Common Security Services Manager

Data Storage Library Interface (DLI) Specification

Release 1.0
October 1996
Updated December 1996

Subject to Change Without Notice

Specification Disclaimer and Limited Use License

This specification is for release version 1.0, October 1996.

THIS SPECIFICATION IS PROVIDED “AS IS” WITH NO WARRANTIES WHATSOEVER, INCLUDING ANY WARRANTY OF MERCHANTABILITY, FITNESS FOR ANY PARTICULAR PURPOSE, OR ANY WARRANTY OTHERWISE ARISING OUT OF ANY PROPOSAL, SPECIFICATION OR SAMPLE.

Some aspects of this Specification may be covered under various United States or foreign patents. No license, express or implied, by estoppel or otherwise, to any other intellectual property rights is granted herein.

Intel disclaims all liability, including liability for infringement of any proprietary rights, relating to implementation of information in this specification. Intel doesn't warrant or represent that such implementation(s) will not infringe such rights.

If you are interested in receiving an appropriate license to Intel's intellectual property rights relating to the interface defined in this specification, contact us for details at cdsa@ibeam.intel.com.

Intel Corporation, 5200 N.E. Elam Young Parkway, Hillsboro, OR 97124-6497

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.

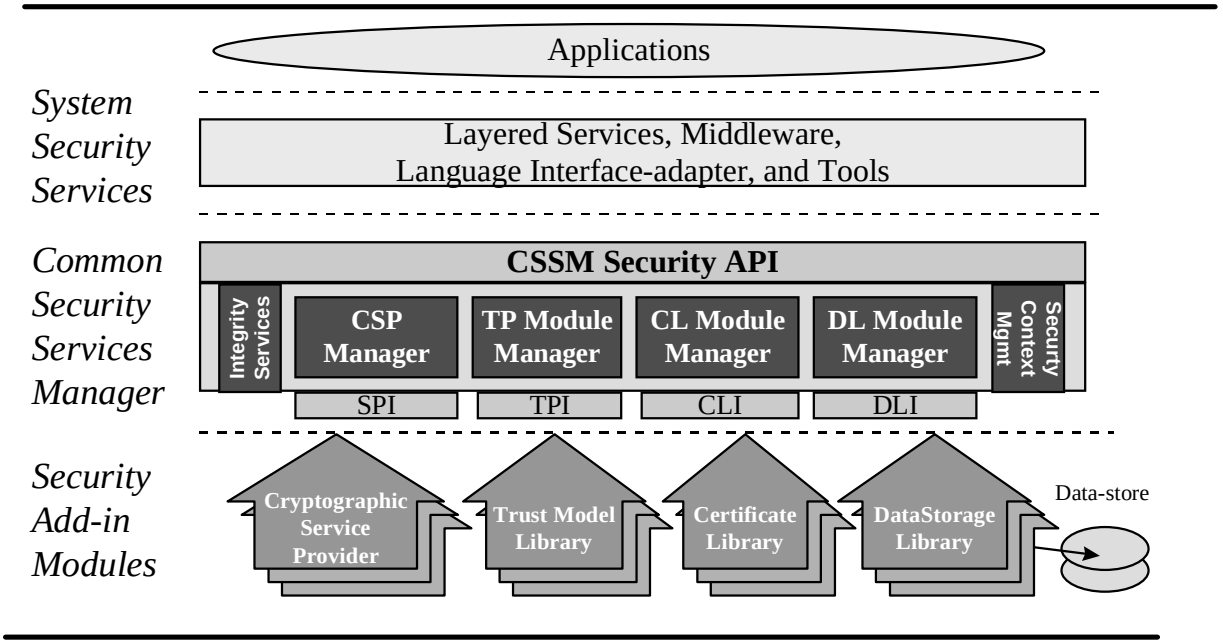

Figure 1. The Common Data Security Architecture for all platforms

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 and certificate revocation lists (CRLs). Stable storage could be provided by a

- Commercially-available database management system product
- Native file system
- Custom hardware-based storage devices

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, the DL operation should not be responsible for checking whether the certificate values or CRL values conform to 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 and CRLs, such as performance-enhancing retrieval functions.

1.2.1 Application Interaction

When a new DL is installed on a system, information specific to the DL is stored in the system or CSSM registry. An application uses that information to find an appropriate DL and to request that CSSM attach to the DL. When CSSM attaches to the DL, it returns a DL handle to the application that uniquely identifies the pairing of the application thread to the 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 the DL's function table. The function table consists of pointers to the data storage functions from the CSSM API, which are supported by the DL. When an application causes CSSM to attach the DL, the data storage library registers its function table with CSSM using *CSSM_DL_RegisterServices*. During future function calls from the application, CSSM will use these function pointers to direct the call 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 the DL when it calls *CSSM_DL_RegisterServices*. These functions must be used whenever a data storage library allocates or de-allocates memory that belongs to or will belong to the application.

1.2.2 DL Structure and Use

A data storage library is composed of functions, which are invoked when the DL is attached and detached and ones that mirror the CSSM API for data storage operations. When the DL is attached, it registers its function table with the CSSM registry and CSSM provides the DL with a set of upcalls for management of the application's memory. When the DL is detached, any necessary cleanup actions are performed. The remainder of the data storage library functions perform basic storage and retrieval operations on 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 the 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 libraries 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. We assume 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 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 data storage library Interface (DLI) specifies the functions that a data storage library makes available to applications via CSSM for storing and retrieving certificates and certificate revocation lists (CRLs). These functions mirror the CSSM API. They include the basic areas of functionality expected of a data storage library: data-source management operations, certificate storage operations, certificate revocation list storage operations, extensibility functions, and module management functions. The data storage library developer may choose to implement some or all of the DLI functions. The available functions are registered with CSSM at attach time in the form of a DL function table. In the function table, all non-supported functions are represented by a NULL function pointer. The data storage library module developer is responsible for making its general functionality known to application developers.

The data storage library Services API defines two categories of operations:

- Data-source management functions
- Persistence operations on certificates and certificate revocation lists

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

- Opening and closing data stores. The 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. The 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 the 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 certificates only, certificate revocation records only, or both. It is unusual for a DL module to manage a data store containing both certificates and certificate revocation records, but there is nothing in the CSSM or the DL module API that prevents a DL module from implementing persistence in this manner. Typically, separate physical data stores will be used to store certificates and CRLs.

The persistence operations on data stores include:

- Adding or updating 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 the DL module. Existing certificates may be updated, but certificate revocation records should only be added and deleted.
- Deleting certificates and certificate revocation records. The DL module removes a single certificate or a single certificate revocation record from the data store. The operation fails if the record is not found in 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. The DL module must provide a search mechanism for selectively retrieving a copy of these persistent objects. Selection is based on a selection criterion, such as a unique certificate ID, date of certificate issue, name of the agent who signed a revocation record and/or other parameters.

Module management functions include:

- Check module version compatibility. A data storage library module must provide support for the DL_CheckVersion operation. CSSM uses this operation to verify that the attached DL module version is compatible with the 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 the DL and the *CSSM_DL_Attach* operation will fail.

Extensibility of data store operations is achieved via:

- 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

DL_DbOpen ( ) - This function opens the data- store with the specified logical name. Before accessing a data store for certificate or CRL operations, a caller must open the data store.

DL_DbClose ( ) - This function closes a previously opened data store.

DL_DbCreate ( ) - This function creates a new, empty data store with the specified logical name. A DL may implement this function by opening the data store if it already exists or creating the data store if it does not exist. The DL may also create the data store schema as part of the implementation of this function. The data store should be opened after this operation.

DL_DbDelete ( ) - This function deletes all records from the specified data store and removes current state information associated with it.

DL_DbImport ( ) - This function imports all records from an exported copy of a certificate or CRL data store into the local system. The certificates or CRLs contained in the file must be in the native format of the DL module. The DL module imports all certificates or CRL records in the file, creating a new certificate data store or CRL data store, respectively. This mechanism can be used to copy data stores among systems or to restore a persistent data store of certificates or CRLs from a backup copy.

DL_DbExport ( ) - This function exports a copy of the records from the data store. This mechanism can be used to copy data stores among systems or to create a data store backup.

2.1.2 Certificate Storage Operations

DL_CertRevoke ( ) - This function makes the revocation of a specified certificate persistent. The revocation may be recorded by marking the certificate record in the persistent data store as revoked, by adding a record to a persistent CRL, or both. 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.

DL_CertInsert ( ) - This function stores the certificate in the data store. This may or may not include the creation of index entries. The mechanisms used to store and retrieve persistent certificates are private to the implementation of the Data Storage Library.

DL_CertDelete ( ) - This function removes a certificate from the data store. If the certificate is not found, the operation fails.

DL_CertGetFirst ( ) - This function retrieves from a data store the first certificate that matches the selection criteria. The selection criterion is the expression formed by connecting relational operators using a conjunctive operator. “Greater than”, “less than”, “equal to”, and “not equal to” are examples of relational operators. “Boolean and” and “Boolean or” are examples of conjunctive operators. The implementation of the data storage library can specify which relational expressions and conjunctive operations it supports. If multiple certificate records match the selection criteria, a selection handle is returned to the caller. The selection handle is used by DL_CertGetNext to retrieve additional records. The caller may terminate the query by calling DL_CertAbortQuery. A data storage library may limit the number of concurrently-managed selection handles. The library developer must document all restrictions and application developers should be aware of these restrictions.

DL_CertGetNext ( ) - This function retrieves the next certificate that matches the selection criteria. If all certificates have already been returned from the set specified by the selection criteria, the function returns a NULL pointer. A data storage library may limit the number of concurrently-managed

selection handles. The library developer must document all restrictions, and application developers should be aware of these restrictions.

DL_CertAbortQuery ( ) - This function cancels the query initiated by DL_CertGetFirst and resets the selection handle.

2.1.3 CRL Storage Operations

DL_CrlInsert ( ) - This function saves the CRL in the data store. This may or may not include the creation of index entries. The mechanisms used to store and retrieve persistent CRL records are private to the implementation of the data storage library.

DL_CrlDelete ( ) - This function deletes a CRL record from the data store. If the record is not found in the specified data store, the operation fails.

DL_CrlGetFirst ( ) - This function retrieves from the data store a CRL record that matches the selection criteria. The selection criterion is the expression formed by connecting relational operators and conjunctive operator. “Greater than”, “less than”, “equal to”, and “not equal to” are examples of relational operators. “Boolean and” and “Boolean or” are examples of conjunctive operators. The implementation of the data storage library can specify which of the relational expressions and conjunctive operations it supports. If multiple certificate records match the selection criteria, a selection handle is returned to the caller. The selection handle is used by DL_CrlGetNext function to retrieve additional records. The caller may terminate the query at any time by calling DL_CrlAbortQuery. A data storage library may limit the number of concurrently-managed selection handles. The library developer must document all restrictions, and application developers should be aware of these restrictions.

DL_CrlGetNext ( ) - This function retrieves the next CRL record that matches the selection criteria. If all CRLs have already been returned from the set specified by selection criteria, the function returns a NULL CRL record. A data storage library may limit the number of concurrently-managed selection handles. The library developer must document all restrictions, and application developers should be aware of these restrictions.

DL_CrlAbortQuery ( ) - This function cancels the query initiated by DL_CrlGetFirst and resets the selection handle.

2.1.4 Module Management Functions

DL_GetDbNames ( ) - This function returns a list of the logical data store names that this module can access and a count of the number of logical names in that list.

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

DL_CheckVersion ( ) - This function checks whether the version of the attached DL module is compatible with the input version number. It is called by the CSSM when attaching the DL. If the versions are incompatible, CSSM detaches the DL module.

2.1.5 Extensibility Functions

DL_PassThrough ( ) - This function performs the DL specific function indicated by the operation ID. The operation ID may specify any type of operation the 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, CRLs or to the relationships between the certificates and CRLs manipulated by the DL module.

2.2 Data Structures

This section describes the data structures that may be passed to or returned from a data storage library function. Applications use these data structures to prepare and then pass input parameters into CSSM API function calls, which are passed without modification to the appropriate DL. The DL is responsible for interpreting them and returning the appropriate data structure to the calling application via CSSM. These data structures are defined in the header file, cssm.h, distributed with CSSM.

```
typedef uint32 CSSM_DL_HANDLE    /* data storage library Handle */
typedef uint32 CSSM_DB_HANDLE    /* data store Handle */
```

2.2.1 CSSM_DB_CONJUNCTIVE

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

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

2.2.2 CSSM_DB_OPERATOR

These are the logical operators that 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
```

2.2.3 CSSM_FIELD

This structure contains the tag/value pair for a single field of a certificate or CRL.

```
typedef struct cssm_field {
    CSSM_OID FieldOid;
    CSSM_VALUE FieldValue;
}CSSM_FIELD, *CSSM_FIELD_PTR
```

Definition:

FieldOid - The object identifier that uniquely identifies this certificate or CRL field.

FieldValue - The value of this certificate field.

2.2.4 CSSM_SELECTION_PREDICATE

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

```
typedef struct cssm_selection_predicate {
    CSSM_FIELD Field;
    CSSM_DB_OPERATOR DbOperator;
} CSSM_SELECTION_PREDICATE, *CSSM_SELECTION_PREDICATE_PTR
```

Definition:

Field - The object identifier that uniquely identifies the search field in a data store record.

dbOperator - The operator to be used when comparing the *Field* to the data store record.

2.2.5 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. Acts as a placeholder for a contiguous block of memory.

2.2.6 CSSM_NAME_LIST

The CSSM_NAME_LIST structure is used to return the logical names of the data stores that a DL module can access.

```
typedef struct cssm_name_list {
    uint32 NumStrings;
    char** String;
} CSSM_NAME_LIST, *CSSM_NAME_LIST_PTR;
```

Defintion:

NumStrings - Number of strings in the array pointed to by *String.*

String - A pointer to an array of strings.

2.3 Data source Operations

This section describes the function prototypes and error codes defined for the data-source operations in the DLI. The functions are exposed to CSSM via a function table, so the function names may vary at the discretion of the data storage library developer. However, the function parameter list and return type must match the prototypes given in this section in order to be used by applications. The error codes listed in this section are the generic codes defined by CSSM for use by all data storage libraries in describing common error conditions. A data storage library developer may define additional module-specific error codes, as described in Section 3.5.2.

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_MEMORY_ERROR	Not enough memory
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_RETURN CSSMDLI DL_DbCreate (CSSM_DL_HANDLE DLHandle,
CSSM_CL_HANDLE CLHandle,
const char *DbName)

This function creates a new, empty 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.

CLHandle(input)
The handle that describes the add-in certificate 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_CREATE_FAIL	Create caused an exception
CSSM_DL_MEMORY_ERROR	Error in allocating memory

See Also

DL_DbOpen, DL_DbClose

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 it.

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, to create a new data store. The data store records must be in the DL module's native format.

Parameters

DLHandle(input)
The handle that describes the add-in data store 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 data store 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_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 store 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 that will contain a copy of the source data store 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

2.4 Certificate Storage Operations

This section describes the function prototypes and error codes defined for the persistent storage operations in the DLI. The functions will be exposed to CSSM via a function table, so the function names may vary at the discretion of the data storage library developer. However, the function parameter list and return type must match the prototypes given in this section in order to be used by applications. The error codes listed in this section are the generic codes defined by CSSM for use by all data storage libraries in describing common error conditions. A data storage library developer may also define their own module-specific error codes, as described in Section 3.5.2.

2.4.1 DL_CertInsert

CSSM_RETURN CSSMDLI DL_CertInsert (CSSM_DL_HANDLE DLHandle,
CSSM_DB_HANDLE DBHandle,
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.

Cert (input)
A pointer to the CSSM_DATA structure that 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

2.4.2 DL_CertDelete

CSSM_RETURN CSSMDLI DL_CertDelete (CSSM_DL_HANDLE DLHandle,
CSSM_DB_HANDLE DBHandle,
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.

Cert (input)
A pointer to the CSSM_DATA structure that 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

2.4.3 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 that contains the certificate to be 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

See Also

DL_CertInsert, DL_CertDelete

2.4.4 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 that matches the selection criteria. The selection criterion is the expression formed by connecting all of the relational expressions of the selection predicate array using the 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 and relational operator pairs. If the pointer is 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 certificate 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

2.4.5 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 returned from the DL_CertGetFirst function.

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_SELECTION_PTR	Invalid certificate pointer
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

2.4.6 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 the 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 to be used when performing this function.

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_RESULTS_HANDLE	Invalid results handle
CSSM_DL_CRL_ABORT_QUERY_FAIL	Unable to abort query

See Also

DL_CertGetFirst, DL_CertGetNext

2.5 CRL Storage Operations

This section describes the function prototypes and error codes defined for the persistent storage operations in the DLI. The functions are exposed to CSSM via a function table, so the function names may vary at the discretion of the data storage library developer. However, the function parameter list and return type must match the prototypes given in this section in order to be used by applications. The error codes listed in this section are the generic codes defined by CSSM for use by all data storage libraries in describing common error conditions. A data storage library developer may also define their own module-specific error codes, as described in Section 3.5.2.

2.5.1 DL_CrlInsert

CSSM_RETURN CSSMDLI DL_CrlInsert (CSSM_DL_HANDLE DLHandle,
CSSM_DB_HANDLE DBHandle,
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 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.

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

Return Value

A CSSM_OK return value signifies that the function completed successfully. If 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 certificate pointer
CSSM_DL_INVALID_DB_HANDLE	Invalid DB handle
CSSM_DL_CRL_INSERT_FAIL	Add caused an exception

See Also

DL_CrlDelete

2.5.2 DL_CrlDelete

CSSM_RETURN CSSMDLI DL_CrlDelete (CSSM_DL_HANDLE DLHandle,
CSSM_DB_HANDLE DBHandle,
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 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.

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

Return Value

A CSSM_OK return value signifies that the function completed successfully. If 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 certificate 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

2.5.3 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 that matches the selection criteria. The selection criterion is the expression formed by connecting all of the relational expressions of the selection predicate array using the 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 that contains field and relational 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 certificate 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

2.5.4 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 that 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 returned from the DL_CrlGetFirst function.

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_SELECTION_PTR	Invalid certificate pointer
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

2.5.5 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 the 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 to be used when performing this function.

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_RESULTS_HANDLE	Invalid results handle
CSSM_DL_CRL_ABORT_QUERY_FAIL	Unable to abort query

See Also

DL_CrlGetFirst, DL_CrlGetNext

2.6 Module Management Functions

This section describes the function prototypes and error codes defined for module management in the DLI. The functions are exposed to CSSM via a function table, so the function name can be assigned at the discretion of the DL module developer. However, the parameter list and return value must match what is shown below. The error codes listed in this section are the generic codes that all data storage libraries 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.

2.6.1 DL_GetDbNames

CSSM_NAME_LIST_PTR CSSMDLI 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_GET_DB_NAMES_FAIL	Get DB Names failed
CSSM_DL_NO_DATASOURCES	No Data sources found
CSSM_DL_INVALID_DL_HANDLE	Invalid DL Handle

See Also

DL_FreeNameList

2.6.2 DL_FreeNameList

CSSM_RETURN CSSMDLI DL_FreeNameList (CSSM_DL_HANDLE DLHandle,
CSSM_NAME_LIST_PTR NameList)

This function frees the list of logical data store names 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_DL_HANDLE	Invalid DL Handle

See Also

DL_GetDbNames

DL_Initialize

CSSM_RETURN CSSMDLI DL_Initialize (uint32 VerMajor,
uint32 VerMinor)

This function checks whether the current version of the DL module is compatible with the input version and performs any module-specific setup activities.

Parameters

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

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

Return Value

A CSSM_OK return value signifies that the current version of the 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

2.6.3 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

2.7 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 the 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.

2.7.1 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 the 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 the 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. The DL must also contain one or more functions that are called when the 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 the 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 certificates and certificate revocation lists
  - Adding new certificates and new certificate revocation records
  - Updating existing certificates
  - 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 the 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 the 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. The 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.

Applicatioons 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 the DL at attach time. The 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 the 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 and to obtain the application's memory management upcalls from CSSM. CSSM uses the DL module's function table to direct calls from the application to the correct function in the data storage library module. The 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 the 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 the 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 the DL module via CSSM is ultimately the responsibility of the calling application. Since the DL module needs to allocate memory to return data to the application, the application must provide the DL module a means of allocating memory that the application has the ability to free. It does this by providing the 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 the 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 a DL handle followed by the normal parameters for that function. The DL handle is used by CSSM to direct the memory operation to the target application. The function return values should be interpreted in the standard manner. The 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, the 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 the 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 the 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. The 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 the 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_FUNC_TBL_PTR UpcallTable;

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_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, &UpcallTable)
         != 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 the 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  the 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,
                                  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( (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 the DL module or by the 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, the 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 DL registry at install time. It can be queried using the command **CSSM_DL_GetInfo ( )**.

```
typedef struct cssm_dlinfo{
    uint32 VerMajor;
    uint32 VerMinor;
    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.

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_SPI_FUNC_TBL

This data structure contains function pointers to the calling application's memory management routines. The DL module uses these routines to allocate and free any memory that belongs to or will belong to the application.

```
typedef struct cssm_spi_func_tbl {
    void *(*malloc_func) (CSSM_HANDLE, uint32);
    void (*free_func) (CSSM_HANDLE, void *);
    void  *(*realloc_func) (CSSM_HANDLE, void *, uint32);
} CSSM_SPI_MEMORY_FUNCS, *CSSM_SPI_MEMORY_FUNCS_PTR;
```

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 the 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 the DL module.

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

Parameters

GUID (input)
A pointer to the CSSM_DATA structure containing the global unique identifier for the 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 void *Reserved)

This function attaches the application with the DL module specified by the GUID. The 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 the DL module.

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

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

Reserved
A reserved input.

Return Value

A handle is returned for the DL module. If the handle 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	Invalid DL GUID
CSSM_DL_INCOMPATIBLE_VERSION	Incompatible version
CSSM_DL_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 the 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 the 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 the DL module.

Parameters

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

Return Value

A pointer to the CSSM_DLINFO structure containing information about the 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_FUNC_TBL_PTR *UpcallTable,
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 the DL module.

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

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

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 the 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 CSSMDLI 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 CSSMDLI 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 CSSMDLI 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