Internet Engineering Task Force	D. Lanz
Internet-Draft	L. Novikov
Intended status: Informational	MITRE
Expires: January 26, 2012	July 25, 2011

Common Interface to Cryptographic Modules (CICM) Module Management
draft-lanz-cicm-mm-01

Abstract

[RFC Editor: Please update the RFC references prior to publication.]
This memo defines a programming interface for high-level management of cryptographic modules as outlined in draft-lanz-cicm-model-00 and required by draft-lanz-cicm-02 including managing the module authentication, software, logs, built-in tests, and responding to module events.
Comments are solicited and should be addressed to the mailing list at cicm@ietf.org.

Status of this Memo

Copyright Notice

Table of Contents

1. Introduction

This document defines the module management functions for the Common Interface to Cryptographic Modules (CICM) as defined in [CICM]. The underlying logical model and terminology are defined in [CICM-LM].

1.1. Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

1.2. Definition Language

This document uses the Interface Definition Language (IDL) [IDL] to specify language-neutral interfaces and is not intended to prescribe or preclude a particular communications protocol such as General Inter-ORB Protocol (GIOP) [CORBA] between programs in different address spaces or on different devices. See Definition Language in [CICM] for more information.

1.3. Conformance and Extension Language

This document contains definitions for several opaque data parameters whose format is not defined by CICM. Instead, implementers are required to create an Implementation Conformance Statement which MUST reference a standard format or define a module developer-specific format implemented by the module for these datatypes. See Conformance and Extensions in [CICM] for more details.

2. CICM Dependencies

This document depends on type definitions and interfaces that are defined in other CICM documents.

2.1. Namespaces

The CICM namespace is defined in [CICM].

2.2. Types

The following type definitions are defined in [CICM]:

*CICM::UInt32

*CICM::CharString

*CICM::Buffer

*CICM::Status (including all return values)

2.3. Interfaces

The interface CICM::Iterator is defined in [CICM]; the CICM::SymKey interface is defined in [CICM-KM].

3. Hardware Access Tokens

Cryptographic modules may rely upon hardware access tokens for two primary functions: to allow subjects (e.g., administrators or users in possession of a token) to be identified and authenticated so that privileges can be enforced and audit log entries can identify the subject; and to unlock all or some subset of cryptographic services. A hardware access token may be associated with a number of specific modules, and a module may likewise be associated with a number of specific tokens. The token management functions below enable tokens and modules to be associated with and disassociated from one another, and allow existing associations to be listed.
The removal of an association between a token and a module is straightforward if both the token and the module are available. However, if either the token or module are unavailable, or if a different module than the one originally associated with the token is used to remove an association from a token, the disassociation is less straightforward.
If a module requires that an administrative token be inserted prior to the token to which the association/disassociation will apply, the methods below may return an CICM::S_TOKEN_NOT_PRESENT or CICM::S_TOKEN_ADMIN_NOT_PRESENT status.
Modules that do not support hardware tokens may instead provide similar support via CICM::LoginManager. Modules may use CICM::LoginManager in tandem with tokens to support multi-factor authentication. See Managing Module Authentication in [CICM-LM] for additional information.

3.1. Token Management Identifiers

Type CICM::TokenRecord

```
typedef CICM::CharString TokenRecord;
```

Unique token association record.
Type CICM::ModuleRecord

```
typedef CICM::CharString ModuleRecord;
```

Unique module association record.

3.2. Interface CICM::TokenManager

Interface CICM::TokenManager

```
interface TokenManager {
```

CICM::TokenManager supports associating and disassociating modules and tokens. It is accessed from CICM::CryptoModule via the CICM::CryptoModule::token_manager attribute. CICM::TokenManager constructs the CICM::ModuleAssnIterator and CICM::TokenAssnIterator interfaces.
Example (C++):

```
// See CICMRoot::get_module_by_id()
// to retrieve reference to CryptoModule.
CICM::CryptoModule device;

CICM::Status sCode;
CICM::tokenManager tokenManager;

// Retrieve reference to the token manager.
tokenManager = device._get_token_manager();

// Associate the current token with the module.
sCode = tokenManager.associate();

// Assume that some time later the token is lost or destroyed.

// Disassociate the token from the module.
CICM::TokenUniqueId tokenId = new CICM::TokenUniqueId("TOKEN1426864");
sCode = tokenManager.disassociate_missing_token(tokenId);
```

3.2.1. CICM::TokenManager Attributes

Attribute CICM::TokenManager::module_association_iterator

```
readonly attribute CICM::ModuleAssnIterator
        module_association_iterator;
```

Returns an iterator to enable each module identifier associated with the current token to be retrieved.
Remarks:

- The returned iterator is set to the beginning of the iterated sequence.

Attribute CICM::TokenManager::token_association_iterator

```
readonly attribute CICM::TokenAssnIterator
        token_association_iterator;
```

Returns an iterator to enable each token identifier associated with the current module to be retrieved.

Remarks:

*The returned iterator is set to the beginning of the iterated sequence.

3.2.2. CICM::TokenManager Methods

Method CICM::TokenManager::associate()

```
CICM::Status associate(
        out CICM::ModuleRecord module_rec,
        out CICM::TokenRecord token_rec
);
```

Associate the module and currently-inserted hardware access token.
Remarks:

*The module and token record identifiers should be recorded for use in the disassociation process in the event that either the module or the token are no longer available or usable.

*The formats of the module and token records are not defined by CICM. The Implementation Conformance Statement (see Conformance and Extensions in [CICM]) MUST reference a standard format or define a module developer-specific format implemented by the module for these datatypes.

Parameters:

*[in] module_rec Module record identifier of the newly associated module.

*[in] token_rec Token record identifier of the newly associated token.

Returns:

*S_OK, S_GENERAL_ERROR, S_NON_FUNCTIONAL, S_OPERATION_FAILED, S_POLICY_VIOLATION, S_MODULE_RESOURCES, S_HOST_RESOURCES, S_INVALID_STATE, S_ALARM_STATE, S_MODULE_NOT_AVAILABLE, S_TIMEOUT, S_NOT_AUTHENTICATED, S_NOT_AUTHORIZED, S_TOKEN_NOT_PRESENT, S_TOKEN_ADMIN_NOT_PRESENT, S_TOKEN_ACCESS, S_TOKEN_RESOURCES, S_TOKEN_ASSOC_EXISTS, S_TOKEN_ASSOC_AT_MODULE, S_TOKEN_ASSOC_AT_TOKEN, S_TOKEN_ASSOC_GENERAL, S_TOKEN_TIMEOUT

Method CICM::TokenManager::disassociate()

```
CICM::Status disassociate();
```

Disassociate the module and currently-inserted hardware access token when the associated module and token are both present and both recognize the association.
Returns:

*S_OK, S_GENERAL_ERROR, S_NON_FUNCTIONAL, S_OPERATION_FAILED, S_POLICY_VIOLATION, S_MODULE_RESOURCES, S_HOST_RESOURCES, S_INVALID_STATE, S_ALARM_STATE, S_MODULE_NOT_AVAILABLE, S_TIMEOUT, S_NOT_AUTHENTICATED, S_NOT_AUTHORIZED, S_TOKEN_NOT_PRESENT, S_TOKEN_ADMIN_NOT_PRESENT, S_TOKEN_ACCESS, S_TOKEN_ASSOC_NOT_EXIST, S_TOKEN_DISASSOC_GENERAL, S_TOKEN_TIMEOUT, S_TOKEN_LAST_ASSOCIATED

Method CICM::TokenManager::disassociate_missing_module()

```
CICM::Status disassociate_missing_module(
        in  CICM::ModuleRecord module_rec
);
```

Remove association information from the currently-inserted hardware access token when the associated module is not present.
Remarks:

*The module on which this method is being executed is used as a surrogate to perform the disassociation (it is not the module that performed the initial association). The specific module to disassociate from the token is identified by a unique module identifier (e.g., a module serial number). Use CICM::ModuleAssnIterator to retrieve module record identifiers corresponding to modules associated with the inserted token.

*The format of the module record is not defined by CICM. The Implementation Conformance Statement (see Conformance and Extensions in [CICM]) MUST reference a standard format or define a module developer-specific format implemented by the module for this datatype.

Parameters:

*[in] module_rec Identifies the module for which module identification information should be removed from the currently-inserted hardware access token.

Returns:

*S_OK, S_GENERAL_ERROR, S_NON_FUNCTIONAL, S_OPERATION_FAILED, S_POLICY_VIOLATION, S_MODULE_RESOURCES, S_HOST_RESOURCES, S_INVALID_STATE, S_ALARM_STATE, S_MODULE_NOT_AVAILABLE, S_TIMEOUT, S_NOT_AUTHENTICATED, S_NOT_AUTHORIZED,

S_TOKEN_NOT_PRESENT, S_TOKEN_ADMIN_NOT_PRESENT, S_TOKEN_ACCESS,
S_TOKEN_DISASSOC_GENERAL, S_TOKEN_REC_NOT_FOUND, S_TOKEN_TIMEOUT

Method CICM::TokenManager::disassociate_missing_token()

```
CICM::Status disassociate_missing_token(
        in  CICM::TokenRecord token_rec
);
```

Remove association information from the module on which this method is being executed when the associated token is not present.
Remarks:

* The specific token to disassociate from the module is identified by a unique token identifier (e.g., a token serial number). Use CICM::TokenAssnIterator to retrieve token record identifiers corresponding to associated tokens from the module.

* The format of the token record is not defined by CICM. The Implementation Conformance Statement (see Conformance and Extensions in [CICM]) MUST reference a standard format or define a module developer-specific format implemented by the module for this datatype.

Parameters:

* [in] token_rec Identifies the hardware access token for which token identification information should be removed from the module.

Returns:

* S_OK, S_GENERAL_ERROR, S_NON_FUNCTIONAL, S_OPERATION_FAILED, S_POLICY_VIOLATION, S_MODULE_RESOURCES, S_HOST_RESOURCES, S_INVALID_STATE, S_ALARM_STATE, S_MODULE_NOT_AVAILABLE, S_TIMEOUT, S_NOT_AUTHENTICATED, S_NOT_AUTHORIZED, S_TOKEN_ADMIN_NOT_PRESENT, S_TOKEN_ASSOC_NOT_EXIST, S_TOKEN_DISASSOC_GENERAL, S_TOKEN_REC_NOT_FOUND, S_TOKEN_TIMEOUT

3.3. Interface CICM::TokenAssnIterator

Interface CICM::TokenAssnIterator

```
interface TokenAssnIterator : CICM::Iterator {
```

CICM::TokenAssnIterator supports retrieving each token record from the token association list in the module.

3.3.1. CICM::TokenAssnIterator Inheritance

CICM::TokenAssnIterator inherits from: CICM::Iterator.

3.3.2. CICM::TokenAssnIterator Methods

Method CICM::TokenAssnIterator::get_next()

```
CICM::Status get_next(
        out CICM::TokenRecord token_rec_ref
);
```

Returns a reference to the next token.
Remarks:

*Use CICM::Iterator::has_next to determine if additional elements exist.

Parameters:

*[out] token_rec_ref Reference to next token.

Returns:

*S_OK, S_GENERAL_ERROR, S_NON_FUNCTIONAL, S_OPERATION_FAILED, S_POLICY_VIOLATION, S_MODULE_RESOURCES, S_HOST_RESOURCES, S_INVALID_STATE, S_ALARM_STATE, S_MODULE_NOT_AVAILABLE, S_TIMEOUT, S_NOT_AUTHENTICATED, S_NOT_AUTHORIZED, S_TOKEN_NOT_PRESENT, S_TOKEN_ADMIN_NOT_PRESENT

3.4. Interface CICM::ModuleAssnIterator

Interface CICM::ModuleAssnIterator

```
interface ModuleAssnIterator : CICM::Iterator {
```

CICM::ModuleAssnIterator supports retrieving each module record from the module association list in the token.

3.4.1. CICM::ModuleAssnIterator Inheritance

CICM::ModuleAssnIterator inherits from: CICM::Iterator.

3.4.2. CICM::ModuleAssnIterator Methods

Method CICM::ModuleAssnIterator::get_next()

```
CICM::Status get_next(
        out CICM::ModuleRecord module_rec_ref
);
```

Returns a reference to the next module record from the module association list in the token.
Remarks:

*Use CICM::Iterator::has_next to determine if additional elements exist.

Parameters:

*[out] module_rec_ref Reference to next module record.

Returns:

*S_OK, S_GENERAL_ERROR, S_NON_FUNCTIONAL, S_OPERATION_FAILED, S_POLICY_VIOLATION, S_MODULE_RESOURCES, S_HOST_RESOURCES, S_INVALID_STATE, S_ALARM_STATE, S_MODULE_NOT_AVAILABLE, S_TIMEOUT, S_NOT_AUTHENTICATED, S_NOT_AUTHORIZED, S_TOKEN_NOT_PRESENT, S_TOKEN_ADMIN_NOT_PRESENT

4. Users

These interfaces enable the management of users in support of password-based login. See the Managing Module Authentication in [CICM-LM] for additional information.

4.1. User Management Identifiers

Type CICM::UserId

typedef CICM::CharString UserId;

Unique user identifier.
Type CICM::RoleId

typedef CICM::CharString RoleId;

Unique role identifier.

4.2. Interface CICM::UserManager

Interface CICM::UserManager

interface UserManager {

CICM::UserManager supports adding a user/password, modifying a user's password, and removing users; and associating and disassociating users from a role. It is accessed from CICM::CryptoModule via the CICM::CryptoModule::user_manager attribute. CICM::UserManager constructs the CICM::UserIdIterator and CICM::RoleIdIterator interfaces.

Example (C++):

```
// See CICMRoot::get_module_by_id()
// to retrieve reference to CryptoModule.
CICM::CryptoModule device;

CICM::Status sCode;
CICM::UserManager userManager;

// Retrieve reference to the user manager.
userManager = device._get_user_manager();

// Create the user.
CICM::UserUniqueId userId = "bob_smith";
CICM::CharString password = "p@$$w0rd";
sCode = userManager.add(userId, password);

// Associate the user with a pre-defined module role.
CICM::RoleUniqueId roleId = "administrator";
sCode = userManager.associate(userId, roleId);

// Destroy the user.
sCode = userManager.remove(userId);
```

4.2.1. CICM::UserManager Attributes

Attribute CICM::UserManager::user_iterator

```
readonly attribute CICM::UserIdIterator user_iterator;
```

Returns an iterator to enable an identifier for each user in the module user database to be retrieved.
Remarks:

- The returned iterator is set to the beginning of the iterated sequence.

Attribute CICM::UserManager::role_iterator

```
readonly attribute CICM::RoleIdIterator role_iterator;
```

Returns an iterator to enable an identifier for each role supported by the module to be retrieved.
Remarks:

- The returned iterator is set to the beginning of the iterated sequence.

4.2.2. CICM::UserManager Methods

Method CICM::UserManager::add()

```
CICM::Status add(
        in  CICM::UserId user,
        in  CICM::CharString password
);
```

Add a user to the module user database.
Parameters:

*[in] user New user to add.

*[in] password New user's password.

Returns:

*S_OK, S_GENERAL_ERROR, S_NON_FUNCTIONAL, S_OPERATION_FAILED,
S_POLICY_VIOLATION, S_MODULE_RESOURCES, S_HOST_RESOURCES,
S_INVALID_STATE, S_ALARM_STATE, S_MODULE_NOT_AVAILABLE,
S_TIMEOUT, S_NOT_AUTHENTICATED, S_NOT_AUTHORIZED,
S_TOKEN_NOT_PRESENT, S_TOKEN_ADMIN_NOT_PRESENT,
S_USERNAME_INVALID, S_USER_EXISTS, S_PASSWORD_INVALID,
S_PASSWORD_INVALID_CHAR, S_PASSWORD_INVALID_LEN

Method CICM::UserManager::modify()

```
CICM::Status modify(
        in  CICM::UserId user,
        in  CICM::CharString password
);
```

Change the password of a user in the module user database.
Parameters:

*[in] user User to modify.

*[in] password User's new password.

Returns:

*S_OK, S_GENERAL_ERROR, S_NON_FUNCTIONAL, S_OPERATION_FAILED,
S_POLICY_VIOLATION, S_MODULE_RESOURCES, S_HOST_RESOURCES,
S_INVALID_STATE, S_ALARM_STATE, S_MODULE_NOT_AVAILABLE,
S_TIMEOUT, S_NOT_AUTHENTICATED, S_NOT_AUTHORIZED,
S_TOKEN_NOT_PRESENT, S_TOKEN_ADMIN_NOT_PRESENT, S_USER_INVALID,
S_PASSWORD_INVALID, S_PASSWORD_INVALID_CHAR,
S_PASSWORD_INVALID_LEN

Method CICM::UserManager::remove()

```
CICM::Status remove(
        in  CICM::UserId user
);

Remove a user from the module user database.
Parameters:

     *[in] user User to remove.

Returns:

     *S_OK, S_GENERAL_ERROR, S_NON_FUNCTIONAL, S_OPERATION_FAILED,
      S_POLICY_VIOLATION, S_MODULE_RESOURCES, S_HOST_RESOURCES,
      S_INVALID_STATE, S_ALARM_STATE, S_MODULE_NOT_AVAILABLE,
      S_TIMEOUT, S_NOT_AUTHENTICATED, S_NOT_AUTHORIZED,
      S_TOKEN_NOT_PRESENT, S_TOKEN_ADMIN_NOT_PRESENT, S_USER_INVALID

Method CICM::UserManager::associate()

CICM::Status associate(
        in  CICM::UserId user,
        in  CICM::RoleId role
);

Associate a role with the specified user.
Parameters:

     *[in] user User to associate.

     *[in] role Role to associate with the user.

Returns:

     *S_OK, S_GENERAL_ERROR, S_NON_FUNCTIONAL, S_OPERATION_FAILED,
      S_POLICY_VIOLATION, S_MODULE_RESOURCES, S_HOST_RESOURCES,
      S_INVALID_STATE, S_ALARM_STATE, S_MODULE_NOT_AVAILABLE,
      S_TIMEOUT, S_NOT_AUTHENTICATED, S_NOT_AUTHORIZED,
      S_TOKEN_NOT_PRESENT, S_TOKEN_ADMIN_NOT_PRESENT, S_USER_INVALID,
      S_ROLE_INVALID, S_ROLE_ASSOCIATED, S_ROLE_MAX

Method CICM::UserManager::disassociate()

CICM::Status disassociate(
        in  CICM::UserId user,
        in  CICM::RoleId role
);

Disassociate a role from the specified user.
```

Parameters:

*[in] user User to disassociate.

*[in] role Role to disassociate from the user.

Returns:

*S_OK, S_GENERAL_ERROR, S_NON_FUNCTIONAL, S_OPERATION_FAILED,
S_POLICY_VIOLATION, S_MODULE_RESOURCES, S_HOST_RESOURCES,
S_INVALID_STATE, S_ALARM_STATE, S_MODULE_NOT_AVAILABLE,
S_TIMEOUT, S_NOT_AUTHENTICATED, S_NOT_AUTHORIZED,
S_TOKEN_NOT_PRESENT, S_TOKEN_ADMIN_NOT_PRESENT, S_USER_INVALID,
S_ROLE_INVALID, S_ROLE_NOT_ASSOCIATED

4.3. Interface CICM::UserIdIterator

Interface CICM::UserIdIterator

interface UserIdIterator : CICM::Iterator {

CICM::UserIdIterator supports retrieving each user configured on a
module.

4.3.1. CICM::UserIdIterator Inheritance

CICM::UserIdIterator inherits from: CICM::Iterator.

4.3.2. CICM::UserIdIterator Methods

Method CICM::UserIdIterator::get_next()

```
CICM::Status get_next(
        out CICM::UserId user_id
);
```

Returns the next user identifier.
Remarks:

*Use CICM::Iterator::has_next to determine if additional elements
exist.

Parameters:

*[out] user_id Next user identifier.

Returns:

*S_OK, S_GENERAL_ERROR, S_NON_FUNCTIONAL, S_OPERATION_FAILED,
S_POLICY_VIOLATION, S_MODULE_RESOURCES, S_HOST_RESOURCES,

S_INVALID_STATE, S_ALARM_STATE, S_MODULE_NOT_AVAILABLE,
S_TIMEOUT, S_NOT_AUTHENTICATED, S_NOT_AUTHORIZED,
S_TOKEN_NOT_PRESENT, S_TOKEN_ADMIN_NOT_PRESENT

4.4. Interface CICM::RoleIdIterator

Interface CICM::RoleIdIterator

interface RoleIdIterator : CICM::Iterator {

CICM::RoleIdIterator supports retrieving each role available on a module.

4.4.1. CICM::RoleIdIterator Inheritance

CICM::RoleIdIterator inherits from: CICM::Iterator.

4.4.2. CICM::RoleIdIterator Methods

Method CICM::RoleIdIterator::get_next()

```
CICM::Status get_next(
        out CICM::RoleId role_id
);
```

Returns the next role identifier.
Remarks:

*Use CICM::Iterator::has_next to determine if additional elements exist.

Parameters:

*[out] role_id Reference to next role identifier.

Returns:

*S_OK, S_GENERAL_ERROR, S_NON_FUNCTIONAL, S_OPERATION_FAILED, S_POLICY_VIOLATION, S_MODULE_RESOURCES, S_HOST_RESOURCES, S_INVALID_STATE, S_ALARM_STATE, S_MODULE_NOT_AVAILABLE, S_TIMEOUT, S_NOT_AUTHENTICATED, S_NOT_AUTHORIZED, S_TOKEN_NOT_PRESENT, S_TOKEN_ADMIN_NOT_PRESENT

5. Login

These interfaces support a user configured on a module to login to a module using a password and, optionally, additional authentication data. See the Managing Module Authentication in [CICM-LM] for additional information.

Modules that support hardware tokens may use the login manager in tandem with the CICM::TokenManager to support multi-factor authentication.

5.1. Interface CICM::LoginManager

Interface CICM::LoginManager

interface LoginManager {

CICM::LoginManager supports user login to a module. It is accessed from CICM::CryptoModule via the CICM::CryptoModule::login_manager attribute. CICM::LoginManager constructs the CICM::Login interface. The LoginManager relies upon the CICM::UserManager to manage the users that are specified to the login methods.
Example (C++):

```
// See CICMRoot::get_module_by_id()
// to retrieve reference to CryptoModule.
CICM::CryptoModule device;
CICM::Status sCode;
CICM::LoginManager loginManager;
CICM::Login loginRef;
// Retrieve reference to the login manager.
loginManager = device._get_login_manager();
// Login to the module.
CICM::UserUniqueId userId = "bob_smith";
CICM::CharString password = "p@$$w0rd";
sCode = loginManager.add(userId, password, &loginRef);
// Logout from the module.
sCode = loginRef.logout();
```

5.1.1. CICM::LoginManager Methods

Method CICM::LoginManager::login()

```
 CICM::Status login(
        in  CICM::UserId user,
        in  CICM::CharString password,
        out CICM::Login login_ref
);
```

Login to the module with username/password.
Parameters:

*[in] user User attempting to login.

*[in] password User's password.

*[out] login_ref Reference to state resulting from successful user login enabling the user to later logout.

Returns:

*S_OK, S_GENERAL_ERROR, S_NON_FUNCTIONAL, S_OPERATION_FAILED, S_POLICY_VIOLATION, S_MODULE_RESOURCES, S_HOST_RESOURCES, S_INVALID_STATE, S_ALARM_STATE, S_MODULE_NOT_AVAILABLE, S_TIMEOUT, S_NOT_AUTHENTICATED, S_NOT_AUTHORIZED, S_TOKEN_NOT_PRESENT, S_TOKEN_ADMIN_NOT_PRESENT, S_AUTHENTICATION_FAILED, S_USER_AUTHENTICATED

Method CICM::LoginManager::login_auth_data()

```
CICM::Status login_auth_data(
        in  CICM::UserId user,
        in  CICM::CharString password,
        in  CICM::Buffer auth_data,
        out CICM::Login login_ref
);
```

Login to the module with username/password, but provide additional (potentially host-stored) authentication data to the module for use in the authentication process.
Remarks:

*This may be used in cases where the host supports a virtual token.

*The format of the authentication data is not defined by CICM. The Implementation Conformance Statement (see Conformance and Extensions in [CICM]) MUST reference a standard format or define a module developer-specific format implemented by the module for this datatype.

Parameters:

*[in] user User attempting to login.

*[in] password User's password.

*[in] auth_data Additional host-stored authentication data.

*[out] login_ref Reference to state resulting from successful user login enabling the user to later logout.

Returns:

*S_OK, S_GENERAL_ERROR, S_NON_FUNCTIONAL, S_OPERATION_FAILED, S_POLICY_VIOLATION, S_MODULE_RESOURCES, S_HOST_RESOURCES,

S_INVALID_STATE, S_ALARM_STATE, S_MODULE_NOT_AVAILABLE,
S_TIMEOUT, S_NOT_AUTHENTICATED, S_NOT_AUTHORIZED,
S_INVALID_DATA_BUFFER, S_TOKEN_NOT_PRESENT,
S_TOKEN_ADMIN_NOT_PRESENT, S_AUTHENTICATION_FAILED,
S_USER_AUTHENTICATED

5.2. Interface CICM::Login

Interface CICM::Login

interface Login {

CICM::Login results from a successful user login to a module and
enables the user to log out from the module.

5.2.1. CICM::Login Methods

Method CICM::Login::logout()

CICM::Status logout();

Logout of the module.
Remarks:

*This may be equivalent to disconnecting a hardware access token
from a module in certain systems.

Returns:

*S_OK, S_GENERAL_ERROR, S_NON_FUNCTIONAL, S_OPERATION_FAILED,
S_POLICY_VIOLATION, S_MODULE_RESOURCES, S_HOST_RESOURCES,
S_INVALID_STATE, S_ALARM_STATE, S_MODULE_NOT_AVAILABLE,
S_TIMEOUT, S_NOT_AUTHENTICATED, S_NOT_AUTHORIZED,
S_TOKEN_NOT_PRESENT, S_TOKEN_ADMIN_NOT_PRESENT

6. Software Packages

These interfaces support software, FPGA images, policy databases,
configuration parameters, or other types of executable or interpretable
code to be imported into and removed from a module.

6.1. Package Management Identifier

Type CICM::PackageId

typedef CICM::CharString PackageId;

Unique package identifier.

6.2. Interface CICM::PackageManager

Interface CICM::PackageManager

interface PackageManager {

CICM::PackageManager supports the management of module software packages. It is accessed from CICM::CryptoModule via the CICM::CryptoModule::package_manager attribute. CICM::PackageManager constructs the CICM::PackageImporter, CICM::PackageIterator, and CICM::Package interfaces.
Example (C++):

```
// See CICMRoot::get_module_by_id()
// to retrieve reference to CryptoModule.
CICM::CryptoModule device;
CICM::Status sCode;
CICM::PackageManager packageManager;
CICM::PackageImporter packageImporter;
// Retrieve reference to the package manager.
packageManager = device._get_package_manager();

// Initialize the import process.
sCode = packageManager.import_package(
        CICM::Package::C_PACKAGE_FPGA_IMAGE, &packageImporter);

// Assume FPGA image data in [fpgaData].
CICM::Buffer fpgaData;
sCode = packageImporter.import_segment(fpgaData);

// Assume all segments are imported.
// Complete the import process.

CICM::Package fpgaPackage;
sCode = packageImporter.complete(&fpgaPackage);
// If successful, [fpgaPackage] is a reference to the imported package.
// Activate the package.
sCode = fpgaPackage.activate();
```

6.2.1. CICM::PackageManager Attributes

Attribute CICM::PackageManager::package_iterator

readonly attribute CICM::PackageIterator package_iterator;

Returns an iterator to enable a reference to each package loaded on the module to be retrieved.
Remarks:

- The returned iterator is set to the beginning of the iterated sequence.

6.2.2. CICM::PackageManager Methods

Method CICM::PackageManager::import_package()

```
CICM::Status import_package(
        in  CICM::Package::PackageType package_type,
        out CICM::PackageImporter importer_ref
);
```

Initiate the process of importing a package into the module.
Remarks:

*The CICM::PackageImporter that results from this call is used to import package segments into the module. It is the responsibility of the caller to break a package into segments, import each individual segment, and then call CICM::PackageImporter::complete to receive a reference to the resulting package. Note that the key required to decrypt any encrypted package segments MUST be referenced within the package and MUST be available to the module; the key MAY be explicitly specified by using the CICM::PackageManager::import_package_with_key version of the call.

Parameters:

*[in] package_type Type of the package being imported.

*[out] importer_ref Reference to package importer interface which enables a package to be imported segment by segment.

Returns:

*S_OK, S_GENERAL_ERROR, S_NON_FUNCTIONAL, S_OPERATION_FAILED, S_POLICY_VIOLATION, S_MODULE_RESOURCES, S_HOST_RESOURCES, S_INVALID_STATE, S_ALARM_STATE, S_MODULE_NOT_AVAILABLE, S_TIMEOUT, S_NOT_AUTHENTICATED, S_NOT_AUTHORIZED, S_TOKEN_NOT_PRESENT, S_TOKEN_ADMIN_NOT_PRESENT, S_PACKAGE_TYPE_INVALID, S_PACKAGE_KEY_NOT_AVAILABLE, S_PACKAGE_KEY_NOT_SPECIFIED

Method CICM::PackageManager::import_package_with_key()

```
CICM::Status import_package_with_key(
        in  CICM::Package::PackageType package_type,
        in  CICM::SymKey key_ref,
        out CICM::PackageImporter importer_ref
);
```

Initiate the process of importing a package into the module, specifying a reference to the key that will be used by CICM::PackageImporter to decrypt each package segment.
Remarks:

*The CICM::PackageImporter that results from this call is used to import package segments into the module. It is the responsibility of the caller to break a package into segments, import each individual segment, and then call CICM::PackageImporter::complete to receive a reference to the resulting package.

Parameters:

*[in] package_type Type of the package being imported.

*[in] key_ref Reference to key to decrypt package segments.

*[out] importer_ref Reference to package importer interface which enables a package to be imported segment by segment.

Returns:

*S_OK, S_GENERAL_ERROR, S_NON_FUNCTIONAL, S_OPERATION_FAILED, S_POLICY_VIOLATION, S_MODULE_RESOURCES, S_HOST_RESOURCES, S_INVALID_STATE, S_ALARM_STATE, S_MODULE_NOT_AVAILABLE, S_TIMEOUT, S_NOT_AUTHENTICATED, S_NOT_AUTHORIZED, S_TOKEN_NOT_PRESENT, S_TOKEN_ADMIN_NOT_PRESENT, S_PACKAGE_TYPE_INVALID

Method CICM::PackageManager::get_package_by_id()

```
CICM::Status get_package_by_id(
        in  CICM::PackageId package_id,
        out CICM::Package package_ref
);
```

Retrieve a reference to a package based upon a unique identifier associated with that package.
Parameters:

*[in] package_id Package identifier.

*[out] package_ref Reference to package corresponding to the specified identifier.

Returns:

*S_OK, S_GENERAL_ERROR, S_NON_FUNCTIONAL, S_OPERATION_FAILED, S_POLICY_VIOLATION, S_MODULE_RESOURCES, S_HOST_RESOURCES, S_INVALID_STATE, S_ALARM_STATE, S_MODULE_NOT_AVAILABLE,

S_TIMEOUT, S_NOT_AUTHENTICATED, S_NOT_AUTHORIZED,
S_NOT_AVAILABLE, S_TOKEN_NOT_PRESENT, S_TOKEN_ADMIN_NOT_PRESENT

Method CICM::PackageManager::reencrypt_software()

CICM::Status reencrypt_software();

Re-encrypt module software with a key managed by the module.
Returns:

*S_OK, S_GENERAL_ERROR, S_NON_FUNCTIONAL, S_OPERATION_FAILED,
S_POLICY_VIOLATION, S_MODULE_RESOURCES, S_HOST_RESOURCES,
S_INVALID_STATE, S_ALARM_STATE, S_MODULE_NOT_AVAILABLE,
S_TIMEOUT, S_NOT_AUTHENTICATED, S_NOT_AUTHORIZED,
S_TOKEN_NOT_PRESENT, S_TOKEN_ADMIN_NOT_PRESENT,
S_INSUFFICIENT_ENTROPY

6.3. Interface CICM::PackageImporter

Interface CICM::PackageImporter

interface PackageImporter {

CICM::PackageImporter supports importing software packages, segment by segment. CICM::PackageImporter is constructed by the CICM::PackageManager::import_package and CICM::PackageManager::import_package_with_key methods and cannot be instantiated independently. CICM::PackageImporter constructs the CICM::Package interface.

6.3.1. CICM::PackageImporter Methods

Method CICM::PackageImporter::import_segment()

```
CICM::Status import_segment(
        in  CICM::Buffer package_data
);
```

Import one segment of a package.
Remarks:

*It is the responsibility of the caller to break a package into segments, import each individual segment, and then call CICM::PackageImporter::complete to receive a reference to the resulting package.

*CICM does not specify the structure of the binary data that constitutes the package being imported. The Implementation Conformance Statement (see Conformance and Extensions in [CICM])

MUST reference a standard format or define a module developer-specific format implemented by the module for this datatype.

Parameters:

*[in] package_data Contents of the package.

Returns:

*S_OK, S_GENERAL_ERROR, S_NON_FUNCTIONAL, S_OPERATION_FAILED, S_POLICY_VIOLATION, S_MODULE_RESOURCES, S_HOST_RESOURCES, S_INVALID_STATE, S_ALARM_STATE, S_MODULE_NOT_AVAILABLE, S_TIMEOUT, S_NOT_AUTHENTICATED, S_NOT_AUTHORIZED, S_TOKEN_NOT_PRESENT, S_TOKEN_ADMIN_NOT_PRESENT, S_PACKAGE_INVALID

Method CICM::PackageImporter::complete()

```
CICM::Status complete(
        out CICM::Package package_ref
);
```

Declare the package import complete and retrieve a reference to the resulting package object.
Remarks:

*If this method is called before the package is fully loaded, the CICM::S_PACKAGE_INVALID status results.

Parameters:

*[out] package_ref Reference to resulting imported package.

Returns:

*S_OK, S_GENERAL_ERROR, S_NON_FUNCTIONAL, S_OPERATION_FAILED, S_MODULE_RESOURCES, S_HOST_RESOURCES, S_INVALID_STATE, S_ALARM_STATE, S_MODULE_NOT_AVAILABLE, S_TIMEOUT, S_NOT_AUTHENTICATED, S_NOT_AUTHORIZED, S_TOKEN_NOT_PRESENT, S_TOKEN_ADMIN_NOT_PRESENT, S_PACKAGE_INVALID

Method CICM::PackageImporter::abort()

```
CICM::Status abort();
```

Abort a package import, resetting this CICM::PackageImporter instance, allowing a new package import session to begin.
Remarks:

*Segments already imported are discarded.

Returns:

*S_OK, S_GENERAL_ERROR, S_NON_FUNCTIONAL, S_OPERATION_FAILED,
S_POLICY_VIOLATION, S_MODULE_RESOURCES, S_HOST_RESOURCES,
S_INVALID_STATE, S_ALARM_STATE, S_MODULE_NOT_AVAILABLE,
S_TIMEOUT, S_NOT_AUTHENTICATED, S_NOT_AUTHORIZED,
S_TOKEN_NOT_PRESENT, S_TOKEN_ADMIN_NOT_PRESENT

6.4. Interface CICM::Package

Interface CICM::Package

interface Package {

CICM::Package serves as a reference to a package previously loaded into a module, and supports activating, deactivating, and deleting the package. CICM::Package is constructed by the CICM::PackageManager::get_package_by_id and CICM::PackageImporter::complete methods and cannot be instantiated independently.

6.4.1. CICM::Package Types and Constants

Type CICM::Package::PackageType

typedef CICM::UInt32 PackageType;

Supported package types.
Constant CICM::Package::C_PACKAGE_ALGORITHM

const CICM::Package::PackageType
C_PACKAGE_ALGORITHM = 0x00006054;

Algorithm package type.
Constant CICM::Package::C_PACKAGE_CONFIG_PARAMS

const CICM::Package::PackageType
C_PACKAGE_CONFIG_PARAMS = 0x00006057;

Configuration parameter package type.
Constant CICM::Package::C_PACKAGE_FPGA_IMAGE

const CICM::Package::PackageType
C_PACKAGE_FPGA_IMAGE = 0x00006058;

FPGA image package type.
Constant CICM::Package::C_PACKAGE_POLICY_DB

const CICM::Package::PackageType
C_PACKAGE_POLICY_DB = 0x0000605B;

Policy database package type.
Constant CICM::Package::C_PACKAGE_SOFTWARE

```
const CICM::Package::PackageType
        C_PACKAGE_SOFTWARE = 0x0000605D;
```

Software package type.

6.4.2. CICM::Package Attributes

Attribute CICM::Package::id

```
readonly attribute CICM::PackageId id;
```

Unique package identifier of this package.

6.4.3. CICM::Package Methods

Method CICM::Package::activate()

```
CICM::Status activate();
```

Activate a specific package on the module.
Remarks:

- It may be necessary to reset the module before the specified package is activated in place of the currently activated package.

Returns:

- S_OK, S_GENERAL_ERROR, S_NON_FUNCTIONAL, S_OPERATION_FAILED, S_POLICY_VIOLATION, S_MODULE_RESOURCES, S_HOST_RESOURCES, S_INVALID_STATE, S_ALARM_STATE, S_MODULE_NOT_AVAILABLE, S_TIMEOUT, S_NOT_AUTHENTICATED, S_NOT_AUTHORIZED, S_TOKEN_NOT_PRESENT, S_TOKEN_ADMIN_NOT_PRESENT, S_PACKAGE_NOT_ACTIVATABLE, S_PACKAGE_ACTIVATED, S_PACKAGE_INVALID

Method CICM::Package::deactivate()

```
CICM::Status deactivate();
```

Deactivate a specific package on the module.
Returns:

- S_OK, S_GENERAL_ERROR, S_NON_FUNCTIONAL, S_OPERATION_FAILED, S_POLICY_VIOLATION, S_MODULE_RESOURCES, S_HOST_RESOURCES, S_INVALID_STATE, S_ALARM_STATE, S_MODULE_NOT_AVAILABLE, S_TIMEOUT, S_NOT_AUTHENTICATED, S_NOT_AUTHORIZED, S_TOKEN_NOT_PRESENT, S_TOKEN_ADMIN_NOT_PRESENT, S_PACKAGE_NOT_ACTIVE, S_PACKAGE_INVALID

Method CICM::Package::delete()

CICM::Status delete();

Delete a package from the module.
Returns:

*S_OK, S_GENERAL_ERROR, S_NON_FUNCTIONAL, S_OPERATION_FAILED, S_POLICY_VIOLATION, S_MODULE_RESOURCES, S_HOST_RESOURCES, S_INVALID_STATE, S_ALARM_STATE, S_MODULE_NOT_AVAILABLE, S_TIMEOUT, S_NOT_AUTHENTICATED, S_NOT_AUTHORIZED, S_TOKEN_NOT_PRESENT, S_TOKEN_ADMIN_NOT_PRESENT, S_PACKAGE_ACTIVATED, S_PACKAGE_INVALID

6.5. Interface CICM::PackageIterator

Interface CICM::PackageIterator

interface PackageIterator : CICM::Iterator {

CICM::PackageIterator supports retrieving a reference to each software package available on a module. CICM::PackageIterator constructs the CICM::Package interface.

6.5.1. CICM::PackageIterator Inheritance

CICM::PackageIterator inherits from: CICM::Iterator.

6.5.2. CICM::PackageIterator Methods

Method CICM::PackageIterator::get_next()

```
CICM::Status get_next(
        out CICM::Package package_ref
);
```

Returns a reference to the next software package.
Remarks:

*Use CICM::Iterator::has_next to determine if additional elements exist.

Parameters:

*[out] package_ref Reference to next software package.

Returns:

*S_OK, S_GENERAL_ERROR, S_NON_FUNCTIONAL, S_OPERATION_FAILED, S_POLICY_VIOLATION, S_MODULE_RESOURCES, S_HOST_RESOURCES,

```
      S_INVALID_STATE, S_ALARM_STATE, S_MODULE_NOT_AVAILABLE,
      S_TIMEOUT, S_NOT_AUTHENTICATED, S_NOT_AUTHORIZED,
      S_TOKEN_NOT_PRESENT, S_TOKEN_ADMIN_NOT_PRESENT
```

7. Logs

These interfaces support the retrieval and removal of log entries.

7.1. Interface CICM::LogManager

Interface CICM::LogManager

interface LogManager {

CICM::LogManager supports retrieving or destroying an entire module log, or retrieving or deleting individual log entries. It is accessed from CICM::CryptoModule via the CICM::CryptoModule::log_manager attribute. CICM::LogManager constructs the CICM::LogEntryIterator interface.
Example (C++):

```
// See CICMRoot::get_module_by_id()
// to retrieve reference to CryptoModule.
CICM::CryptoModule device;
CICM::Status sCode;
CICM::LogManager logManager;

// Retrieve reference to the log manager.
logManager = device._get_log_manager();

// Retrieve reference to a log entry iterator.
CICM::LogEntryIterator iter;
iter = logManager._get_log_entry_iterator();
CICM::Iterator::Status status;
CICM::LogEntry entry;

// Confirm that there are log entries.
sCode = iter.hasNext(&status);

// Iterate over the log entries.
while( CICM::Iterator::C_ITERATOR_HAS_NEXT == status ) {
        sCode = iter.get_next(&entry);
                // Perform an operation on [entry].
        sCode = iter.hasNext(&status);
}
// Delete all of the log entries.
sCode = logManager.destroy();
```

7.1.1. CICM::LogManager Attributes

Attribute CICM::LogManager::log_entry_iterator

readonly attribute CICM::LogEntryIterator log_entry_iterator;

Returns an iterator to enable a reference to each module CICM::LogEntry to be retrieved.
Remarks:

*The returned iterator is set to the beginning of the iterated sequence.

7.1.2. CICM::LogManager Methods

Method CICM::LogManager::retrieve()

```
 CICM::Status retrieve(
        out CICM::Buffer log_ref
);
```

Retrieve a reference to the entire module log.
Parameters:

*[out] log_ref Reference to entire module log.

Returns:

*S_OK, S_GENERAL_ERROR, S_NON_FUNCTIONAL, S_OPERATION_FAILED, S_POLICY_VIOLATION, S_MODULE_RESOURCES, S_HOST_RESOURCES, S_INVALID_STATE, S_ALARM_STATE, S_MODULE_NOT_AVAILABLE, S_TIMEOUT, S_NOT_AUTHENTICATED, S_NOT_AUTHORIZED, S_TOKEN_NOT_PRESENT, S_TOKEN_ADMIN_NOT_PRESENT

Method CICM::LogManager::destroy()

CICM::Status destroy();

Destroy all entries in the module log.
Returns:

*S_OK, S_GENERAL_ERROR, S_NON_FUNCTIONAL, S_OPERATION_FAILED, S_POLICY_VIOLATION, S_MODULE_RESOURCES, S_HOST_RESOURCES, S_INVALID_STATE, S_ALARM_STATE, S_MODULE_NOT_AVAILABLE, S_TIMEOUT, S_NOT_AUTHENTICATED, S_NOT_AUTHORIZED, S_TOKEN_NOT_PRESENT, S_TOKEN_ADMIN_NOT_PRESENT

7.2. Interface CICM::LogEntry

Interface CICM::LogEntry

```
interface LogEntry {
```

CICM::LogEntry serves as a reference to an individual log entry in the module log, and supports retrieving information about an individual log entry and deleting an individual log entry.

7.2.1. CICM::LogEntry Attributes

Attribute CICM::LogEntry::user_id

readonly attribute CICM::UserId user_id;

User initiating the module action resulting in this log entry.
Attribute CICM::LogEntry::role_id

readonly attribute CICM::RoleId role_id;

Role under which the module action resulting in this log entry was initiated.
Attribute CICM::LogEntry::message

readonly attribute CICM::CharString message;

Log message associated with this log entry.
Attribute CICM::LogEntry::date_time

readonly attribute CICM::CharString date_time;

Date/time of creation of this log entry.

7.2.2. CICM::LogEntry Methods

Method CICM::LogEntry::delete()

CICM::Status delete();

Remove the current entry from the module log.
Returns:

*S_OK, S_GENERAL_ERROR, S_NON_FUNCTIONAL, S_OPERATION_FAILED, S_POLICY_VIOLATION, S_MODULE_RESOURCES, S_HOST_RESOURCES, S_INVALID_STATE, S_ALARM_STATE, S_MODULE_NOT_AVAILABLE, S_TIMEOUT, S_NOT_AUTHENTICATED, S_NOT_AUTHORIZED, S_TOKEN_NOT_PRESENT, S_TOKEN_ADMIN_NOT_PRESENT, S_LOG_ENTRY_INVALID

7.3. Interface CICM::LogEntryIterator

Interface CICM::LogEntryIterator

interface LogEntryIterator : CICM::Iterator {

CICM::LogEntryIterator supports retrieving a reference to each log entry in the module log. CICM::LogEntryIterator constructs the CICM::LogEntry interface.

7.3.1. CICM::LogEntryIterator Inheritance

CICM::LogEntryIterator inherits from: CICM::Iterator.

7.3.2. CICM::LogEntryIterator Methods

Method CICM::LogEntryIterator::get_next()

```
CICM::Status get_next(
        out CICM::LogEntry log_entry_ref
);
```

Returns a reference to the next log entry.
Remarks:

*Use CICM::Iterator::has_next to determine if additional elements exist.

Parameters:

*[out] log_entry_ref Reference to next log entry.

Returns:

*S_OK, S_GENERAL_ERROR, S_NON_FUNCTIONAL, S_OPERATION_FAILED, S_POLICY_VIOLATION, S_MODULE_RESOURCES, S_HOST_RESOURCES, S_INVALID_STATE, S_ALARM_STATE, S_MODULE_NOT_AVAILABLE, S_TIMEOUT, S_NOT_AUTHENTICATED, S_NOT_AUTHORIZED, S_TOKEN_NOT_PRESENT, S_TOKEN_ADMIN_NOT_PRESENT

8. Tests

These interfaces support the initiation of module internal tests by client programs.

8.1. Interface CICM::TestManager

Interface CICM::TestManager

```
interface TestManager {
```

CICM::TestManager supports initiating client program-invoked module built-in tests. It is accessed from CICM::CryptoModule via the CICM::CryptoModule::test_manager attribute.

8.1.1. CICM::TestManager Types and Constants

Type CICM::TestManager::Status

typedef CICM::UInt32 Status;

Test state at completion.
Constant CICM::TestManager::C_TEST_SUCCESS

const CICM::TestManager::Status
        C_TEST_SUCCESS = 0x00006062;

The test completed successfully.
Constant CICM::TestManager::C_TEST_FAILURE

const CICM::TestManager::Status
        C_TEST_FAILURE = 0x00006064;

The test failed.

8.1.2. CICM::TestManager Methods

Method CICM::TestManager::run_test()

```
CICM::Status run_test(
        in  CICM::Buffer test_parameters,
        out CICM::TestManager::Status status
);
```

Run module built-in tests specifying module-specific test parameters.
Remarks:

*This method can only initiate tests that a client program can explicitly request (e.g., this method does not apply to a series of tests automatically initiated during a module's start-up sequence). Running built-in tests on some modules may result in an alarm if an error is encountered during the test run.

*The format of the test parameters value is not defined by CICM. The Implementation Conformance Statement (see Conformance and Extensions in [CICM]) MUST reference a standard format or define a module developer-specific format implemented by the module for this datatype.

Parameters:

*[in] test_parameters Module-specific test parameters.

*[out] status Status of test at completion.

Returns:

*S_OK, S_GENERAL_ERROR, S_NON_FUNCTIONAL, S_OPERATION_FAILED, S_POLICY_VIOLATION, S_MODULE_RESOURCES, S_HOST_RESOURCES, S_INVALID_STATE, S_ALARM_STATE, S_MODULE_NOT_AVAILABLE, S_TIMEOUT, S_NOT_AUTHENTICATED, S_NOT_AUTHORIZED, S_MODULE_IN_USE, S_INVALID_DATA_BUFFER, S_TOKEN_NOT_PRESENT, S_TOKEN_ADMIN_NOT_PRESENT

See also:

*CICM::TestManager::run_test_get_results for the version of this method that returns test results.

Method CICM::TestManager::run_test_get_results()

```
CICM::Status run_test_get_results(
        in  CICM::Buffer test_parameters,
        out CICM::Buffer test_results
);
```

Run module built-in tests specifying module-specific test parameters and receiving module-specific results or data for later evaluation from the test run.
Remarks:

*This method can only initiate tests that a client program can explicitly request (e.g., this method does not apply to a series of tests automatically initiated during a module's start-up sequence). Running built-in tests on some modules may result in an alarm if an error is encountered during the test run.

*The formats of the test parameters and test results values are not defined by CICM. The Implementation Conformance Statement (see Conformance and Extensions in [CICM]) MUST reference a standard format or define a module developer-specific format implemented by the module for these datatypes.

Parameters:

*[in] test_parameters Module-specific test parameters.

*[out] test_results Results of the test.

Returns:

*S_OK, S_GENERAL_ERROR, S_NON_FUNCTIONAL, S_OPERATION_FAILED, S_POLICY_VIOLATION, S_MODULE_RESOURCES, S_HOST_RESOURCES, S_INVALID_STATE, S_ALARM_STATE, S_MODULE_NOT_AVAILABLE, S_TIMEOUT, S_NOT_AUTHENTICATED, S_NOT_AUTHORIZED,

S_MODULE_IN_USE, S_INVALID_DATA_BUFFER, S_TOKEN_NOT_PRESENT, S_TOKEN_ADMIN_NOT_PRESENT

See also:

*CICM::TestManager::run_test for the version of this Method that returns a simple test status value.

9. Module Events

In certain cases it may be necessary for a module to asynchronously notify a client program of an event. Client programs can register to receive module notifications using CICM::ModuleEventManager. This manager enables a client program to register a listener (callback) method designed to handle a specific condition. The event method prototype provided by the client program is defined in CICM::ModuleEventListener. CICM::ModuleEventListener also defines the conditions that may result in a notification, including: hardware requires attention, alarm, key expired, and health test failure. In certain cases, a single event on a module may result in the generation of multiple notification messages. For example, CICM::ModuleEventListener::C_MODULE_ALARM may be followed by a CICM::ModuleEventListener::C_MODULE_NOT_READY_FOR_TRAFFIC.

9.1. Interface CICM::ModuleEventManager

Interface CICM::ModuleEventManager

interface ModuleEventManager {

CICM::ModuleEventManager supports registering and unregistering user-defined module event listeners (CICM::ModuleEventListener) for specific module events. It is accessed from CICM::CryptoModule via the CICM::CryptoModule::event_manager attribute.

9.1.1. CICM::ModuleEventManager Methods

Method CICM::ModuleEventManager::register()

```
CICM::Status register(
        in  CICM::ModuleEventListener::ModuleEvent event,
        in  CICM::ModuleEventListener listener
);
```

Registers the listener for a specific module event.
Remarks:

*The provided listener applies only to the client program from which the registration is initiated.

Parameters:

*[in] event Event for which this listener is being registered.

*[in] listener Listener that will receive a notification about the specified event.

Returns:

*S_OK, S_GENERAL_ERROR, S_NON_FUNCTIONAL, S_OPERATION_FAILED, S_MODULE_RESOURCES, S_HOST_RESOURCES, S_INVALID_STATE, S_ALARM_STATE, S_MODULE_NOT_AVAILABLE, S_TIMEOUT, S_NOT_AUTHENTICATED, S_NOT_AUTHORIZED, S_TOKEN_NOT_PRESENT, S_TOKEN_ADMIN_NOT_PRESENT, S_EVENT_REGISTERED, S_EVENT_NOT_SUPPORTED

Method CICM::ModuleEventManager::unregister()

```
CICM::Status unregister(
        in  CICM::ModuleEventListener::ModuleEvent event
);
```

Unregisters the listener associated with the specified event.
Remarks:

*The listener associated with the specified event is only unregistered from the client program from which this method is called.

Parameters:

*[in] event Event that will no longer have a listener associated with it.

Returns:

*S_OK, S_GENERAL_ERROR, S_NON_FUNCTIONAL, S_OPERATION_FAILED, S_MODULE_RESOURCES, S_HOST_RESOURCES, S_INVALID_STATE, S_ALARM_STATE, S_MODULE_NOT_AVAILABLE, S_TIMEOUT, S_NOT_AUTHENTICATED, S_NOT_AUTHORIZED, S_TOKEN_NOT_PRESENT, S_TOKEN_ADMIN_NOT_PRESENT, S_EVENT_NOT_REGISTERED

9.2. Interface CICM::ModuleEventListener

Interface CICM::ModuleEventListener

```
interface ModuleEventListener {
```

CICM::ModuleEventListener is unlike other CICM interfaces in that the interface is implemented by the developer of the client program to

service a specific module event and is then registered via the CICM::ModuleEventManager.

9.2.1. CICM::ModuleEventListener Types and Constants

Type CICM::ModuleEventListener::ModuleEvent

```
typedef CICM::UInt32 ModuleEvent;
```

Events for which a ModuleEventListener can be notified.
Constant CICM::ModuleEventListener::C_MODULE_ACCESS_TOKEN_INSERTED

```
const CICM::ModuleEventListener::ModuleEvent
        C_MODULE_ACCESS_TOKEN_INSERTED = 0x00002001;
```

Access token has been inserted.
Constant CICM::ModuleEventListener::C_MODULE_ACCESS_TOKEN_REMOVED

```
const CICM::ModuleEventListener::ModuleEvent
        C_MODULE_ACCESS_TOKEN_REMOVED = 0x00002002;
```

Access token has been removed.
Constant CICM::ModuleEventListener::C_MODULE_ALARM

```
const CICM::ModuleEventListener::ModuleEvent
        C_MODULE_ALARM = 0x00002004;
```

Module has entered an alarm state.
Constant CICM::ModuleEventListener::C_MODULE_FAILURE

```
const CICM::ModuleEventListener::ModuleEvent
        C_MODULE_FAILURE = 0x00002007;
```

Non-critical module failure detected.
Constant CICM::ModuleEventListener::C_MODULE_INSUFFICIENT_ENTROPY

```
const CICM::ModuleEventListener::ModuleEvent
        C_MODULE_INSUFFICIENT_ENTROPY = 0x00002008;
```

Insufficient entropy available to a cryptographic operation that requires it.
Constant CICM::ModuleEventListener::C_MODULE_KEY_EXPIRED_HARD

```
const CICM::ModuleEventListener::ModuleEvent
        C_MODULE_KEY_EXPIRED_HARD =     0x0000200B;
```

Specific key has expired; the module can optionally include identifying information about the specific key that expired in the event_data buffer that is provided with the event itself.
Constant CICM::ModuleEventListener::C_MODULE_KEY_EXPIRED_SOFT

```
const CICM::ModuleEventListener::ModuleEvent
        C_MODULE_KEY_EXPIRED_SOFT = 0x0000200D;

Specific key is within some system-defined delta of hard expiration;
the module can optionally include identifying information about the
specific key that is about to expire in the event_data buffer that is
provided with the event itself.
Constant CICM::ModuleEventListener::C_MODULE_KEY_FILL_COMPLETE

const CICM::ModuleEventListener::ModuleEvent
        C_MODULE_KEY_FILL_COMPLETE = 0x0000200E;

Key fill is complete.
Constant CICM::ModuleEventListener::C_MODULE_KEY_FILL_CONNECTED

const CICM::ModuleEventListener::ModuleEvent
        C_MODULE_KEY_FILL_CONNECTED = 0x00002010;

Key fill device has been connected.
Constant CICM::ModuleEventListener::C_MODULE_KEY_FILL_INITIATED

const CICM::ModuleEventListener::ModuleEvent
        C_MODULE_KEY_FILL_INITIATED = 0x00002013;

Key fill has been initiated.
Constant CICM::ModuleEventListener::C_MODULE_KEY_MEMORY

const CICM::ModuleEventListener::ModuleEvent
        C_MODULE_KEY_MEMORY = 0x00002015;

Out of internal key memory condition.
Constant CICM::ModuleEventListener::C_MODULE_KEY_PROTO_MESSAGE

const CICM::ModuleEventListener::ModuleEvent
        C_MODULE_KEY_PROTO_MESSAGE = 0x00002016;

Key protocol message is available; see the Key Protocol Management
documentation for additional information.
Constant CICM::ModuleEventListener::C_MODULE_LOG_FULL

const CICM::ModuleEventListener::ModuleEvent
        C_MODULE_LOG_FULL = 0x00002019;

Module log is full.
Constant CICM::ModuleEventListener::C_MODULE_LOG_NEAR_FULL

const CICM::ModuleEventListener::ModuleEvent
        C_MODULE_LOG_NEAR_FULL = 0x0000201A;

Module log is nearly full.
```

```
Constant CICM::ModuleEventListener::C_MODULE_LOGIN_FAILURE

const CICM::ModuleEventListener::ModuleEvent
        C_MODULE_LOGIN_FAILURE = 0x0000201C;

Attempted login failed.
Constant CICM::ModuleEventListener::C_MODULE_NOT_READY_FOR_TRAFFIC

const CICM::ModuleEventListener::ModuleEvent
        C_MODULE_NOT_READY_FOR_TRAFFIC = 0x0000201F;

Module is not able to process traffic.
Constant CICM::ModuleEventListener::C_MODULE_POWER_MGMT_ENTER

const CICM::ModuleEventListener::ModuleEvent
C_MODULE_POWER_MGMT_ENTER = 0x00002020;

Module has entered power management state.
Constant CICM::ModuleEventListener::C_MODULE_POWER_MGMT_EXIT

const CICM::ModuleEventListener::ModuleEvent
        C_MODULE_POWER_MGMT_EXIT = 0x00002023;

Module has exited power management state.
Constant CICM::ModuleEventListener::C_MODULE_POWER_OFF

const CICM::ModuleEventListener::ModuleEvent
        C_MODULE_POWER_OFF = 0x00002025;

Change in module power state to OFF detected.
Constant CICM::ModuleEventListener::C_MODULE_POWER_OFF_FAILURE

const CICM::ModuleEventListener::ModuleEvent
        C_MODULE_POWER_OFF_FAILURE = 0x00002026;

Disorderly change in module power state to OFF detected.
Constant CICM::ModuleEventListener::C_MODULE_POWER_ON

const CICM::ModuleEventListener::ModuleEvent
        C_MODULE_POWER_ON = 0x00002029;

Change in module power state to ON detected.
Constant CICM::ModuleEventListener::C_MODULE_READY_FOR_TRAFFIC

const CICM::ModuleEventListener::ModuleEvent
        C_MODULE_READY_FOR_TRAFFIC = 0x0000202A;

Module is ready to process traffic.
Constant CICM::ModuleEventListener::C_MODULE_REKEY_REQUEST
```

```
const CICM::ModuleEventListener::ModuleEvent
        C_MODULE_REKEY_REQUEST = 0x0000202C;
```

Rekey of a specific key is required; the module can optionally include identifying information about the specific key to be rekeyed in the event_data buffer that is provided with the event itself.
Constant CICM::ModuleEventListener::C_MODULE_TEST_FAILURE

```
const CICM::ModuleEventListener::ModuleEvent
        C_MODULE_TEST_FAILURE = 0x0000202F;
```

Module internal test has failed; the module can optionally include identifying information about the specific test that failed in the event_data buffer that is provided with the event itself.
Constant CICM::ModuleEventListener::C_MODULE_ZEROIZED

```
const CICM::ModuleEventListener::ModuleEvent
        C_MODULE_ZEROIZED = 0x00002031;
```

Module has been zeroized.

9.2.2. CICM::ModuleEventListener Methods

Method CICM::ModuleEventListener::event_occurred()

```
void event_occurred(
        in  CICM::ModuleEventListener::ModuleEvent event,
        in  CICM::Buffer event_data
);
```

Method implemented by client program that is called by the host runtime system to notify that a specific module event has occurred.
Remarks:

* An opaque data field with additional information about the event in a module-specific format MAY optionally be provided with the event itself. This field MAY be of length zero.

* The format of the event data value is not defined in this specification. The Implementation Conformance Statement (see Conformance and Extensions in [CICM]) MUST reference a standard format or define a module-specific format for this datatype.

Note:

* Because this method is called by the runtime system and not a client program, it does not return a status value.

Parameters:

*[in] event Event that occurred.

*[in] event_data Opaque data associated with the event (e.g., specific test that failed, key that will expire).

10. IANA Considerations

[RFC Editor: Please remove this section prior to publication.]
This document has no IANA actions.

11. Security Considerations

11.1. Unauthorized Usage

A cryptographic module is typically a controlled resource which requires appropriate authorization to use. Specific implementations may use a combination of hardware access tokens, usernames and passwords, access control lists, or other means.
CICM defines the TokenManager, UserManager, and LoginManager interfaces to facilitate with the management of authorized users and to provide authentication capabilities.

11.2. Inappropriate Usage

Although CICM does not define audit logs as a separate concept, the LogManager interface can conceivably provide enough information to act as a means for tracking inappropriate usage which is especially important for the operations that manage the module itself: managing users, updating the module software, and running the built-in tests. Additionally, manipulation of the module logs may undermine the value of the auditing countermeasure.

11.3. Denial of Service

As suggested by [RFC3552], implementers are advised to include mechanisms that mitigate against denial of service attacks. This is primarily an issue for modules that authenticate using a user name and password, although this may also be an issue for hardware access tokens.

12. References

12.1. Normative References

[RFC2119]	Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
[CICM]	Lanz, D. and L. Novikov, "Common Interface to Cryptographic Modules (CICM) [RFC Editor: Please update

	the RFC reference and date prior to publication.]", January 2011.
[CICM-KM]	Lanz, D. and L. Novikov, "Common Interface to Cryptographic Modules (CICM) Key Management [RFC Editor: Please update the RFC reference and date prior to publication.]", January 2011.
[IDL]	International Standards Organization, "Information technology — Open Distributed Processing — Interface Definition Language", ISO/IEC 14750:1999(E), March 1999.

12.2. Informative References

[RFC3552]	Rescorla, E. and B. Korver, "Guidelines for Writing RFC Text on Security Considerations", BCP 72, RFC 3552, July 2003.
[CICM-LM]	Lanz, D. and L. Novikov, "Common Interface to Cryptographic Modules (CICM) Logical Model [RFC Editor: Please update the RFC reference and date prior to publication.]", January 2011.
[CORBA]	Object Management Group, "Common Object Request Broker Architecture (CORBA) Specification, Version 3.1", January 2008.

Appendix A. IDL Definitions

```
module CICM {
  typedef CICM::CharString TokenRecord;
  typedef CICM::CharString ModuleRecord;

  interface TokenAssnIterator : CICM::Iterator {
    CICM::Status get_next(
      out CICM::TokenRecord token_rec_ref );
  };

  interface ModuleAssnIterator : CICM::Iterator {
    CICM::Status get_next(
      out CICM::ModuleRecord module_rec_ref );
  };

  interface TokenManager {
    readonly attribute CICM::ModuleAssnIterator
      module_association_iterator;

    readonly attribute CICM::TokenAssnIterator
      token_association_iterator;

    CICM::Status associate(
      out CICM::ModuleRecord module_rec,
      out CICM::TokenRecord token_rec );

    CICM::Status disassociate();

    CICM::Status disassociate_missing_module(
      in  CICM::ModuleRecord module_rec );

    CICM::Status disassociate_missing_token(
      in  CICM::TokenRecord token_rec );
  };

  typedef CICM::CharString UserId;
  typedef CICM::CharString RoleId;

  interface UserIdIterator : CICM::Iterator {
    CICM::Status get_next(
      out CICM::UserId user_id );
  };

  interface RoleIdIterator : CICM::Iterator {
    CICM::Status get_next(
      out CICM::RoleId role_id );
  };

  interface UserManager {
    readonly attribute CICM::UserIdIterator user_iterator;
    readonly attribute CICM::RoleIdIterator role_iterator;
```

```
  CICM::Status add(
    in  CICM::UserId user,
    in  CICM::CharString password );

  CICM::Status modify(
    in  CICM::UserId user,
    in  CICM::CharString password );

  CICM::Status remove(
    in  CICM::UserId user );

  CICM::Status associate(
    in  CICM::UserId user,
    in  CICM::RoleId role );

  CICM::Status disassociate(
    in  CICM::UserId user,
    in  CICM::RoleId role );
};

interface Login {
  CICM::Status logout();
};

interface LoginManager {
   CICM::Status login(
    in  CICM::UserId user,
    in  CICM::CharString password,
    out CICM::Login login_ref );

  CICM::Status login_auth_data(
    in  CICM::UserId user,
    in  CICM::CharString password,
    in  CICM::Buffer auth_data,
    out CICM::Login login_ref );
};

typedef CICM::CharString PackageId;

interface Package {
  typedef CICM::UInt32 PackageType;
  const CICM::Package::PackageType
    C_PACKAGE_ALGORITHM = 0x00006054;

  const CICM::Package::PackageType
    C_PACKAGE_CONFIG_PARAMS = 0x00006057;

  const CICM::Package::PackageType
    C_PACKAGE_FPGA_IMAGE = 0x00006058;
```

```
  const CICM::Package::PackageType
    C_PACKAGE_POLICY_DB = 0x0000605B;

  const CICM::Package::PackageType
    C_PACKAGE_SOFTWARE = 0x0000605D;

  readonly attribute CICM::PackageId id;

  CICM::Status activate();
  CICM::Status deactivate();
  CICM::Status delete();
};

interface PackageIterator : CICM::Iterator {
  CICM::Status get_next(
    out CICM::Package package_ref );
};

interface PackageImporter {
  CICM::Status import_segment(
    in  CICM::Buffer package_data );

  CICM::Status complete(
    out CICM::Package package_ref );

  CICM::Status abort();
};

interface PackageManager {
  readonly attribute CICM::PackageIterator package_iterator;

  CICM::Status import_package(
    in  CICM::Package::PackageType package_type,
    out CICM::PackageImporter importer_ref );

  CICM::Status import_package_with_key(
    in  CICM::Package::PackageType package_type,
    in  CICM::SymKey key_ref,
    out CICM::PackageImporter importer_ref );

  CICM::Status get_package_by_id(
    in  CICM::PackageId package_id,
    out CICM::Package package_ref );

  CICM::Status reencrypt_software();
};

interface LogEntry {
  readonly attribute CICM::UserId user_id;
```

```
  readonly attribute CICM::RoleId role_id;
  readonly attribute CICM::CharString message;
  readonly attribute CICM::CharString date_time;

  CICM::Status delete();
};

interface LogEntryIterator : CICM::Iterator {
  CICM::Status get_next(
    out CICM::LogEntry log_entry_ref );
};

interface LogManager {
  readonly attribute CICM::LogEntryIterator log_entry_iterator;

  CICM::Status retrieve(
    out CICM::Buffer log_ref );

  CICM::Status destroy();
};

interface TestManager {
  typedef CICM::UInt32 Status;
  const CICM::TestManager::Status C_TEST_SUCCESS = 0x00006062;
  const CICM::TestManager::Status C_TEST_FAILURE = 0x00006064;

  CICM::Status run_test(
    in  CICM::Buffer test_parameters,
    out CICM::TestManager::Status status );

  CICM::Status run_test_get_results(
    in  CICM::Buffer test_parameters,
    out CICM::Buffer test_results );
};

interface ModuleEventListener {
  typedef CICM::UInt32 ModuleEvent;
  const CICM::ModuleEventListener::ModuleEvent
    C_MODULE_ACCESS_TOKEN_INSERTED = 0x00002001;

  const CICM::ModuleEventListener::ModuleEvent
    C_MODULE_ACCESS_TOKEN_REMOVED = 0x00002002;

  const CICM::ModuleEventListener::ModuleEvent
    C_MODULE_ALARM = 0x00002004;

  const CICM::ModuleEventListener::ModuleEvent
    C_MODULE_FAILURE = 0x00002007;

  const CICM::ModuleEventListener::ModuleEvent
```

```
  C_MODULE_INSUFFICIENT_ENTROPY = 0x00002008;

const CICM::ModuleEventListener::ModuleEvent
  C_MODULE_KEY_EXPIRED_HARD = 0x0000200B;

const CICM::ModuleEventListener::ModuleEvent
  C_MODULE_KEY_EXPIRED_SOFT = 0x0000200D;

const CICM::ModuleEventListener::ModuleEvent
  C_MODULE_KEY_FILL_COMPLETE = 0x0000200E;

const CICM::ModuleEventListener::ModuleEvent
  C_MODULE_KEY_FILL_CONNECTED = 0x00002010;

const CICM::ModuleEventListener::ModuleEvent
  C_MODULE_KEY_FILL_INITIATED = 0x00002013;

const CICM::ModuleEventListener::ModuleEvent
  C_MODULE_KEY_MEMORY = 0x00002015;

const CICM::ModuleEventListener::ModuleEvent
  C_MODULE_KEY_PROTO_MESSAGE = 0x00002016;

const CICM::ModuleEventListener::ModuleEvent
  C_MODULE_LOG_FULL = 0x00002019;

const CICM::ModuleEventListener::ModuleEvent
  C_MODULE_LOG_NEAR_FULL = 0x0000201A;

const CICM::ModuleEventListener::ModuleEvent
  C_MODULE_LOGIN_FAILURE = 0x0000201C;

const CICM::ModuleEventListener::ModuleEvent
  C_MODULE_NOT_READY_FOR_TRAFFIC = 0x0000201F;

const CICM::ModuleEventListener::ModuleEvent
  C_MODULE_POWER_MGMT_ENTER = 0x00002020;

const CICM::ModuleEventListener::ModuleEvent
  C_MODULE_POWER_MGMT_EXIT = 0x00002023;

const CICM::ModuleEventListener::ModuleEvent
  C_MODULE_POWER_OFF = 0x00002025;

const CICM::ModuleEventListener::ModuleEvent
  C_MODULE_POWER_OFF_FAILURE = 0x00002026;

const CICM::ModuleEventListener::ModuleEvent
  C_MODULE_POWER_ON = 0x00002029;
```

```
    const CICM::ModuleEventListener::ModuleEvent
      C_MODULE_READY_FOR_TRAFFIC = 0x0000202A;

    const CICM::ModuleEventListener::ModuleEvent
      C_MODULE_REKEY_REQUEST = 0x0000202C;

    const CICM::ModuleEventListener::ModuleEvent
      C_MODULE_TEST_FAILURE = 0x0000202F;

    const CICM::ModuleEventListener::ModuleEvent
      C_MODULE_ZEROIZED = 0x00002031;

    void event_occurred(
      in  CICM::ModuleEventListener::ModuleEvent event,
      in  CICM::Buffer event_data );
  };

  interface ModuleEventManager {
    CICM::Status register(
      in  CICM::ModuleEventListener::ModuleEvent event,
      in  CICM::ModuleEventListener listener );

    CICM::Status unregister(
      in  CICM::ModuleEventListener::ModuleEvent event );
  };
};
```

Authors' Addresses

Daniel J. Lanz Lanz The MITRE Corporation EMail: dlanz@mitre.org

Lev Novikov Novikov The MITRE Corporation EMail: lnovikov@mitre.org