSMDLI DL_ListModules ( ) CSSM returns a list of all currently-registered DL modules.

CSSM_DL_HANDLE CSSMDLI DL_Attach ( ) accepts as input the GUID of a DL module, the major and minor version number desired by the caller, and an optional deviceID and device access flags for special storage devices. The caller is requesting a dynamic load of the specified DL module, if the available version of a DL module is compatible with the version level specified by the caller. The caller may be an application, a TP module, a CL module, or another DL module.

CSSM_RETURN CSSMDLI DL_Detach ( ) the caller is requesting the dynamic unload of a specified DL module.

CSSM_NAME_LIST_PTR CSSMDLI DL_GetDbNames ( ) - the specified DL module returns a memory-resident list of the logical data store names that this module can access and a count of the number of logical names in that list. A DL module is responsible for allocating the memory required for the list.

CSSM_RETURN CSSMDLI DL_FreeNameList ( ) - frees the list returned by DL_GetDbNames function.

2.1.6 Extensibility Functions

CSSM_DATA_PTR CSSMDLI DL_PassThrough ( ) - accepts as input an operation ID and a set of arbitrary input parameters. The operation ID may specify any type of operation a DL wishes to export for use by an application or by another module. Such operations may include queries or services that are specific to certain types of certificates, or to the relationships between the certificates and CRLs manipulated by a DL module.

2.2 Data storage Data Structures

```
typedef uint32 CSSM_DL_HANDLE    /* data storage library Handle */
typedef uint32 CSSM_DB_HANDLE    /* Data storage Handle */
typedef uint32 CSSM_MODULE_HANDLE, *CSSM_MODULE_HANDLE_PTR /* Service provider
                                                            Handle*/
```

2.2.1 CSSM_DB_LONGHANDLE

A pair of data library module handle and database handle.

```
typedef struct {
    CSSM_DL_HANDLE DLHandle;
    CSSM_DB_HANDLE DBHandle;
} CSSM_DB_LONGHANDLE, *CSSM_DB_LONGHANDLE_PTR;
```

2.2.2 CSSM_DB_LIST

This list pairs a data store with a data storage library Module that can be used to manage that data store. The list is often used to provide other modules with the set of handles required to search an application-selected data store.

```
typedef struct {
    uint32 NumHandles;
    CSSM_DB_LONGHANDLE_PTR DBLongHandle;
} CSSM_DB_LIST, *CSSM_DB_LIST_PTR;
```

Definition:

NumHandles - Number of database sources in the list.

`DBLongHandle` - List of data library module/database pairs.

2.2.3 CSSM_DB_TYPE

This is the initial enumeration of semantic, user-defined data store types. Each data store can have an associated, semantic type defined by the user/creator of the data store. New types should be added as required to describe key databases and policy object databases, among others.

```
typedef enum {
       CSSM_DB_TAG_NONE=0L,
       CSSM_DB_TAG_ROOTS,    /* contains self-signed root certs */
       CSSM_DB_TAG_TRUSTED,         /* re-issued locally (DL must protect?) */
       CSSM_DB_TAG_SYSTEM,   /* contains CSSM system certs */
       CSSM_DB_TAG_OWNER,   /* certs owned by users (ie private key in CSP) */
       CSSM_DB_TAG_REVOKED   /* contains revocation info - used w   ith CRL APIs */
       /* others? */
} CSSM_DB_TYPE;
```

2.2.4 CSSM_DATA_RECORD_TYPE

These are data record types that may be stored and managed by data storage library modules. New types can be added to this list. Storage modules may store and manage multiple types in a single data store or in separate data stores.

```
typedef enum  cssm_data_record_type {
       CSSM_DATA_RECORD_ANY,            /* lib can store and manage any objects*/
       CSSM_DATA_RECORD_CERTS,          /* lib can store & manage certs */
       CSSM_DATA_RECORD_CRLS,           /* lib can  store & manage CRLS */
       CSSM_DATA_RECORD_KEYS,           /* lib can store & manage keys */
       CSSM_DATA_RECORD_POLICIES,       /* lib can store & manage policies */
       CSSM_DATA_RECORD_GENERIC         /* lib can store & manage generic data */
} CSSM_DATA_RECORD_TYPE, *CSSM_DATA_RECORD_TYPE_PTR;
```

2.2.5 CSSM_DB_ATTRIBUTE_USAGE

These are the ways in which an attribute of a data store record can be treated by the data storage library module. The current options include indexed attributes and non-indexed attributes. Indexes may be one-to-many (which is a regular index), or one-to-one (which is a unique index). Any given DL may or may not support all attribute usages.

```
typedef enum  cssm_db_attribute_usage {
       CSSM_DB_INDEX,
       CSSM_DB_UNIQUE_INDEX,
       CSSM_DB_NON_INDEX
}   CSSM_DB_ATTRIBUTE_USAGE, *CSSM_DB_ATTRIBUTE_USAGE_PTR;
```

2.2.6 CSSM_ATTRIBUTE_ID_FORMAT

These are the distinct representations for an attribute identifier. An attribute field of a data record may be identified by an OID value or an attribute name.

```
typedef enum cssm_attribute_id_format {
    CSSM_OID_NAME_FORMAT,
    CSSM_STRING_NAME_FORMAT
} CSSM_ATTRIBUTE_ID_FORMAT, *CSSM_ATTRIBUTE_ID_FORMAT_PTR
```

2.2.7 CSSM_ATTRIBUTE_NAME

This structure defines an attribute name that can be used to reference an attribute field of a data store record when selecting records from the data store or when creating a new data store.

```
typedef struct cssm_attribute_name {
       CSSM_ATTRIBUTE_ID_FORMAT AttributeIdFormat;
       CSSM_DATA AttributeId;
} CSSM_ATTRIBUTE_NAME, *CSSM_ATTRIBUTE_NAME_PTR
```

Definition:
AttributeIdFormat - Indicates the format of the attribute identifier in the *AttributeId*.

AttributeId - An identifier that uniquely identifies an attribute field contained in data store records. The format of the identifier is specified by the *AttributeIdFormat*.

2.2.8 CSSM_DB_ATTRIBUTE_INFO

These are the semantic types of data stores that can be managed by a data storage library module.

```
typedef struct {
       CSSM_DB_ATTRIBUTE_NAME  AttributeName;
       CSSM_DB_ATTRIBUTE_USAGE  AttributeUsage;
}  CSSM_DB_ATTRIBUTE_INFO, CSSM_ATTRIBUTE_INFO_PTR;
```

Definition:

AttributeName - Name of an attribute field of a data store record.

AttributeUsage - Indicates how this attribute can or should be managed by the data storage library module.

2.2.9 CSSM_DB_RECORD_INFO

This structure defines the meta-information about an individual record type in a single data store. This information is maintained by the data storage library module for each data store it manages.

```
typedef struct {
       CSSM_DATA_RECORD_TYPE DataRecordType;
       uint32 NumberOfAttributes;
       CSSM_DB_ATTRIBUTE_INFO_PTR AttributeInfo;
} CSSM_DB_RECORD_INFO,  *CSSM_DB_RECORD_INFO_PTR;
```

Definition:

DataRecordType - A type of data record stored in this data store.

NumberOfAttributes - The number of directly or indirectly named attributes for this record type.

AttributeInfo - A pointer to an array of structures describing each attribute for this record type. Each structure provides a name for the attribute, and a usage mode for the attribute. Usage modes include indexed attributes and non-indexed attributes. An index may be one-to-many or one-to-one with the data store records. The array contains *NumberOfAttributes* entries.

2.2.10 CSSM_DBINFO

This structure defines all meta-information about an individual data store. This information is maintained by the data storage library module for each data store it manages.

```
typedef struct {
       CSSM_DB_TYPE DbType;
       uint32 NumberOfRecordTypes;
       CSSM_DB_RECORD_INFO_PTR RecordInfo;
       void *reserved;
} CSSM_DBINFO,  *CSSM_DBINFO_PTR;
```

Definition:

DbType - Indicates the user-defined semantic type of the records stored in this data store. Typically this indicates how applications should use the records in this data store.

NumberOfRecordTypes-The number of distinct record type (formats) stored in this data store.

RecordInfo - A pointer to an array of structures describing the attribute format of each record type stored in this data store. The array contains*NumberOfRecordTypes*entries.

Reserved1 - Reserved for future use.

2.2.11 CSSM_DB_CONJUNCTIVE

These are the conjunctive operations which can be used when specifying a selection criterion.

```
typedef enum cssm_db_conjunctive{
     CSSM_NONE
     CSSM_AND,
     CSSM_OR
} CSSM_DB_CONJUNCTIVE
```

2.2.12 CSSM_DB_OPERATOR

These are the logical operators which can be used when specifying a selection predicate.

```
typedef enum cssm_db_operator {
    CSSM_EQUAL,
    CSSM_NOT_EQUAL,
    CSSM_LESS_THAN,
    CSSM_GREATER_THAN
} CSSM_DB_OPERATOR
```

2.2.13 CSSM_ QUERY_TAG

This tag decides whether the data object retrieval is based on query string or selection predicates.

```
typedef enum cssm_tag_query {
    CSSM_QUERY_NONE,
    CSSM_QUERY_STRING,
    CSSM_QUERY_PREDICATES
} CSSM_QUERY_TAG
```

2.2.14 CSSM_SELECTION_PREDICATE

This structure defines the selection predicate to be used for database queries.

```
typedef struct cssm_selection_predicate {
       CSSM_DB_OPERATOR dbOperator;
       CSSM_ATTRIBUTE_NAME AttributeName;
       CSSM_DATA AttributeValue;
} CSSM_SELECTION_PREDICATE, *CSSM_SELECTION_PREDICATE_PTR
```

Definition:

dbOperator - The relational operator to be used when comparing the value contained in *AttributeValue* to values contained in the data store.

AttributeName - The name of an attribute in a data store record. This attribute is a target for comparison when selecting records from the data store.

AttributeValue - The value used in comparisons with values stored in the *AttributeName* field of data store records. If no *AttributeName* is specified, then the *AttributeValue* can be a selection predicate of any format supported by the data storage library module.

2.2.15 CSSM_QUERY_PREDICATE

This structure defines the Query predicate to be used for database queries.

```
typedef struct cssm_query_predicate {
       CSSM_DB_CONJUNCTIVE Conjunctive;
       uint32 NumSelectionPredicates;
       CSSM_SELECTION_PREDICATE_PTR SelectionPredicate;
} CSSM_QUERY_PREDICATE, *CSSM_QUERY_PREDICATE_PTR
```

Definition:
Conjunctive - Indicates the conjunctive to be used to join the predicates.

NumSelectionPredicates- Number of election predicates in the query.

SelectionPredicate - Pointer to the array of selection predicates.

2.2.16 CSSM_QUERY

This union contains the array of selection predicates and a CSSM_DATA_PTR.
Depending on the CSSM_QUERY_TAG, one of these has to be selected..

```
typedef union cssm_query {
       CSSM_DATA QueryString;
       CSSM_QUERY_PREDICATE_PTR QueryPredicate;
} CSSM_QUERY, *CSSM_QUERY_PTR
```

Definition:
QueryString - Pointer to query string.

QueryPredicate - Query predicate pointer which points to an array of selection predicates.

2.2.17 CSSM_INDEX_RECORD

This structure defines the index values to be inserted or deleted.

```
typedef struct cssm_index_record {
       CSSM_ATTRIBUTE_NAME IndexName;
       CSSM_DATA IndexValue;
} CSSM_INDEX_RECORD, *CSSM_INDEX_RECORD_PTR
```

Definition:

IndexName - The name of an attribute in a data store record. This attribute is a target for comparison when selecting records from the data store.

IndexValue - The value used in comparisons with values stored in the *IndexName* field of data store records. If no *IndexName* is specified, then the *IndexValue* can be any format supported by the data storage library module.

2.2.18 CSSM_DL_MODULE_TYPE

These are the module implementation types for data storage library modules.

typedef uint32 CSSM_DL_MODULE_TYPE

```
#define CSSM_STORE_LOCAL_MEMORY 0x00000001       /* implementation uses a local
                                                  memory cache */
#define CSSM_STORE_LOCAL_FILE   0x00000002       /* implement. uses a file system
                                                  service */
#define CSSM_STORE_LOCAL_DBMS   0x00000004       /* implementation uses a local
                                                  DBMS */
#define CSSM_STORE_REMOTE_DBMS  0x00000008       /* implementation uses a remote
                                                 DBMS */
#define CSSM_STORE_REMOTE_DIR   0x00000010       /* implementation uses a remote
                                                  directory service */
#define CSSM_STORE_TOKEN        0x00000020       /* implementation uses a
                                                 hardware token */
#define CSSM_STORE_REMOVEABLE   0x00000040       /* implementation uses a
removable

                                                 store device */
#define CSSM_STORE_UNSPECIFIED  0x00000080       /* implementation is unspecified
                                                 */
```

2.2.19 CSSM_DL_ACCESS_TYPE

```
#define       CSSM_DL_STORE_ACCESS_SERIAL              CSSM_CSP_SESSION_SERIAL
#define       CSSM_DL_STORE_ACCESS_EXCLUSIVE     CSSM_CSP_SESSION_EXCLUSIVE
```

2.2.20 CSSM_DL_INFO

This structure contains all of the static data associated with a data storage library add-in module. This information is added to the CSSM registry at install time. It can be queried using the command **CSSM_DL_GetInfo ( )**

```
typedef struct cssm_dlinfo{
    uint32 VerMajor;
    uint32 VerMinor;
    CSSM_DL_MODULE_TYPE DLModuleType;
    uint32 DeviceID;
    CSSM_BOOL CapabilitiesInitialized;
    uint32 DeviceAccessFlags;
    CSSM_DATA ExclusiveDLMCertificate;
    CSSM_BOOL LoginRequired;
    uint32 NumberOfRecordTypes;
    CSSM_DATA_RECORD_TYPE_PTR DataRecordTypes;
    uint32 NumberOfAttributeUsageTypes;
    CSSM_DB_ATTRIBUTE_USAGE_PTR AttributeUsageTypes;
    uint32 NumberOfAttributeIdFormats;
    CSSM_ATTRIBUTE_ID_FORMAT_PTR AttributeIdFormats;
    uint32 NumberOfRelOperatorTypes;
    CSSM_DB_OPERATOR_PTR RelOperatorTypes;
    uint32 NumberOfConjOperatorTypes;
    CSSM_DB_CONJUNCTIVE_PTR ConjOperatorTypes;
    CSSM_DATA_PTR Reserved1;
}CSSM_DLINFO, *CSSM_DLINFO_PTR
```

Definition:

VerMajor - The major version number of the add-in module.

VerMinor - The minor version number of the add-in module.

DLModuleType - Indicates the underlying implementation approach for this library module.

DeviceID - The ID of a hardware storage device managed by this data storage library module.

CapabilitiesInitialized - True or false, indicating whether complete capabilities are currently specified in this DLinfo structure.

DeviceAccessFlags - A bitmask of the device access modes supported by this library module.

ExclusiveDLMCertificate - The certificate used to sign certificates issued to exclusive users of this data storage library module.

LoginRequired - True or false, indicating whether a DL requires caller login and logout.

NumberOfRecordTypes - The number of distinct data record types that can be stored and managed by this library module in one or more data stores.

DataRecordTypes - An array listing the data record types that can be stored and managed by this library module in one or more data stores. The array contains *NumberOfRecordTypes* entries.

NumberOfAttributeUsageTypes - The number of distinct attribute usages supported by a DL.

AttributeUsageTypes - An array listing the attribute usages supported by the data storage library when defining the schema for a new data store. Currently-defined usages include: indexed attribute, unique-index attribute, and non-indexed attribute. The array contains *NumberOfAttributeUsageTypes* entries.

NumberOfAttributeIdFormats - The number of distinct attribute identification formats that are supported by this data storage library module.

AttributeIdFormats - An array listing the formats accepted by the data storage library to identify record fields in a selection query. Currently-defined usages include: OID-name format and string-name format. The array contains *NumberOfAttributeIdFormats* entries.

NumberOfRelOperatorTypes - The number of distinct binary relational operators supported by this library module.

RelOperatorTypes - An array listing the relational operators supported by this library module for defining selection predicates for retrieving stored data objects. The array contains *NumberOfRelOperatorTypes* entries.

NumberOfConjOperatorTypes - The number of distinct conjunctive operators supported by this library module.

ConjOperatorTypes - An array listing the conjunctive operators supported by this library module for defining selection predicates for retrieving stored data objects. The array contains *NumberOfConjOperatorTypes* entries.

2.3 *Reserved1* - Reserved for future use.

Data source Operations

2.3.1 DL_DbOpen

CSSM_DB_HANDLE CSSMDLI DL_DbOpen (CSSM_DL_HANDLE DLHandle,
const char *DbName)

This function opens the data store with the specified logical name.

Parameters

DLHandle (input)
The handle that describes the add-in data storage library module to be used to perform this function.

DbName (input)
A pointer to the string containing the logical name of the data store.

Return Value

Returns the CSSM_DB_HANDLE of the opened data store. If the handle is NULL, an error has occurred. Use CSSM_GetError to obtain the error code.

Error Codes

Value	Description
CSSM_DL_INVALID_DL_HANDLE	Invalid DL handle
CSSM_DL_DATASTORE_NOT_EXISTS	The data store with the logical name does not exist
CSSM_DL_DB_OPEN_FAIL	Open caused an exception
CSSM_DL_MEMORY_ERROR	Error in allocating memory

See Also

DL_DbClose.

2.3.2 DL_DbClose

CSSM_RETURN CSSMDLI DL_DbClose (CSSM_DL_HANDLE DLHandle,
CSSM_DB_HANDLE DBHandle)

This function closes an open data store.

Parameters

DLHandle (input)
The handle that describes the add-in data storage library module to be used to perform this function.

DBHandle (input)
The handle that describes the data store to be used when performing this function.

Return Value

A CSSM_OK return value signifies that the function completed successfully. When CSSM_FAIL is returned, an error has occurred. Use CSSM_GetError to obtain the error code.

Error Codes

Value	Description
CSSM_DL_INVALID_DL_HANDLE	Invalid DL handle
CSSM_DL_INVALID_DB_HANDLE	Invalid DB handle
CSSM_DL_DB_CLOSE_FAIL	Close caused an exception

See Also

DL_DbOpen.

2.3.3 DL_DbCreate

```
CSSM_DB_HANDLE CSSMDLI DL_DbCreate
                              (CSSM_DL_HANDLE DLHandle,
                              CSSM_TP_HANDLE TPHandle,
                              const char *DbName,
                              CSSM_MODULE_HANDLE_PTR ModuleHandles,
                              CSSM_DBINFO_PTR DataStoreSchema)
```

This function creates and opens a new data store. The name of the new data store is specified by the input parameter DbName. The record schema for the data store is specified in the DBINFO structure.

Parameters

DLHandle (input)
The handle that describes the add-in data storage library module used to perform this function.

TPHandle (input)
The handle that describes the module to be used to determine the semantic, user-defined type to be associated with the new data store.

DbName (input)
The general, external name for the new data store.

ModuleHandles (input)
An array of handles, one per record type to be stored in this data store. Each handle describes a service provider module that can be invoked to indirectly access values contained in records that will be stored in this new data store. The array must contain *DataStoreSchema.NumberOfRecordTypes* entries. Handles corresponding to attributes named by user-defined strings rather than by library-processed OIDs are ignored.

DataStoreSchema (input)
A pointer to a structure describing the format/schema of each record type that will be stored in the new data store.

Return Value

A handle to the newly created, open data store. When NULL is returned, an error has occurred. Use CSSM_GetError to obtain the error code.

Error Codes

Value	Description
CSSM_DL_INVALID_DL_HANDLE	Invalid DL handle
CSSM_DL_INVALID_CSP_HANDLE	Invalid Cryptographic Service Provider handle
CSSM_DL_INVALID_TP_HANDLE	Invalid Trust Policy Module handle
CSSM_DL_INVALID_MODULE_HANDLE	Invalid Module handle for accessing the data record
CSSM_DL_INVALID_DATA_PTR	Invalid data pointer
CSSM_DL_DB_CREATE_FAIL	Create caused an exception
CSSM_DL_MEMORY_ERROR	Error in allocating memory

See Also

DL_DbOpen, DL_DbClose, DL_DbDelete.

2.3.4 DL_DbDelete

CSSM_RETURN CSSMDLI DL_DbDelete (CSSM_DL_HANDLE DLHandle,
const char *DbName)

This function deletes all records from the specified data store and removes all state information associated with that data store.

Parameters

DLHandle (input)
The handle that describes the add-in data storage library module to be used to perform this function.

DbName (input)
A pointer to the string containing the logical name of the data store.

Return Value

A CSSM_OK return value signifies that the function completed successfully. When CSSM_FAIL is returned, an error has occurred. Use CSSM_GetError to obtain the error code.

Error Codes

Value	Description
CSSM_DL_INVALID_DL_HANDLE	Invalid DL handle
CSSM_DL_INVALID_DB_HANDLE	Invalid DB handle
CSSM_DL_DB_DELETE_FAIL	Delete caused an exception

See Also

DL_DbCreate, DL_DbOpen, DL_DbClose.

2.3.5 DL_DbImport

CSSM_RETURN CSSMDLI DL_DbImport (CSSM_DL_HANDLE DLHandle,
const char *DbDestLogicalName,
const char *DbSrcFileName)

This function imports data store records from a file into a new data store.

Parameters

DLHandle (input)
The handle that describes the add-in data storage library module to be used to perform this function.

DbDestLogicalName (input)
The name of the destination data store in which to insert the records.

DbSrcFileName(input)
The name of the source file from which to obtain the records that are added to the data store.

Return Value

A CSSM_OK return value signifies that the function completed successfully. When CSSM_FAIL is returned, an error has occurred. Use CSSM_GetError to obtain the error code.

Error Codes

Value	Description
CSSM_DL_INVALID_DL_HANDLE	Invalid DL handle
CSSM_DL_INVALID_PTR	NULL source or destination file names
CSSM_DL_DB_IMPORT_FAIL	DB exception doing import function
CSSM_DL_MEMORY_ERROR	Error in allocating memory

See Also

DL_DbExport.

2.3.6 DL_DbExport

```
CSSM_RETURN CSSMDLI DL_DbExport  (CSSM_DL_HANDLE DLHandle,
                                 const char *DbSrcLogicalName,
                                 const char *DbDestFileName)
```

This function exports a copy of the data store records from the source data store to a file.

Parameters

DLHandle (input)
The handle that describes the add-in data storage library module to be used to perform this function.

DbSrcLogicalName (input)
The name of the data store from which the records are to be exported.

DbDestFileName (*input)*
The name of the destination file which will contain a copy of the source data store's records.

Return Value

A CSSM_OK return value signifies that the function completed successfully. When CSSM_FAIL is returned, an error has occurred. Use CSSM_GetError to obtain the error code.

Error Codes

Value	Description
CSSM_DL_INVALID_DL_HANDLE	Invalid DL handle
CSSM_DL_INVALID_PTR	NULL source or destination file names
CSSM_DL_DB_EXPORT_FAIL	DB exception doing export function
CSSM_DL_MEMORY_ERROR	Error in allocating memory

See Also

DL_DbImport.

2.3.7 DL_DbSetInfo

CSSM_RETURN CSSMDLI DL_DbSetInfo (CSSM_DL_HANDLE DLHandle,
const char* DbName
CSSM_DBINFO_PTR DbInfo)

This function sets information describing the datasource.

Parameters

DLHandle (input)
The handle that describes the add-in data storage library module used to perform this function.

DbName(input)
A pointer to the string containing the data source name.

DbInfo
A pointer to CSSM_DBINFO structure.

Return Value

A CSSM_OK return value signifies that the function completed successfully. When CSSM_FAIL is returned, an error has occurred. Use CSSM_GetError to obtain the error code.

Error Codes

Value	Description
CSSM_DL_INVALID_DL_HANDLE	Invalid DL handle
CSSM_DL_INVALID_POINTER	Invalid pointer
CSSM_REGISTRY_ERROR	Unable to write to registry
CSSM_ INVALID_DBINFO_ POINTER	Invalid CSSM_DBINFO pointer

See Also

DL_GetInfo, DL_FreeDbInfo.

2.3.8 DL_DbGetInfo

CSSM_DBINFO_PTR CSSMDLI DL_DbGetInfo (CSSM_DL_HANDLE DLHandle,
const char* DbName)

This function returns information describing a DL module and its capabilities.

Parameters

DLHandle (input)
The handle that describes the add-in data storage library module used to perform this function.

DbName(input)
A pointer to the string containing the data source name.

Return Value

A pointer to an array of one or more CSSM_DLINFO structures containing information about a DL module. There is one Dlinfo structure for each distinct hardware storage device managed by this DL. Most DLs manage only one device. If the pointer is NULL, an error has occurred. Use CSSM_GetError to obtain the error code.

Error Codes

Value	Description
CSSM_INVALID_POINTER	Invalid pointer
CSSM_MEMORY_ERROR	Internal memory error

See Also

DL_DbSetInfo, DL_DbFreeDbInfo.

2.3.9 DL_FreeDbInfo

CSSM_RETURN CSSMDLI DL_FreeDbInfo (CSSM_DBNFO_PTR DbInfo,
uint32 NumberOfDbInfos)

This function frees the memory allocated by a DL module for the CSSM_DLINFO structure returned by the CSSM_DL_GetInfo function.

Parameters

DLInfo (input)
A pointer to CSSM_DL_Info structure.

numberOfInfos (input)
The number of DL Info structures to be freed.

Return Value

A CSSM_OK return value signifies that the function completed successfully. When CSSM_FAIL is returned, an error has occurred. Use CSSM_GetError to obtain the error code.

Error Codes

Value	Description
CSSM_ INVALID_DLINFO_ POINTER	Invalid CSSM_DLINFO pointer

See Also

DL_DbGetInfo.

2.3.10 DL_GetDbHandleToName

```
CSSM_NAME_LIST_PTR CSSMDLI DL_GetDbHandleToName(CSSM_DL_HANDLE DLHandle,
                                                CSSM_DB_HANDLE DbHandle)
```

This function retrieves the data source name corresponding to an opened database handle. A DL module is responsible for allocating the memory required for the list.

Parameters

DLHandle (input)
The handle that describes the add-in data storage library module to be used to perform this function.

DbHandle (input)
The handle to an opened database.

Return Value

Returns a pointer to a CSSM_NAME_LIST structure which contains a list of data store names. If the pointer is NULL, an error has occurred. Use CSSM_GetError to obtain the error code.

Error Codes

Value	Description
CSSM_DL_MEMORY_ERROR	Error allocating memory
CSSM_DL_INVALID_DB_HANDLE	Invalid DB Handle
CSSM_DL_INVALID_DL_HANDLE	Invalid DL Handle

See Also

DL_FreeNameList.

2.4 Generic Security Object Storage Operations

2.4.1 DL_DataInsert

CSSM_RETURN CSSMDLI DL_DataInsert

```
                                        (CSSM_DL_HANDLE DLHandle,
                                        CSSM_DB_HANDLE DBHandle,
                                        CSSM_MODULE_HANDLE_PTR  ModuleHandles,
                                        CSSM_DATA_RECORD_TYPE DataRecordType,
                                        CSSM_INDEX_RECORD_PTR IndexRecord,
                                        const CSSM_DATA_PTR DataRecord)
```

This function makes the DataRecord persistent by inserting it into the specified data store. The DataRecord contains two logically distinct sets of data values: immutable values and update-able values. The ModuleHandle specifies the library a DL should use to access sub-fields of the data record as required during the add process.

Parameters

DLHandle (input)
The handle that describes the add-in data storage library module to be used to perform this function.

DBHandle (input)
The handle that describes the data store to be used when performing this function.

ModuleHandles (input/optional)
An array of handles, one per record type to be stored in this data store. Each handle describes a service provider module that can be invoked to indirectly access values contained in records that will be stored in this new data store. The array must contain *DataStoreSchema.NumberOfRecordTypes* entries. Handles corresponding to attributes named by user-defined strings rather than by library-processed OIDs are ignored.

DataRecordType (input)
Indicates the type of data record being added to the data store. The value cannot be CSSM_DATA_RECORD_ANY.

IndexRecord (input/Optional)
Indicates the values for the meta data for the record if module handle is NULL

DataRecord (input)
A pointer to the CSSM_DATA structure which contains the data to be added to the data store.

Return Value

A CSSM_OK return value signifies that the function completed successfully. When CSSM_FAIL is returned, an error has occurred. Use CSSM_GetError to obtain the error code.

Error Codes

Value	Description
CSSM_DL_INVALID_DL_HANDLE	Invalid data storage library handle

CSSM_DL_INVALID_DB_HANDLE	Invalid Data Store handle
CSSM_DL_INVALID_MODULE_HANDLE	Invalid Module handle for accessing the data record
CSSM_DL_INVALID_DATA_PTR	Invalid data pointer
CSSM_DL_DATA_INSERT_FAIL	Add caused an exception

See Also

DL_DataDelete.

2.4.2 DL_DataDelete

```
CSSM_RETURN CSSMDLI DL_DataDelete
                    (CSSM_DL_HANDLE DLHandle,
                    CSSM_DB_HANDLE DBHandle,
                    CSSM_MODULE_HANDLE_PTR  ModuleHandles,
                    CSSM_DATA_RECORD_TYPE DataRecordType,
                    CSSM_INDEX_RECORD_PTR IndexRecord,
                    const CSSM_DATA_PTR DataRecord)
```

This function removes the DataRecord of the specified DataRecordType from the specified data store.

Parameters

DLHandle (input)
The handle that describes the add-in data storage library module to be used to perform this function.

DBHandle (input)
The handle that describes the data store to be used when performing this function.

ModuleHandles (input/Optional)
An array of handles, one per record type to be stored in this data store. Each handle describes a service provider module that can be invoked to indirectly access values contained in records that will be stored in this new data store. The array must contain *DataStoreSchema.NumberOfRecordTypes* entries. Handles corresponding to attributes named by user-defined strings rather than by library-processed OIDs are ignored.

DataRecordType (input)
Indicates the type of data record being deleted from the data store.

IndexRecord (input/Optional)
Indicates the values for the meta data for the record if module handle is NULL

DataRecord (input)
A pointer to the CSSM_DATA structure which contains the data to be deleted from the data store.

Return Value

A CSSM_OK return value signifies that the function completed successfully. When CSSM_FAIL is returned, an error has occurred. Use CSSM_GetError to obtain the error code.

Error Codes

Value	Description
CSSM_DL_INVALID_DL_HANDLE	Invalid data storage library handle
CSSM_DL_INVALID_DATA_PTR	Invalid data pointer
CSSM_DL_INVALID_DB_HANDLE	Invalid Data Storage handle
CSSM_DL_DATA_NOT_IN_DB	Data record not in Data Store
CSSM_DL_DATA_DELETE_FAIL	Delete caused an exception

See Also

DL_DataInsert.

2.4.3 DL_DataGetFirst

CSSM_DATA_PTR CSSMDLI DL_DataGetFirst

```
(CSSM_DL_HANDLE DLHandle,
CSSM_DB_HANDLE DBHandle,
CSSM_DATA_RECORD_TYPE DataRecordType,
CSSM_QUERY_TAG QueryTag,
CSSM_QUERY_PTR Query,
CSSM_HANDLE_PTR ResultsHandle,
uint32 *NumberOfMatchedDataRecords)
```

This function retrieves the first object in the data store that matches the selection criteria. The selection criteria is expressed in one of two ways: a predicate formed from a set of triples (OID, value, relational operator) connected by a single conjunctive operator, or a single string whose required substructure and semantics are defined by the implementing module. This function returns a count of the total number of objects matching the selection criteria, the first object matching the criteria, and a selection handle that may be used to retrieve the subsequent objects matching the selection criteria.

Parameters

DLHandle (input)
The handle that describes the add-in data storage library module to be used to perform this function.

DBHandle (input)
The handle that describes the data store to be used when performing this function.

DataRecordType (input)
A type of data records to be searched by this query.

QueryTag (input/Optional)
Indicates whether a Query string or an array of seletion predicates is used for the query.

Query (input/Optional)
If the Query Tag is Query_String, CSSM_DATA_PTR is used as query string, otherwise CSSM_SELECTIONA_PREDICATE_PTR is used to build the query string.

ResultsHandle (output)
This handle should be used to retrieve subsequent records that matched this selection criteria.

NumberOfMatchedDataRecords (output)
Returns the total number of data records that match the selection criteria.

Return Value

Returns a pointer to a CSSM_DATA structure which contains the first data record in the data store that matches the selection criteria. If the pointer is NULL, an error has occurred. Use CSSM_GetError to obtain the error code.

Error Codes

Value	Description
CSSM_DL_INVALID_DL_HANDLE	Invalid DL handle
CSSM_DL_INVALID_SELECTION_PTR	Invalid selection predicate pointer
CSSM_DL_INVALID_DB_HANDLE	Invalid DB handle
CSSM_DL_NO_DATA_FOUND	No data records match the selection predicate
CSSM_DL _DATA_GETFIRST_FAIL	Opening the records caused an exception
CSSM_DL_MEMORY_ERROR	Error in allocating memory

See Also

DL_DataGetNext, DL_DataAbortQuery.

2.4.4 DL_DataGetNext

CSSM_DATA_PTR CSSMDLI DL_DataGetNext

```
                                        (CSSM_DL_HANDLE DLHandle,
                                         CSSM_DB_HANDLE DBHandle,
                                         CSSM_HANDLE ResultsHandle,
                                         CSSM_DATA_RECORD_TYPE_PTR DataRecordType)
```

This function returns the next object referenced by the ResultsHandle. The ResultsHandle was returned by the DataGetFirst functions.

Parameters

DLHandle (input)
The handle that describes the add-in data storage library module to be used to perform this function.

DBHandle (input)
The handle that describes the data store to be used when performing this function.

ResultsHandle (input)
The selection handle returned from the DL_DataGetFirst function.

DataRecordType (output)
The type of the data record returned by this query.

Return Value

Returns a pointer to a CSSM_DATA structure which contains the next data record in the data store that matches the original selection criteria. If the pointer is NULL, an error has occurred. Use CSSM_GetError to obtain the error code.

Error Codes

Value	Description
CSSM_DL_INVALID_DL_HANDLE	Invalid data storage library handle
CSSM_DL_INVALID_RESULTS_HANDLE	Invalid query handle
CSSM_DL_INVALID_DB_HANDLE	Invalid Data Store handle
CSSM_DL_NO_MORE_RECORDS	No more records for this selection handle
CSSM_DL_DATA_GETNEXT_FAIL	Opening the records caused an exception
CSSM_DL_MEMORY_ERROR	Error in allocating memory

See Also

DL_DataGetFirst, DL_DataAbortQuery.

2.4.5 DL_DataAbortQuery

```
CSSM_RETURN CSSMDLI DL_DataAbortQuery(CSSM_DL_HANDLE DLHandle,
                                     CSSM_DB_HANDLE DBHandle,
                                     CSSM_HANDLE ResultsHandle)
```

This function terminates the query initiated by DL_DataGetFirst, and allows a DL to release all intermediate state information associated with the query.

Parameters

DLHandle (input)
The handle that describes the add-in data storage library module used to perform this function.

DBHandle (input)
The handle that describes the data store associated with the query.

ResultsHandle (input)
The selection handle returned from the initial query function.

Return Value

CSSM_OK if the function was successful. CSSM_FAIL if an error condition occurred. Use CSSM_GetError to obtain the error code.

Error Codes

Value	Description
CSSM_DL_INVALID_DL_HANDLE	Invalid data storage library Handle
CSSM_DL_INVALID_DB_HANDLE	Invalid data store handle
CSSM_DL_INVALID_RESULTS_HANDLE	Invalid results handle
CSSM_DL_DATA_ABORT_QUERY_FAIL	Unable to abort query

See Also

DL_DataGetFirst, DL_DataGetNext.

2.5 Certificate Storage Operations

2.5.1 DL_CertInsert

CSSM_RETURN CSSMDLI DL_CertInsert (CSSM_DL_HANDLE DLHandle,
CSSM_DB_HANDLE DBHandle,
CSSM_CL_HANDLE CLHandle,
const CSSM_DATA_PTR Cert)

This function makes the certificate persistent by inserting it into the specified data store.

Parameters

DLHandle (input)
The handle that describes the add-in data storage library module to be used to perform this function.

DBHandle (input)
The handle that describes the data store to be used when performing this function.

CLHandle (input)
The handle that describes the add-in certificate library module to be used to perform this function.

Cert (input)
A pointer to the CSSM_DATA structure which contains the certificate to be added to the data store.

Return Value

A CSSM_OK return value signifies that the function completed successfully. When CSSM_FAIL is returned, an error has occurred. Use CSSM_GetError to obtain the error code.

Error Codes

Value	Description
CSSM_DL_INVALID_DL_HANDLE	Invalid DL handle
CSSM_DL_INVALID_CERTIFICATE_PTR	Invalid certificate pointer
CSSM_DL_INVALID_DB_HANDLE	Invalid DB handle
CSSM_DL_CERT_INSERT_FAIL	Add caused an exception

See Also

DL_CertDelete.

2.5.2 DL_CertDelete

```
CSSM_RETURN CSSMDLI DL_CertDelete  (CSSM_DL_HANDLE DLHandle,
                                    CSSM_DB_HANDLE DBHandle,
                                    CSSM_CL_HANDLE CLHandle,
                                    const CSSM_DATA_PTR Cert)
```

This function removes the certificate from the specified data store.

Parameters

DLHandle (input)
The handle that describes the add-in data storage library module to be used to perform this function.

DBHandle (input)
The handle that describes the data store to be used when performing this function.

CLHandle (input)
The handle that describes the add-in certificate library module to be used to perform this function.

Cert (input)
A pointer to the CSSM_DATA structure which contains the certificate to be deleted from the data store.

Return Value

A CSSM_OK return value signifies that the function completed successfully. When CSSM_FAIL is returned, an error has occurred. Use CSSM_GetError to obtain the error code.

Error Codes

Value	Description
CSSM_DL_INVALID_DL_HANDLE	Invalid DL handle
CSSM_DL_INVALID_CERTIFICATE_PTR	Invalid certificate pointer
CSSM_DL_INVALID_DB_HANDLE	Invalid DB handle
CSSM_DL_CERTIFICATE_NOT_IN_DB	Certificate not in DB
CSSM_DL_CERT_DELETE_FAIL	Delete caused an exception

See Also

DL_CertInsert.

2.5.3 DL_CertRevoke

```
CSSM_RETURN CSSMDLI DL_CertRevoke  (CSSM_DL_HANDLE DLHandle,
                                    CSSM_DB_HANDLE DBHandle,
                                    const CSSM_DATA_PTR CertToBeRevoked)
```

This function makes persistent the knowledge that this certificate has been revoked.

Parameters

DLHandle (input)
The handle that describes the add-in data storage library module to be used to perform this function.

DBHandle (input)
The handle that describes the data store to be used when performing this function.

CertToBeRevoked (input)
A pointer to the CSSM_DATA structure which contains the certificate to be marked as revoked.

Return Value

A CSSM_OK return value signifies that the function completed successfully. When CSSM_FAIL is returned, an error has occurred. Use CSSM_GetError to obtain the error code.

Error Codes

Value	Description
CSSM_DL_INVALID_DL_HANDLE	Invalid DL handle
CSSM_DL_INVALID_CERTIFICATE_PTR	Invalid certificate pointer
CSSM_DL_INVALID_DB_HANDLE	Invalid DB handle
CSSM_DL_CERT_REVOKE_FAIL	Update caused an exception

2.5.4 DL_CertGetFirst

CSSM_DATA_PTR CSSMDLI DL_CertGetFirst

```
(CSSM_DL_HANDLE DLHandle,
CSSM_DB_HANDLE DBHandle,
CSSM_SELECTION_PREDICATE_PTR SelectionPredicate,
uint32 SizeSelectionPredicate,
CSSM_DB_CONJUNCTIVE Conjunctive,
CSSM_HANDLE_PTR ResultsHandle,
uint32 *NumberOfMatchedCerts)
```

This function locates the first certificate in the data store which matches the selection criteria. The selection criteria is the expression formed by connecting all of the relational expressions of the selection predicate array using the one conjunctive operator. This function returns a count of the total number of certificates matching the selection criteria, the first certificate matching the criteria, and a selection handle that may be used to retrieve the subsequent certificates matching the selection criteria.

Parameters

DLHandle (input)
The handle that describes the add-in data storage library module to be used to perform this function.

DBHandle (input)
The handle that describes the data store to be used when performing this function.

SelectionPredicate (input)
A pointer to a CSSM_SELECTION_PREDICATE array which contains field-value/operator pairs.
If NULL, the first certificate in the data store is returned.

SizeSelectionPredicate(input)
The size of the selection predicate array.

Conjunctive (input)
The Boolean operator used to connect the selection predicates. If the selection predicate is null, this parameter is ignored.

ResultsHandle (output)
This handle should be used for subsequent retrievals for the same selection criteria.

NumberOfMatchedCerts (output)
Returns the total number of certificates that match the selection criteria.

Return Value

Returns a pointer to a CSSM_DATA structure which contains the first certificate in the data store that matches the selection criteria. If the pointer is NULL, an error has occurred. Use CSSM_GetError to obtain the error code.

Error Codes

Value	Description
CSSM_DL_INVALID_DL_HANDLE	Invalid DL handle
CSSM_DL_INVALID_SELECTION_PTR	Invalid selection predicate pointer
CSSM_DL_INVALID_DB_HANDLE	Invalid DB handle
CSSM_DL_NO_CERTIFICATE_FOUND	No certificates that match the selection predicate
CSSM_DL _CERT_GETFIRST_FAIL	Opening the records caused an exception
CSSM_DL_MEMORY_ERROR	Error in allocating memory

See Also

DL_CertGetNext, DL_CertAbortQuery.

2.5.5 DL_CertGetNext

```
CSSM_DATA_PTR CSSMDLI DL_CertGetNext  (CSSM_DL_HANDLE DLHandle,
                                       CSSM_DB_HANDLE DBHandle,
                                       CSSM_HANDLE ResultsHandle)
```

This function returns the next certificate matching the selection criteria used to establish the ResultsHandle.

Parameters

DLHandle (input)
The handle that describes the add-in data storage library module to be used to perform this function.

DBHandle (input)
The handle that describes the data store to be used when performing this function.

ResultsHandle (input)
The selection handle obtained from the DL_CertGetFirst function call.

Return Value

Returns a pointer to a CSSM_DATA structure which contains the next certificate in the data store that matches the selection criteria. If the pointer is NULL, an error has occurred. Use CSSM_GetError to obtain the error code.

Error Codes

Value	Description
CSSM_DL_INVALID_DL_HANDLE	Invalid DL handle
CSSM_DL_INVALID_RESULTS_HANDLE	Invalid query handle
CSSM_DL_INVALID_DB_HANDLE	Invalid DB handle
CSSM_DL_NO_MORE_CERTS	No more certificates for that selection handle
CSSM_DL_CERT_GETNEXT_FAIL	Opening the records caused an exception
CSSM_DL_MEMORY_ERROR	Error in allocating memory

See Also

DL_CertGetFirst, DL_CertAbortQuery.

2.5.6 DL_CertAbortQuery

```
CSSM_RETURN CSSMDLI DL_CertAbortQuery (CSSM_DL_HANDLE DLHandle,
                                       CSSM_DB_HANDLE DBHandle,
                                       CSSM_HANDLE ResultsHandle)
```

This function terminates the query initiated by DL_CertGetFirst and allows a DL to release all intermediate state information associated with the query.

Parameters

DLHandle (input)
The handle that describes the add-in data storage library module used to perform this function.

DBHandle (input)
The handle that describes the data store associated with the query.

ResultsHandle (input)
The selection handle returned from the DL_CertGetFirst function.

Return Value

CSSM_OK if the function was successful. CSSM_FAIL if an error condition occurred. Use CSSM_GetError to obtain the error code.

Error Codes

Value	Description
CSSM_DL_INVALID_DL_HANDLE	Invalid data storage library Handle
CSSM_DL_INVALID_DB_HANDLE	Invalid data store handle
CSSM_DL_INVALID_RESULTS_HANDLE	Invalid results handle
CSSM_DL_CERT_ABORT_QUERY_FAIL	Unable to abort query

See Also

DL_CertGetFirst, DL_CertGetNext.

2.6 CRL Storage Operations

2.6.1 DL_CrlInsert

```
CSSM_RETURN CSSMDLI DL_CrlInsert  (CSSM_DL_HANDLE DLHandle,
                                   CSSM_DB_HANDLE DBHandle,
                                   CSSM_CL_HANDLE CLHandle,
                                   const CSSM_DATA_PTR Crl)
```

This function makes the CRL persistent by inserting it into the specified data store.

Parameters

DLHandle (input)
The handle that describes the add-in data store library module to be used to perform this function.

DBHandle (input)
The handle that describes the data store to be used when performing this function.

CLHandle (input)
The handle that describes the add-in certificate library module to be used to perform this function.

Crl (input)
A pointer to the CSSM_DATA structure which contains the CRL to be added to the data store.

Return Value

A CSSM_OK return value signifies that the function completed successfully. When CSSM_FAIL is returned, an error has occurred. Use CSSM_GetError to obtain the error code.

Error Codes

Value	Description
CSSM_DL_INVALID_DL_HANDLE	Invalid DL handle
CSSM_DL_INVALID_CRL_PTR	Invalid CRL pointer
CSSM_DL_INVALID_DB_HANDLE	Invalid DB handle
CSSM_DL_CRL_INSERT_FAIL	Add caused an exception

See Also

DL_CrlDelete

2.6.2 DL_CrlDelete

```
CSSM_RETURN CSSMDLI DL_CrlDelete  (CSSM_DL_HANDLE DLHandle,
                                   CSSM_DB_HANDLE DBHandle,
                                   CSSM_CL_HANDLE CLHandle,
                                   const CSSM_DATA_PTR Crl)
```

This function removes the CRL from the specified data store.

Parameters

DLHandle (input)
The handle that describes the add-in data store library module to be used to perform this function.

DBHandle (input)
The handle that describes the data store to be used when performing this function.

CLHandle (input)
The handle that describes the add-in certificate library module to be used to perform this function.

Crl (input)
A pointer to the CSSM_DATA structure which contains the CRL to be removed from the data store.

Return Value

A CSSM_OK return value signifies that the function completed successfully. When CSSM_FAIL is returned, an error has occurred. Use CSSM_GetError to obtain the error code.

Error Codes

Value	Description
CSSM_DL_INVALID_DL_HANDLE	Invalid DL handle
CSSM_DL_INVALID_CRL_PTR	Invalid CRL pointer
CSSM_DL_INVALID_DB_HANDLE	Invalid DB handle
CSSM_DL_CRL_NOT_IN_DB	CRL not in DB
CSSM_DL_CRL_DELETE_FAIL	Delete caused an exception

See Also

DL_CrlInsert.

2.6.3 DL_CrlGetFirst

CSSM_DATA_PTR CSSMDLI DL_CrlGetFirst

```
(CSSM_DL_HANDLE DLHandle,
CSSM_DB_HANDLE DBHandle,
CSSM_SELECTION_PREDICATE_PTR SelectionPredicate,
uint32 SizeSelectionPredicate,
CSSM_DB_CONJUNCTIVE Conjunctive,
CSSM_HANDLE_PTR ResultsHandle,
uint32 *NumberOfMatchedCrls)
```

This function locates the first CRL in the data store which matches the selection criteria. The selection criteria is the expression formed by connecting all of the relational expressions of the selection predicate array using the one conjunctive operator. This function returns a count of the total number of CRLs matching the selection criteria, the first CRL matching the criteria, and a selection handle that may be used to retrieve the subsequent CRLs matching the selection criteria.

Parameters

DLHandle (input)
The handle that describes the add-in data storage library module to be used to perform this function.

DBHandle (input)
The handle that describes the data store to be used when performing this function.

SelectionPredicate (input)
A pointer to a CSSM_SELECTION_PREDICATE array which contains field-value/operator pairs. If NULL, the first CRL in the data store is returned.

SizeSelectionPredicate(input)
The size of the selection predicate array.

Conjunctive (input)
The Boolean operator used to connect the selection predicates. If the selection predicate is null, this parameter is ignored.

ResultsHandle (output)
This handle should be used for subsequent retrievals for the same selection criteria.

NumberOfMatchedCrls (output)
Returns the total number of CRLs that match the selection criteria.

Return Value

Returns a pointer to a CSSM_DATA structure which contains the first CRL in the data store that matches the selection criteria. If the pointer is NULL, an error has occurred. Use CSSM_GetError to obtain the error code.

Error Codes

Value	Description
CSSM_DL_INVALID_DL_HANDLE	Invalid DL handle
CSSM_DL_INVALID_SELECTION_PTR	Invalid selection predicate pointer
CSSM_DL_INVALID_DB_HANDLE	Invalid DB handle
CSSM_DL_NO_CRL_FOUND	No Crls that match the selection predicate
CSSM_DL_CRL_GET_FIRST_FAIL	Get first caused an exception
CSSM_DL_MEMORY_ERROR	Error in allocating memory

See Also

DL_CrlGetNext, DL_CrlAbortQuery.

2.6.4 DL_CrlGetNext

CSSM_DATA_PTR CSSMDLI DL_CrlGetNext (CSSM_DL_HANDLE DLHandle,
CSSM_DB_HANDLE DBHandle,
CSSM_HANDLE ResultsHandle)

This function returns the next certificate which matches the selection criteria used to establish the ResultsHandle.

Parameters

DLHandle (input)
The handle that describes the add-in data storage library module to be used to perform this function.

DBHandle (input)
The handle that describes the data store to be used when performing this function.

ResultsHandle (*input)*
The selection handle obtained from the DL_CrlGetFirst function call.

Return Value

Returns a pointer to a CSSM_DATA structure which contains the next CRL in the data store that matches the selection criteria. If the pointer is NULL, an error has occurred. Use CSSM_GetError to obtain the error code.

Error Codes

Value	Description
CSSM_DL_INVALID_DL_HANDLE	Invalid DL handle
CSSM_DL_INVALID_RESULTS_HANDLE	Invalid query handle
CSSM_DL_INVALID_DB_HANDLE	Invalid DB handle
CSSM_DL_NO_MORE_CRLS	No more Crls for that selection handle
CSSM_DL_CRL_GET_NEXT_FAIL	Opening the records caused an exception
CSSM_DL_MEMORY_ERROR	Error in allocating memory

See Also

DL_CrlGetFirst, DL_CrlAbortQuery.

2.6.5 DL_CrlAbortQuery

```
CSSM_RETURN CSSMDLI DL_CrlAbortQuery  (CSSM_DL_HANDLE DLHandle,
                                       CSSM_DB_HANDLE DBHandle,
                                       CSSM_HANDLE ResultsHandle)
```

This function terminates the query initiated by DL_CrlGetFirst and allows a DL to release all intermediate state information associated with the query.

Parameters

DLHandle (input)
The handle that describes the add-in data storage library module used to perform this function.

DBHandle (input)
The handle that describes the data store associated with the query.

ResultsHandle (input)
The selection handle returned from the DL_CrlGetFirst function.

Return Value

CSSM_OK if the function was successful. CSSM_FAIL if an error condition occurred. Use CSSM_GetError to obtain the error code.

Error Codes

Value	Description
CSSM_DL_INVALID_DL_HANDLE	Invalid data storage library Handle
CSSM_DL_INVALID_DB_HANDLE	Invalid data store handle
CSSM_DL_INVALID_RESULTS_HANDLE	Invalid results handle
CSSM_DL_CRL_ABORT_QUERY_FAIL	Unable to abort query

See Also

DL_CrlGetFirst, DL_CrlGetNext.

2.7 Module Management Functions

2.7.1 CSSM_DL_Install

```
CSSM_RETURN CSSMAPI CSSM_DL_Install
                                    (const char *DLName,
                                    const char *DLFileName,
                                    const char *DLPathName,
                                    const CSSM_GUID_PTR GUID,
                                    const CSSM_DLINFO_PTR DLInfo,
                                    const void *Reserved1,
                                    const CSSM_DATA_PTR Reserved2)
```

This function updates the CSSM persistent internal information about a DL module.

Parameters

DLName (input)
The name of the data storage library module to be installed.

DLFileName (input)
The name of the file that contains the data storage library implementation.

DLPathName (input)
The path to the file that implements the data storage library.

GUID (input)
A pointer to the CSSM_DATA structure containing the global unique identifier for a DL module.

DLInfo (input)
A pointer to the CSSM_DLINFO structure containing information about a DL module.

Reserved1
Reserved data for the function.

Reserved2
Reserved data for the function.

Return Value

A CSSM_OK return value signifies that information has been updated. When CSSM_FAIL is returned, an error has occurred. Use CSSM_GetError to obtain the error code.

Error Codes

Value	Description
CSSM_INVALID_POINTER	Invalid pointer
CSSM_REGISTRY_ERROR	Error in writing registry

See Also

CSSM_DL_Uninstall.

2.7.2 CSSM_DL_Uninstall

CSSM_RETURN CSSMAPI CSSM_DL_Uninstall (const CSSM_GUID_PTR GUID)

This function deletes the CSSM persistent internal information about a DL module.

Parameters

GUID (input)
A pointer to the CSSM_DATA structure containing the global unique identifier for a DL module.

Return Value

A CSSM_TRUE return value signifies that information has been updated. When CSSM_FALSE is returned, an error has occurred. Use CSSM_GetError to obtain the error code.

Error Codes

Value	Description
CSSM_INVALID_POINTER	Invalid pointer
CSSM_REGISTRY_ERROR	Error in writing registry

See Also

CSSM_DL_Install.

2.7.3 CSSM_DL_ListModules

CSSM_LIST_PTR CSSMAPI CSSM_DL_ListModules (void)

This function returns a list containing the GUID/name pair for each of the currently-installed DL modules.

Parameters

None

Return Value

A pointer to the CSSM_LIST structure containing the names of DL modules. If the pointer is NULL, an error has occurred. Use CSSM_GetError to obtain the error code.

Error Codes

Value	Description
CSSM_MEMORY_ERROR	Error in memory allocation

See Also

CSSM_FreeList.

2.7.4 CSSM_DL_Attach

```
CSSM_DL_HANDLE CSSMAPI CSSM_DL_Attach
                        (const CSSM_GUID_PTR GUID,
                        uint32 CheckCompatibleVerMajor,
                        uint32 CheckCompatibleVerMinor,
                        const CSSM_API_MEMORY_FUNCS_PTR MemoryFuncs,
                        const uint32 DeviceID,
                        const uint32 DeviceAccessFlags,
                        uint32 Application,
                        const CSSM_NOTIFY_CALLBACK Notification,
                        const void *Reserved)
```

This function attaches the application with a DL module. A DL module tests for compatibility with the version specified.

Parameters

GUID (input)
A pointer to the CSSM_DATA structure containing the global unique identifier for a DL module.

CheckCompatibleVerMajor (input)
The major version number of a DL module that the application is compatible with.

CheckCompatibleVerMinor (input)
The minor version number of a DL module that the application is compatible with.

MemoryFuncs (input)
The caller’s memory allocation and deallocation functions that can be jointly used by a DL and the caller to manage a common memory pool.

DeviceID (input)
Device ID number of the target hardware storage device. This value should always be taken from the CSSM_DLINFO structure to insure that a compatible identifier is used. (Software-only implementations can always use zero for this value.)

DeviceAccessFlags(input)
Bitmask of default access modes. Legal values are defined in the table below. DL service providers may or may not support all combinations of access modes.

Application(input/optional)
Passed to the application when its callback is invoked, allowing the application to determine the proper context of operation.

Notification (input/optional)
Callback provided by the application that is called by a DL when one of three things takes place: a parallel operation completes, a token running in serial mode surrenders control to the application, or the token is removed (hardware specific).

Reserved
A reserved input.

Valid *DeviceAccessFlags* Values

Value	Description
CSSM_DL_STORE_ACCESS_SERIAL	All storage accesses are in serial mode
CSSM_ DL_STORE _ACCESS_EXCLUSIVE	Storage access is exclusive to this caller

Return Value

A handle is returned for a DL module. If the handle is NULL, an error has occurred. Use CSSM_GetError to obtain the error code.

Error Codes

Value	Description
CSSM_INVALID_POINTER	Invalid pointer
CSSM_MEMORY_ERROR	Internal memory error
CSSM_INCOMPATIBLE_VERSION	Incompatible version
CSSM_ATTACH_FAIL	Unable to attach to DL module

See Also

CSSM_DL_Detach.

2.7.5 CSSM_DL_Detach

CSSM_RETURN CSSMAPI CSSM_DL_Detach (CSSM_DL_HANDLE DLHandle)

This function detaches the application from a DL module.

Parameters

DLHandle (input)
The handle that describes the add-in data storage library module to be detached.

Return Value

A CSSM_OK return value signifies that a DL module has been detached. When CSSM_FAIL is returned, an error has occurred. Use CSSM_GetError to obtain the error code.

Error Codes

Value	Description
CSSM_INVALID_ADDIN_HANDLE	Invalid DL handle

See Also

CSSM_DL_Attach.

2.7.6 CSSM_DL_GetInfo

CSSM_DLINFO_PTR CSSMAPI CSSM_DL_GetInfo

```
                                        (const CSSM_GUID_PTR GUID,
                                        CSSM_BOOL CompleteCapabilitiesOnly,
                                        uint32 *NumberOfInfos)
```

This function returns information describing a DL module and its capabilities.

Parameters

GUID (input)
A pointer to the CSSM_DATA structure containing the global unique identifier for a DL module.

CompleteCapabilitiesOnly (input)
Boolean value indicating whether or not partially-specified capabilities should be returned. If set to TRUE only a completely-specified capability should be returned. If set to false, the registered structure should be returned regardless of the completeness of its current state.

NumberOfInfos (output)
The number of DLinfo structures returned by the execution of this function.

Return Value

A pointer to an array of one or more CSSM_DLINFO structures containing information about a DL module. There is one Dlinfo structure for each distinct hardware storage device managed by this DL. Most DLs manage only one device. If the pointer is NULL, an error has occurred. Use CSSM_GetError to obtain the error code.

Error Codes

Value	Description
CSSM_INVALID_POINTER	Invalid pointer
CSSM_MEMORY_ERROR	Internal memory error
CSSM_INVALID_GUID	No known DL module with specified GUID

See Also

CSSM_DL_FreeInfo.

2.7.7 CSSM_DL_FreeInfo

CSSM_RETURN CSSMAPI CSSM_DL_FreeInfo (CSSM_DLINFO_PTR DLInfo,
uint32 NumberOfInfos)

This function frees the memory allocated by a DL module for the CSSM_DLINFO structure returned by the CSSM_DL_GetInfo function.

Parameters

DLInfo (input)
A pointer to CSSM_DL_Info structure.

numberOfInfos (input)
The number of DL Info structures to be freed.

Return Value

A CSSM_OK return value signifies that the function completed successfully. When CSSM_FAIL is returned, an error has occurred. Use CSSM_GetError to obtain the error code.

Error Codes

Value	Description
CSSM_ INVALID_DLINFO_ POINTER	Invalid CSSM_DLINFO pointer

See Also

CSSM_DL_GetInfo.

2.7.8 CSSM_DL_GetDbNames

CSSM_NAME_LIST_PTR CSSMAPI CSSM_DL_GetDbNames
(CSSM_DL_HANDLE DLHandle)

This function returns a list of the logical data store names that the specified DL module can access and a count of the number of logical names in that list.

Parameters

DLHandle (input)
The handle that describes the add-in data storage library module to be used to perform this function.

Return Value

Returns a pointer to a CSSM_NAME_LIST structure which contains a list of data store names. If the pointer is NULL, an error has occurred. Use CSSM_GetError to obtain the error code.

Error Codes

Value	Description
CSSM_DL_MEMORY_ERROR	Error allocating memory
CSSM_DL_NO_DATA_SOURCES	No known data store names
CSSM_DL_GET_DB_NAMES_FAIL	Get DB Names failed
CSSM_DL_INVALID_DL_HANDLE	Invalid DL Handle

See Also

CSSM_DL_FreeNameList.

2.7.9 CSSM_DL_FreeNameList

CSSM_RETURN CSSMAPI CSSM_DL_FreeNameList

```
(CSSM_DL_HANDLE DLHandle,
CSSM_NAME_LIST_PTR NameList)
```

This function frees the list of the logical data store names that was returned by DL_GetDbNames ( ).

Parameters

DLHandle (input)
The handle that describes the add-in data storage library module to be used to perform this function.

NameList (input)
A pointer to the CSSM_NAME_LIST.

Return Value

CSSM_OK if the function was successful. CSSM_FAIL if an error condition occurred. Use CSSM_GetError to obtain the error code.

Error Codes

Value	Description
CSSM_DL_MEMORY_ERROR	Error allocating memory
CSSM_DL_INVALID_PTR	Invalid pointer to the name list
CSSM_DL_INVALID_DL_HANDLE	Invalid DL Handle

See Also

CSSM_DL_GetDbNames.

2.8 Extensibility Functions

2.8.1 DL_PassThrough

CSSM_DATA_PTR CSSMDLI DL_PassThrough

```
(CSSM_DL_HANDLE DLHandle,
CSSM_DB_HANDLE DBHandle,
uint32 PassThroughId,
const CSSM_DATA_PTR InputParams)
```

This function allows applications to call data storage library module-specific operations that have been exported. Such operations may include queries or services that are specific to the domain represented by a DL module.

Parameters

DLHandle (input)
The handle that describes the add-in data storage library module to be used to perform this function.

DBHandle (input)
The handle that describes the data store to be used when performing this function.

PassThroughId (input)
An identifier assigned by a DL module to indicate the exported function to perform.

InputParams (input)
A pointer to the CSSM_DATA structure containing parameters to be interpreted in a function-specific manner by the requested DL module. This parameter can be used as a pointer to an array of CSSM_DATA_PTRs.

Return Value

A pointer to the CSSM_DATA structure containing the output from the pass-through function. The output data must be interpreted by the calling application based on externally available information. If the pointer is NULL, an error has occurred. Use CSSM_GetError to obtain the error code.

Error Codes

Value	Description
CSSM_DL_INVALID_DL_HANDLE	Invalid DL handle
CSSM_DL_INVALID_DB_HANDLE	Invalid DB handle
CSSM_DL_INVALID_PASSTHROUGH_ID	Invalid passthrough ID
CSSM_DL_INVALID_ PTR	Invalid pointer
CSSM_DL_ PASS_THROUGH_FAIL	DB exception doing passthrough function
CSSM_DL_MEMORY_ERROR	Error in allocating memory

2.8.2 DL_Initialize

```
CSSM_RETURN CSSMDLI DL_Initialize
            (uint32 VerMajor,
 uint32 VerMinor)
```

This function checks whether the current version of a DL module is compatible with the input version and performs any module-specific setup activities. Memory management calls are also passed to a DL through this call.

Parameters

VerMajor (input)
The major version number of a DL module expected by the calling application.

VerMinor (input)
The minor version number of a DL module expected by the calling application.

UpcallTable (output)
A structure containing pointers to the memory routines to be used by a DL module to allocate and free memory owned by the calling application.

Return Value

A CSSM_OK return value signifies that the current version of a DL module is compatible with the input version numbers and all setup operations were successfully performed. When CSSM_FAIL is returned, either the current DL module is incompatible with the requested DL module version or an error has occurred. Use CSSM_GetError to obtain the error code.

Error Codes

Value	Description
CSSM_DL_INITIALIZE_FAIL	Unable to perform module initialization

See Also

DL_Uninitialize.

2.8.3 DL_Uninitialize

CSSM_RETURN CSSMDLI DL_Uninitialize (void)

This function performs any module-specific cleanup activities.

Parameters

None

Return Value

A CSSM_OK return value signifies that all cleanup operations were successfully performed. When CSSM_FAIL is returned, an error has occurred. Use CSSM_GetError to obtain the error code.

Error Codes

Value	Description
CSSM_DL_UNINITIALIZE_FAIL	Unable to perform module cleanup

See Also

DL_Initialize.

2.9 Extensibility Functions

The DL_PassThrough function is provided to allow DL developers to extend the certificate and CRL format-specific storage functionality of the CSSM API. Because it is exposed to CSSM as only a function pointer, its name internal to the data storage library can be assigned at the discretion of a DL module developer. However, its parameter list and return value must match what is shown below. The error codes listed in this section are the generic codes all data storage libraries may use to describe common error conditions. Data storage library developers may also define their own module-specific error codes, as described in Section 3.5.2.

2.9.1 DL_PassThrough

```
CSSM_DATA_PTR CSSMDLI DL_PassThrough  (CSSM_DL_HANDLE DLHandle,
                                       CSSM_DB_HANDLE DBHandle,
                                       uint32 PassThroughId,
                                       const CSSM_DATA_PTR InputParams)
```

This function allows applications to call additional module-specific operations that have been exported by the data storage library. Such operations may include queries or services specific to the domain represented by a DL module.

Parameters

DLHandle(input)
The handle that describes the add-in data storage library module to be used to perform this function.

DBHandle(input)
The handle that describes the data storage to be used when performing this function.

PassThroughId (input)
An identifier assigned by a DL module to indicate the exported function to perform.

InputParams(input)
A pointer to the CSSM_DATA structure containing parameters to be interpreted in a function-specific manner by the requested DL module.

Return Value

A pointer to the CSSM_DATA structure containing the output from the pass-through function. The output data must be interpreted by the calling application based on externally-available information and documentation. If the pointer is NULL, an error has occurred. Use CSSM_GetError to obtain the error code.

Error Codes

Value	Description
CSSM_DL_INVALID_DL_HANDLE	Invalid DL handle
CSSM_DL_INVALID_DB_HANDLE	Invalid DB handle
CSSM_INVALID_PASSTHROUGH_ID	Invalid passthrough Id
CSSM_INVALID_DATA_PTR	Invalid pointer

CSSM_DL_ PASSTHROUGH_FAIL	DB exception doing passthrough function
CSSM_DL_MEMORY_ERROR	Error in allocating memory

3. Data Storage Library Structure and Management

3.1 Data Storage Library Composition

A data storage library is a dynamically-linkable library, which is composed of functions that implement some or all of the CSSM DLI described in Section 2. A DL must also contain one or more functions that are called when a DL is attached and detached. Within the data storage library, the attach function is responsible for registering a function table with CSSM, accepting the memory management upcalls, and performing any module-specific setup. The detach function is responsible for any cleanup the module requires. The remaining functions implement a subset of the DLI, as determined by a DL developer.

The data storage library composition can be broadly classified into the following categories:

- Module management
- Memory management
- Data store management functions
  - Opening and closing data stores
  - Creating and deleting data stores
  - Importing and exporting data stores
- Persistence operations on data objects
  - Adding new data objects
  - Deleting data objects
  - Retrieving data objects
- Persistence operations on certificates and certificate revocation lists
  - Adding new certificates and new certificate revocation records
  - Deleting certificates and certificate revocation records
  - Retrieving certificates and certificate revocation records
- Pass-through for unique, module-specific operations

3.2 Data Storage Library Installation

Before an application can use a data storage library, its name and other descriptive information must be registered with CSSM by an installation application. The name given to a data storage library module is both a logical name and a globally-unique identifier (GUID). The logical name is a string chosen by the data storage library developer to describe a DL module. The GUID is a structure used to differentiate between library modules in the CSSM registry. GUIDs are discussed in more detail below. The location of a DL module is required at installation time so the CSSM can locate the module when an application requests an attach.

3.2.1 Global Unique Identifiers (GUIDs)

Each data storage library must have a globally-unique identifier (GUID) that the CSSM, applications, and DL modules use to uniquely identify a DL. The DL GUID is used by the CSSM registry to expose add-in module availability to applications. A DL module uses its GUID to identify itself when it sets an error. When attaching the library, the application uses the DL GUID to identify the requested DL module.

A GUID is defined as:

```
typedef struct cssm_guid {
    uint32 Data1;
    uint16 Data2;
    uint16 Data3;
    uint8 Data4[8];
} CSSM_GUID, *CSSM_GUID_PTR;
```

GUID generators are publicly available for Windows* 95, Windows NT*, and on many UNIX* platforms.

3.2.2 Data Storage characteristics

The version number of each data storage library is registered with CSSM during module installation. The application may query a module's version number by querying the CSSM module registry.

Applications can use the version number of a data storage library to determine the compatibility of the installed DL with its required DL. If the compatible versions are unknown, the application can pass the required version number to a DL at attach time. A DL will check for compatibility before attaching. If the versions are not compatible, the attach request fails.

3.2.3 Object Identifiers (OIDs)

Object Identifiers (OIDs) are used to reference specific data types or data structures within a given certificate or CRL format. OIDs are defined by a certificate library developer at a granularity appropriate for the expected usage of the CL.

To support search operations on fields within a certificate or CRL, a data storage library recognizes the OIDs defined by a related certificate library. DL developers may define OIDs in addition to or in place of those defined by a certificate library.

3.3 Attaching a Data Storage Library

Before an application can use the functions of a specific DL, it must attach a DL to CSSM using the *CSSM_DL_Attach* function. On attach, the data storage library uses the *CSSM_DL_RegisterServices* function to register its function table with CSSM. CSSM uses a DL module's function table to direct calls from the application to the correct function in the data storage library module. During the attach process, a DL's *DL_Initialize* function is called. At this time version compatibility is confirmed and a table of memory function upcalls is passed to a DL. A DL module uses the memory management upcalls to allocate any memory that will be returned to the calling application and to free any memory that it received from the calling application.

When CSSM attaches to or detaches from a data storage library module, it initiates a function in a DL that performs the necessary setup and cleanup operations. The attach and detach functions will vary depending on the target operating system for the data storage library module. For example, DLLMain would be used to implement these functions in a DL targeted to Windows NT*. _init and _fini would be used to implement these functions in a DL targeted to SunOS.

3.3.1 The DL module function table

The function table for a data storage library module is a structure that contains pointers to a DL module's implementation of the functions specified in the data storage library Interface. This structure is specified as a part of the CSSM header file, cssm.h. If a DL does not support some function in the DLI, the pointer to that function should be set to NULL.

3.3.2 Memory management upcalls

All memory allocation and de-allocation for data passed between the application and a DL module via CSSM is ultimately the responsibility of the calling application. Since a DL module needs to allocate memory to return data to the application, the application must provide a DL module a means of allocating memory that the application has the ability to free. It does this by providing a DL module with memory management upcalls.

Memory management upcalls are pointers to the memory management functions used by the calling application. They are provided to a DL module via CSSM as a structure of function pointers. The functions will be the calling application's equivalent of malloc, free and re-alloc and will be expected to have the same behavior as those functions. The function parameters will consist of the normal parameters for that function. The function return values should be interpreted in the standard manner. A DL module is responsible for making the memory management functions available to all of its internal functions.

3.4 Data Storage Library Basic Services

3.4.1 Function Implementation

A data storage library developer may choose to implement some or all of the functions specified in the DLI. Section 2 of this document details the expected behavior of each function.

A data storage library developer may choose to leverage the capabilities of another DL module to implement certain functions. To do this, a DL would attach to another DL using *CSSM_DL_Attach* Subsequent function calls to the first DL call the corresponding function in the second DL for some or all of its implementation.

3.4.2 Error handling

When an error occurs, the function in a DL module should call the *CSSM_SetError* function. The *CSSM_SetError* function takes the module's GUID and an error number as inputs. The module's GUID is used to identify where the error occurred. The error number will be used to describe the error.

The error number set by a DL module should fall into one of two ranges. The first range of error numbers is pre-defined by CSSM. These are errors that are common to all DL modules implementing a given function. They are described in this document as part of the function definitions in Sections 2.3 through 2.7. They are defined in the header file cssmerr.h, which is distributed as part of CSSM. The second range of error numbers is used to define module-specific error codes. These module-specific error codes should be in the range of CSSM_DL_PRIVATE_ERROR to CSSM_DL_END_ERROR. CSSM_DL_PRIVATE_ERROR and CSSM_DL_END_ERROR are also defined in the header file cssmerr.h. A DL module developer is responsible for making the definition and interpretation of their module-specific error codes available to applications.

When no error has occurred, but the appropriate return value from a function is CSSM_FALSE, that function should call *CSSM_ClearError* before returning. When the application receives a CSSM_FALSE return value, it is responsible for checking whether an error has occurred by calling *CSSM_GetError*. If the function in a DL module has called *CSSM_ClearError*, the calling application receives a CSSM_OK response from the *CSSM_GetError* function, indicating no error has occurred.

3.5 Data Storage Utility Libraries

Data storage utility libraries are software components that may be provided by a data storage library developer for use by other data storage library developers. They are expected to contain functions that may be useful to several data storage library modules. The data storage utility library developer is responsible for making the definition, interpretation, and usage of their library available to other DL module developers.

3.6 Attach/Detach Example

The data storage library module is responsible for performing certain operations when CSSM attaches to and detaches from it. DL modules that have been developed for Windows-based systems use the DllMain routine to perform those operations, as shown in the example below.

3.6.1 DLLMain

```
#include “cssm.h”
CSSM_GUID dl_guid =
{ 0x5fc43dc1, 0x732, 0x11d0, { 0xbb, 0x14, 0x0, 0xaa, 0x0, 0x36, 0x67, 0x2d }
};
CSSM_FUNCTIONTABLE FunctionTable;
CSSM_SPI_MEMORY_FUNCS DLemoryFunctions;

BOOL WINAPI DllMain ( HANDLE hInstance, DWORD dwReason, LPVOID lpReserved)
{
   switch (dwReason)
   {
      case DLL_PROCESS_ATTACH:
      {

         /* Fill in FunctionTable with function pointers */
         FunctionTable.DbOpen = DL_DbOpen;
         FunctionTable.DbClose = DL_DbClose;
         FunctionTable.DbCreate = DL_DbCreate;
         FunctionTable.DbDelete = DL_DbDelete;
         FunctionTable.CertInsert = DL_DataInsert;
         FunctionTable.CertDelete = DL_DataDelete;
         FunctionTable.DataGetFirst = DL_DataGetFirst;
         FunctionTable.DataGetNext = DL_DataGetNext;
         FunctionTable.DataAbortQuery = DL_DataAbortQuery;
         FunctionTable.CertInsert = DL_CertInsert;
         FunctionTable.CertDelete = DL_CertDelete;
         FunctionTable.CertRevoke = DL_CertRevoke;
         FunctionTable.CertGetFirst = DL_CertGetFirst;
         FunctionTable.CertGetNext = DL_CertGetNext;
         FunctionTable.CertAbortQuery = DL_CertAbortQuery;
         FunctionTable.CrlInsert = DL_CrlInsert;
         FunctionTable.CrlDelete = DL_CrlDelete;
         FunctionTable.CrlGetFirst = DL_CrlGetFirst;
         FunctionTable.CrlGetNext = DL_CrlGetNext;
         FunctionTable.CrlAbortQuery = DL_CrlAbortQuery;
         FunctionTable.DbImport = DL_DbImport;
         FunctionTable.DbExport = DL_DbExport;
         FunctionTable.DLGetDbNames = DL_GetDbNames;
         FunctionTable.DLFreeNameList = DL_FreeNameList;
         FunctionTable.PassThrough = DL_PassThrough;
         FunctionTable.Initialize = DL_Initialize;
         FunctionTable.Uninitialize = DL_Uninitialize;

         /* Call CSSM_DL_RegisterServices to register the FunctionTable */
         /* with CSSM and to receive the application’s memory upcall table */
        if (CSSM_DL_RegisterServices (&dl_guid, &FunctionTable,
                                        & DLemoryFunctions,NULL) != CSSM_OK)
            return FALSE;

         /* Make the upcall table available to all functions in this library
         */

         break;
      }
   case DLL_THREAD_ATTACH:
      break;
   case DLL_THREAD_DETACH:
      break;
```

```
    case DLL_PROCESS_DETACH:
        if (CSSM_DL_DeregisterServices (&dl_guid) != CSSM_OK)
            return FALSE;
        break;
    }
return TRUE;
}
```

3.7 Data source Operations Examples

This section contains a template for the DL_DbOpen function.

```
/*------------------------------------------------------------------------
 * Name: DL_DbOpen
 *
 * Description:
 * This function opens a Data store and returns a handle back to the
 * caller which should be used for further access to the data store.
 *
 * Parameters:
 * DLHandle(input)        : Handle identifying a a DL module.
 * DbName                 : Handle identifying the opened Data store.
 *
 * Return value:
 * Handle to the Opened Data store.
 * If NULL, use CSSM_GetError to get the follwing return codes
 *
 * Error Codes:
 * CSSM_DL_INVALID_DL_HANDLE
 * CSSM_DL_DATASTORE_NOT_EXISTS
 * CSSM_DL_MEMORY_ERROR
 * CSSM_DL_DB_OPEN_FAIL
 * CSSM_DL_MEMORY_ERROR
 * CSSM_DL_INVALID_DATASTORE_NAME
 *
 *
 *
*------------------------------------------------------------------------*/

CSSM_DB_HANDLE CSSMDLI DL_DbOpen(CSSM_DL_HANDLE DLHandle,
                                    const char *DbName)
{
        if(DLHandle == NULL)
        {
                CSSM_SetError(&dl_guid, CSSM_DL_INVALID_DL_HANDLE);
                return NULL;
        }
        if(DbName == NULL)
        {
                CSSM_SetError(&dl_guid, CSSM_DL_INVALID_DATASTORE_NAME);
                return NULL;
        }
        if(!dl_IfDataStoreExists(DLHandle, DbName))
        {
                CSSM_SetError(&dl_guid, CSSM_DL_DATASTORE_NOT_EXISTS);
                return NULL;
        }
        /*DL specific internal implementation of DbOpen*/
        CSSM_DB_Handle Handle = dl_OpenDataStore(DbName);
        return Handle;
}
```

3.8 Certificate Storage Operations Examples

This section contains a template for the DL_CertInsert function.

```
/*-------------------------------------------------------------------------
 * Name:  DL_CertInsert
 *
 * Description:
 * This function stores the Certificate into the data storage specified by
 * the DbHandle
 *
 * Parameters:
 * DLHandle(input)       : Handle identifying  a DL module.
 * DbHandle(input)       : Handle identifying an opened DbModule.
 * Cert                  : Certificate data pointer to be added.
 *
 * Return value:
 * CSSM_OK if success.CSSM_FAIL, if failed
 * Use CSSM_GetError to get the following return codes
 *
 * Error Codes:
 * CSSM_DL_INVALID_DL_HANDLE
 * CSSM_DL_INVALID_DB_HANDLE
 * CSSM_DL_INVALID_CERTIFICATE_PTR
 * CSSM_DL_CERT_INSERT_FAIL
 * CSSM_DL_MEMORY_ERROR
 *
 *
*-------------------------------------------------------------------------*/

CSSM_RETURN CSSMDLI DL_CertInsert (CSSM_DL_HANDLE DLHandle,
                                    CSSM_DB_HANDLE DBHandle,
                                   CSSM_CL_HANDLE CLHandle,
                                   const CSSM_DATA_PTR Cert)
{
        CSSM_RETURN ret = CSSM_OK;
        if(DLHandle == NULL)
        {
                CSSM_SetError(&dl_guid, CSSM_DL_INVALID_DL_HANDLE);
                return CSSM_FAIL;
        }
        if(DBHandle == NULL)
        {
                CSSM_SetError(&dl_guid, CSSM_DL_INVALID_DB_HANDLE);
                return CSSM_FAIL;
        }
        if(CLHandle == NULL)
        {
                CSSM_SetError(&dl_guid, CSSM_DL_INVALID_Cl_HANDLE);
                return CSSM_FAIL;
        }

        if( (Cert == NULL) && cssm_IsBadReadPtr(Cert->Data, Cert->Length) )
        {
                CSSM_SetError(&dl_guid, CSSM_DL_INVALID_CERTIFICATE_PTR);
                return CSSM_FAIL;
        }
        /*Do the DL specific Certificate storage here*/
        return CSSM_OK;
}
```

4. Appendix A. Relevant CSSM API functions

4.1 Overview

Several API functions are particularly relevant to data storage library developers, because they are used either by the application to access a DL module or by a DL module to access CSSM services, such as the CSSM registry or the error-handling routines. They are included in this appendix for quick-reference by DL module developers. For additional information, a DL module developer is encouraged to reference the *CSSM Application Programming Interface*

4.2 Data Structures

4.2.1 CSSM_DATA

The CSSM_DATA structure is used to associate a length, in bytes, with an arbitrary block of contiguous memory. This memory must be allocated and freed using the memory management routines provided by the calling application via CSSM.

```
typedef struct cssm_data {
    uint32 Length;
    uint8  Data[0];
} CSSM_DATA, *CSSM_DATA_PTR
```

Definition:

Length - The length, in bytes, of the memory block pointed to by *Data*.

Data - A byte array of size 0. This is a place holder for a contiguous block of memory.

4.2.2 CSSM_OID

The object identifier (OID) is used to identify the data types and data structures of a certificate or CRL.

```
typedef CSSM_VALUE CSSM_OID
```

4.2.3 CSSM_GUID

A GUID is a globally-unique identifier used to uniquely identify a DL.

```
typedef struct cssm_guid {
    uint32 Data1;
    uint16 Data2;
    uint16 Data3;
    uint8 Data4[8];
} CSSM_GUID, *CSSM_GUID_PTR;
```

4.2.4 CSSM_DL_INFO

This structure contains all of the static data associated with a data storage library add-in module. This information is added to the CSSM registry at install time. It can be queried using the command **CSSM_DL_GetInfo ( )**

```
typedef struct cssm_dlinfo{
    uint32 VerMajor;
    uint32 VerMinor;
    CSSM_DL_MODULE_TYPE DLModuleType;
    uint32 DeviceID;
    CSSM_BOOL CapabilitiesInitialized;
    uint32 DeviceAccessFlags;
    CSSM_DATA ExclusiveDLMCertificate;
    CSSM_BOOL LoginRequired;
    uint32 NumberOfRecordTypes;
    CSSM_DATA_RECORD_TYPE_PTR DataRecordTypes;
    uint32 NumberOfAttributeUsageTypes;
    CSSM_DB_ATTRIBUTE_USAGE_PTR AttributeUsageTypes;
    uint32 NumberOfAttributeIdFormats;
    CSSM_ATTRIBUTE_ID_FORMAT_PTR AttributeIdFormats;
    uint32 NumberOfRelOperatorTypes;
    CSSM_DB_OPERATOR_PTR RelOperatorTypes;
    uint32 NumberOfConjOperatorTypes;
    CSSM_DB_CONJUNCTIVE_PTR ConjOperatorTypes;
    CSSM_DATA_PTR Reserved1;
}CSSM_DLINFO, *CSSM_DLINFO_PTR
```

Definition:

VerMajor - The major version number of the add-in module.

VerMinor - The minor version number of the add-in module.

DLModuleType - Indicates the underlying implementation approach for this library module.

DeviceID - The ID of a hardware storage device managed by this data storage library module.

CapabilitiesInitialized - True or false, indicating whether complete capabilities are currently specified in this DLinfo structure.

DeviceAccessFlags - A bitmask of the device access modes supported by this library module.

ExclusiveDLMCertificate - The certificate used to sign certificates issued to exclusive users of this data storage library module.

LoginRequired - True or false, indicating whether a DL requires caller login and logout.

NumberOfRecordTypes - The number of distinct data record types that can be stored and managed by this library module in one or more data stores.

DataRecordTypes - An array listing the data record types that can be stored and managed by this library module in one or more data stores. The array contains *NumberOfRecordTypes* entries.

NumberOfAttributeUsageTypes - The number of distinct attribute usages supported by a DL.

AttributeUsageTypes - An array listing the attribute usages supported by the data storage library when defining the schema for a new data store. Currently-defined usages include: indexed attribute, unique-index attribute, and non-indexed attribute. The array contains *NumberOfAttributeUsageTypes* entries.

NumberOfAttributeIdFormats - The number of distinct attribute identification formats that are supported by this data storage library module.

AttributeIdFormats - An array listing the formats accepted by the data storage library to identify record fields in a selection query. Currently-defined usages include: OID-name format and string-name format. The array contains *NumberOfAttributeIdFormats* entries.

NumberOfRelOperatorTypes - The number of distinct binary relational operators supported by this library module.

RelOperatorTypes - An array listing the relational operators supported by this library module for defining selection predicates for retrieving stored data objects. The array contains *NumberOfRelOperatorTypes* entries.

NumberOfConjOperatorTypes - The number of distinct conjunctive operators supported by this library module.

ConjOperatorTypes - An array listing the conjunctive operators supported by this library module for defining selection predicates for retrieving stored data objects. The array contains *NumberOfConjOperatorTypes* entries.

Reserved1 - Reserved for future use.

4.2.5 CSSM_HANDLE

Handle used to identify the caller.

```
typedef uint32 CSSM_HANDLE, *CSSM_HANDLE_PTR;
```

4.2.6 CSSM_API_MEMORY_FUNCS Data Structure

This structure is used by applications to supply memory functions for the CSSM and the add-ins modules. The functions are used when memory needs to be allocated by the CSSM or add-ins for returning data structures to the applications.

```
typedef struct cssm_api_memory_funcs {
    void * (*malloc_func) (uint32 size, void *allocRef);
    void (*free_func) (void *memblock, void *allocRef);
    void * (*realloc_func) (void *memblock, uint32 size, void *allocRef);
    void * (*calloc_func) (uint32 num, uint32 size, void *allocRef);
} CSSM_API_MEMORY_FUNCS, *CSSM_API_MEMORY_FUNCS_PTR
```

Definition:

malloc_func - pointer to function that returns a void pointer to the allocated memory block of atleast *size* bytes from heap allocRef

free_func - pointer to function that deallocates a previously-allocated memory block (*memblock*) from heap allocRef

realloc_func - pointer to function that returns a void pointer to the reallocated memory block (*memblock*) of at least *size* bytes from heap allocRef

calloc_func - pointer to function that returns a void pointer to an array of *num* elements of length *size* initialized to zero from heap allocRef

4.3 Function Definitions

4.3.1 CSSM_DL_Install

CSSM_BOOL CSSMAPI CSSM_DL_Install (const char *DLName,
const char *DLFileName,
const char *DLPathName,
const CSSM_GUID_PTR GUID,
const CSSM_DLINFO_PTR DLInfo,
const void *Reserved1,
const CSSM_DATA_PTR Reserved2)

This function updates the CSSM-persistent internal information about a DL module.

Parameters

DLName(input)
The name of the data storage library module to be installed.

DLFileName(input)
The name of the file that implements the data storage library.

DLPathName(input)
The path to the file that implements the data storage library.

GUID (input)
A pointer to the CSSM_DATA structure containing the global unique identifier for a DL module.

DLInfo(input)
A pointer to the CSSM_DLINFO structure containing information about a DL module.

Reserved1
Reserved data for the function.

Reserved2
Reserved data for the function.

Return Value

A CSSM_TRUE return value signifies that information has been updated. When CSSM_FALSE is returned, an error has occurred. Use CSSM_GetError to obtain the error code.

Error Codes

Value	Description
CSSM_DL_INVALID_DATA_POINTER	Invalid pointer
CSSM_DL_INVALID_DLINFO_POINTER	Invalid pointer
CSSM_DL_INVALID_POINTER	Invalid pointer
CSSM_DL_INSTALL_FAIL	Unable to update internal information

See Also

CSSM_DL_Uninstall.

4.3.2 CSSM_DL_Uninstall

CSSM_BOOL CSSMAPI CSSM_DL_Uninstall (const CSSM_GUID_PTR GUID)

This function deletes the CSSM persistent internal information about a DL module.

Parameters

GUID (input)
A pointer to the CSSM_DATA structure containing the global unique identifier for a DL module.

Return Value

A CSSM_TRUE return value signifies that information has been updated. When CSSM_FALSE is returned, an error has occurred. Use CSSM_GetError to obtain the error code.

Error Codes

Value	Description
CSSM_DL_INVALID_DATA_POINTER	Invalid pointer
CSSM_DL_INVALID_ GUID	DL module was not installed
CSSM_DL_UNINSTALL_FAIL	Unable to delete information

See Also

CSSM_DL_Install.

4.3.3 CSSM_DL_ListModules

CSSM_LIST_PTR CSSMAPI CSSM_DL_ListModules (void)

This function returns a list containing the GUID/name pair for each of the currently-installed DL modules.

Parameters

None

Return Value

A pointer to the CSSM_LIST structure containing the names of DL modules. If the pointer is NULL, an error has occurred. Use CSSM_GetError to obtain the error code.

Error Codes

Value	Description
CSSM_DL_MEMORY_ERROR	Error in memory allocation
CSSM_DL_LIST_MODULES_FAIL	Unable to find DL modules

See Also

CSSM_FreeList.

4.3.4 CSSM_FreeList

CSSM_RETURN CSSMAPI CSSM_FreeList (CSSM_LIST_PTR CSSMList)

This function frees the memory allocated to hold a list of strings.

Parameters

CSSMList (input)
A pointer to the CSSM_LIST structure containing the GUID, name pair of add-ins.

Return Value

CSSM_OK if the function was successful. CSSM_FAIL if an error condition occurred. Use CSSM_GetError to obtain the error code.

Error Codes

Value	Description
CSSM_INVALID_POINTER	Invalid pointer input

See Also

CSSM_DL_ListModules.

4.3.5 CSSM_DL_Attach

CSSM_DL_HANDLE CSSMAPI CSSM_DL_Attach

```
                    (const CSSM_GUID_PTR GUID,
                    uint32 CheckCompatibleVerMajor,
                    uint32 CheckCompatibleVerMinor,
                    const CSSM_API_MEMORY_FUNCS_PTR MemoryFuncs,
                    const uint32 DeviceID,
                    const uint32 DeviceAccessFlags,
                    uint32 Application,
                    const CSSM_NOTIFY_CALLBACK Notification,
                    const void *Reserved)
```

This function attaches the application with a DL module. A DL module tests for compatibility with the version specified.

Parameters

GUID (input)
A pointer to the CSSM_DATA structure containing the global unique identifier for a DL module.

CheckCompatibleVerMajor (input)
The major version number of a DL module that the application is compatible with.

CheckCompatibleVerMinor (input)
The minor version number of a DL module that the application is compatible with.

MemoryFuncs (input)
The caller’s memory allocation and deallocation functions that can be jointly used by a DL and the caller to manage a common memory pool.

DeviceID (input)
Device ID number of the target hardware storage device. Should value should always be taken from the CSSM_DLINFO structure to insure that a compatible identifier is used. (Software-only implementations can always use zero for this value.)

DeviceAccessFlags(input)
Bitmask of default access modes. Legal values are defined in the table below. DL service providers may or may not support all combinations of access modes.

Application(input/optional)
Nonce passed to the application when its callback is invoked allowing the application to determine the proper context of operation.

Notification (input/optional)
Callback provided by the application that is called by a DL when one of three things takes place: a parallel operation completes, a token running in serial mode surrenders control to the application or the token is removed (hardware specific).

Reserved
A reserved input.

Valid *DeviceAccessFlags* Values

Value	Description
CSSM_DL_STORE_ACCESS_SERIAL	All storage accesses are in serial mode
CSSM_ DL_STORE _ACCESS_EXCLUSIVE	Storage access is exclusive to this caller

Return Value

A handle is returned for a DL module. If the handle is NULL, an error has occurred. Use CSSM_GetError to obtain the error code.

Error Codes

Value	Description
CSSM_INVALID_POINTER	Invalid pointer
CSSM_MEMORY_ERROR	Internal memory error
CSSM_INCOMPATIBLE_VERSION	Incompatible version
CSSM_ATTACH_FAIL	Unable to attach to DL module

See Also

CSSM_DL_Detach.

4.3.6 CSSM_DL_Detach

CSSM_BOOL CSSMAPI CSSM_DL_Detach (CSSM_DL_HANDLE DLHandle)

This function detaches the application from a DL module.

Parameters

DLHandle (input)
The handle that describes the add-in data storage library module to be detached.

Return Value

A CSSM_TRUE return value signifies that a DL module has been detached. If CSSM_FALSE is returned, an error has occurred. Use CSSM_GetError to obtain the error code.

Error Codes

Value	Description
CSSM_DL_INVALID_DL_HANDLE	Invalid handle
CSSM_DL_DETACH_FAIL	Unable to detach from DL module

See Also

CSSM_DL_Attach.

4.3.7 CSSM_DL_GetInfo

CSSM_DLINFO_PTR CSSMAPI CSSM_DL_GetInfo (const CSSM_GUID_PTR GUID)

This function returns the CSSM registry describing a DL module.

Parameters

GUID (input)
A pointer to the CSSM_DATA structure containing the global unique identifier for a DL module.

Return Value

A pointer to the CSSM_DLINFO structure containing information about a DL module. If the pointer is NULL, an error has occurred. Use CSSM_GetError to obtain the error code.

Error Codes

Value	Description
CSSM_DL_INVALID_DATA_POINTER	Invalid pointer
CSSM_DL_INVALID_GUID	Can’t find DL
CSSM_DL_MEMORY_ERROR	Error allocating memory
CSSM_DL_GET_INFO_FAIL	Unable to retrieve DL module information

See Also

CSSM_DL_FreeInfo.

4.3.8 CSSM_DL_FreeInfo

CSSM_RETURN CSSMAPI CSSM_DL_FreeInfo (CSSM_DLINFO_PTR DLInfo)

This function frees the memory allocated by CSSM for the CSSM_DL_INFO structure and returned by the CSSM_DL_GetInfo function.

Parameters

DLInfo(input)
A pointer to the CSSM_DL_Info structure.

Return Value

A CSSM_OK return value signifies the function completed successfully. When CSSM_FAIL is returned, an error has occurred. Use CSSM_GetError to obtain the error code.

Error Codes

Value	Description
CSSM_DL_FREE_INFO_FAIL	Failed to free the DLINFO structure
CSSM_DL_INVALID_DLINFO_PTR	Invalid CSSM_DL_INFO pointer

See Also

CSSM_DL_GetInfo.

4.3.9 CSSM_DL_RegisterServices

CSSM_RETURN CSSMAPI CSSM_DL_RegisterServices

```
                                        (const CSSM_GUID_PTR GUID,
                                        CSSM_FUNCTIONTABLE FunctionTable,
                                        CSSM_SPI_MEMORY_FUNCS_PTR
                                        MemoryFuncs,
                                        void *Reserved)
```

This function is used by a data storage library module to register its function table with CSSM and to receive a memory management upcall table from CSSM.

Parameters

GUID (input)
A pointer to the CSSM_GUID structure containing the global unique identifier for a DL module.

FunctionTable (input)
A structure containing pointers to the data storage library Interface functions implemented by a DL module.

MemoryFuncs(output)
A pointer to SPI_MEMORY_FUNCS which gets filled up when the function returns.
Reserved(input)
A reserved input.

Return Value

CSSM_OK if the function was successful. CSSM_FAIL if an error condition occurred. Use CSSM_GetError to obtain the error code.

Error Codes

Value	Description
CSSM_INVALID_GUID	Invalid GUID
CSSM_INVALID_FUNCTION_TABLE	Invalid function table
CSSM_REGISTER_SERVICES_FAIL	Unable to register services

See Also

CSSM_DL_DeRegisterServices.

4.3.10 CSSM_DL_DeregisterServices

CSSM_RETURN CSSMAPI CSSM_DL_DeregisterServices (const CSSM_GUID_PTR GUID)

This function is used by a data storage library module to de-register its function table with CSSM.

Parameters

GUID (input)
A pointer to the CSSM_GUID structure containing the global unique identifier for a DL module.

Return Value

CSSM_OK if the function was successful. CSSM_FAIL if an error condition occurred. Use CSSM_GetError to obtain the error code.

Error Codes

Value	Description
CSSM_INVALID_GUID	Invalid GUID
CSSM_DEREGISTER_SERVICES_FAIL	Unable to deregister services

See Also

CSSM_DL_RegisterServices.

4.3.11 CSSM_GetError

CSSM_ERROR_PTR CSSMAPI CSSM_GetError (void)

This function returns the current error information.

Parameters

None

Return Value

Returns the current error information. If there is currently no valid error, the error number will be CSSM_OK. A NULL pointer indicates that the CSSM_InitError was not called by the CSSM Core or that a call to CSSM_DestoryError has been made by the CSSM Core. No error information is available.

See Also

CSSM_ClearError, CSSM_SetError.

4.3.12 CSSM_SetError

CSSM_RETURN CSSMAPI CSSM_SetError (CSSM_GUID_PTR guid,
uint32 error_number)

This function sets the current error information to *error_number* and *guid*.

Parameters

guid (input)
Pointer to the GUID (global unique ID) of the add-in module.

error_number (input)
An error number. It should fall within one of the valid CSSM, CL, TP, DL, or CSP error ranges.

Return Value

CSSM_OK if error was successfully set. A return value of CSSM_FAIL indicates that the error number passed is not within a valid range, the GUID passed is invalid, CSSM_InitError was not called by the CSSM Core, or CSSM_DestroyError has been called by the CSSM Core. No error information is available.

See Also

CSSM_ClearError, CSSM_GetError.

4.3.13 CSSM_ClearError

void CSSMAPI CSSM_ClearError (void)

This function sets the current error value to CSSM_OK. This can be called if the current error value has been handled and therefore is no longer a valid error.

Parameters

None

See Also

CSSM_SetError, CSSM_GetError.