CHAPTER 13

Using the Core Client Registry

The Core Client Registry (CCR) manages the seamless installation, upgrade, patching and uninstall of Multiple Device Contoller (MDC) modules and the Core module itself.

The following topics describe how to use CCR to manage these tasks:

For more information about CCR, refer to the *Core Client Registry Software Unit Design Specification*, EDCS ENG-129945.

Understanding CCR

The Core Client Registry:

- Tracks overlaps in module requirements to prevent addition of modules that already exist.
- Maintains these requirements when modules are removed that are still needed for execution of other modules.
- Stores information needed to instantiate and run a module properly.

CCR tracks:

- Application locations.
- Application configurations.
- MDC client extension libraries.
- Namespaces.

- Initialization information.
- Registry and environmental entries.

CCR can:

- Add entries.
- Update configuration files.
- Remove entries.
- Track entry references.
- Increment references.
- Decrement the references.

You can access CCR from both C++ and Java applications, usually in the form of a servlet. CCR uses:

- XML to store the data.
- Generic C++ libraries and STL to allow porting to other operating systems.
- JNI to create a bridge from C++ to Java.
- Xerces to manage the DOM tree.

Note CCR is an information container only. It does not define the meaning of the content; this is up to the component developer. For example, an application can register user roles and the Core Admin Module (CAM) will pick them up. CCR will not understand the format and information needed for this kind of entry. The contract is between CAM and applications.

About the CCR Components

CCR has five major components, which are described in the following topics:

1. CCR Local System Data (LSD) Component
2. CCRProcess Component
3. CCRInterface Component
4. CCREntry Component
5. CCRResponse Component

CCR Local System Data (LSD) Component

LSD is the data structure that contains the registration information. It is the repository for all the entries tracked by the CCR. It is in the form of an XML file and is loaded by the CCRprocess. The XML file contains one root element called CCRRoot. CCRRoot will always have an element called *resources* that contains all of the resources that have been added to an MDC. All of the other elements under CCRRoot are MDCs.

Resources

If any resources are added to CCR, then CCRRoot will have a Resources element. Elements within Resources must adhere to specific rules. Each element is followed by a number that allows differentiation of the resources.

All immediate children of Resources will be called either Library, Custom, ApacheConf, or InitializationInfo followed by a number. For example, Custom1, Library2, ApacheConf5, InitializationInfo7.

These elements are:

- Library—describes any kind of library.
- InitializationInfo—describes any initialization information for the MDCs.
- ApacheConf—describes a change to an Apache configuration file.
- Custom—describes any other kind of resources.

All other elements under CCRRoot are assumed to be MDC elements:

- ExtensionLibraries.
- Configurations.
- Java.
- Notifications (this is deprecated).
- Logging.

All these elements (except for Logging) will contain children that conform to the Resources child element rules (Custom, Library, etc.).

All of these four types of child elements can have the following types of children.

- `<name>value</name>`
- `<data>value</data>`
- `<location>value</location>`
- `<custom_name>value</custom_name>`

These can be changed by privileged users.

The Logging child will have only one of the following elements:

```
<Location>log_file_name_w_location</Location>
```

However, it can have any number of these elements:

```
<categoryname priority="priority value"/>
```

Where the priority value can be DEBUG, INFO, WARN, ERROR, FATAL.

CCRProcess Component

This is the heart of CCR. It includes the code that loads the XML file as well as the code that manipulates the elements.

This is the thread that waits for requests to act on the LSD. Upon instantiation, the CCRProcess will either load up the existing LSD into a DOM tree, or if one is not available, it will create a new skeleton LSD that will become the new default one. It also makes backups of the LSD at certain intervals to prevent information from being lost or corrupted. It also creates backups when the LSD has been updated to allow recovery of previous versions if necessary.

There should only be one instance of CCRProcess per process. It uses the sync libraries to prevent deadlocking when multiple CCRInterfaces access the CCRProcess.

CCRInterface Component

The CCRInterface allows modules to modify the data based on their needs (like installation, removal). It is the access point for all CCR functionality. The CCRInterface starts the CCRProcess (if it has not been started already) and it also communicates requests to it. JNI is used to provide the servlets with a means to access the daemon from Java.

C++ client uses a CCRInterface object to interact with the CCR.

Two objects are primarily used with CCRInterface: CCREntry and CCRResponse.

CCREntry Component

Most CCRInterface functions will involve the addition, subtraction, or manipulation of CCREntry objects. These basically contain a list of `std::string` objects that will allow the CCRProcess to find entries or to properly place the entries with the LSD. It is important to use `std::string` objects as this will provide for easy translation between Java and C++. CCREntry also provides encryption & decryption capabilities

There are three important fields that help CCR find a CCREntry object within the DOM tree.

- rootElement–This field describes either the name of the MC, or it is *resources*. It has getter/setter methods.
- subElement–This field describes the child element of the rootElement. If the root is an MC it could be Libraries, etc. If it was Resources then it could be Custom1, etc.
- type–This field is one of the allowable child types for Resources.

CCRResponse Component

This contains the response information for many of the CCRInterface retrieval function calls. It has a Java corollary.

It contains a success value and a vector full of CCREntry objects that were retrieved.

About CCR System Flow

This topic provides an overview of the major functions of CCR:

Adding an LSD Entry (Installation)

The requesting process determines whether or not the CCRProcess has been started.

It creates an CCREntry with the following fields:

- Rootcomponent: The tree within the LSD where the new entry should reside.
- Subdirectory: The subdirectory where the entry will reside within the rootcomponent.
- Resourcetype: The type of resource that is being stored.
- Resourcedata: An actual string representation of the resource.

The CCREntry is passed to the CCRInterace method that handles the addition of entries.

The existing references are searched to determine whether the entry already exists. If it does, it is added, and the resource reference count is incremented. If it does not exist, it is added, and the resource is also added with the new reference and reference count of 1.

Removing an LSD Entry (Uninstall)

The requesting process determines whether or not the CCRProcess has been started.

It creates an CCREntry with the following fields:

- Rootcomponent: The tree within the LSD where the entry resides.
- Subdirectory: The subdirectory where the entry resides within the rootcomponent.
- resourcetype: The type of resource that is being removed.

The CCREntry is passed to the CCRInterace method that handles the removal of entries. If the resource is referenced by only one rootcomponent, then it is completely removed. Otherwise the resource's reference count is decremented by one and the actual reference of the component below the resource is removed.

Modifying an LSD Entry (Patching/Upgrading)

The requesting process determines whether or not the CCRProcess has been started. The requesting process then locates the entry to be updated. It either creates or is given the entry information.

The entry is updated. First the current entry is removed. If the old entry was the last reference to the resource, that resource is removed.

The existing references are searched to determine whether the entry already exists. If it does, it is added, and the resource reference count is incremented. If it does not exist, it is added, and the resource is also added with the new reference and reference count of 1.

Retrieving an LSD Entry

Pertinent information in the retrieval of the entry is entered into an CCREntry.

The retrieval request is submitted. If the information in the entry correlates to more than one entry in the repository, several entries will be returned in a `std::list` object. Otherwise the `std::list` object will only contain one entry. If no entry is found, the `std::list` object will be empty.

About CCR Data Structures

This topic describes the data structures for the following:

- Local System Data (LSD) Data Structure
- CCREntry Data Structure
- CCRResponse Data Structure

Local System Data (LSD) Data Structure

The LSD is an XML document that will contain all of the relevent registration information. There will be a root element which will contain:

- Core registration information
- MDC registration information
- Resources element

All of the leafs of the Core and MDCs will refer to elements within the resources subtree. These include location data, Apache configuration data, library data, initialization data, Java libraries, logging information, notification information, and any custom data that an MDC might need. The resources subtree will maintain the data, location, reference count, and the specific reference of each resource.

Whenever a new resource is to be added to the Core or an MDC, the resource subtree should be checked to see if there is a duplicate resource already available. If the resource is already available, the reference in the Core or MDC branch should point to the existing resource, the reference count of the resource should be incremented and the MDC that is referencing the resource should be added as a child of the resource. Otherwise, a new resource is added and the MDC or Core will point to that.

```
- <CCRRoot>
    - <Resources>
    - <Library1>
          <Name>ite-nosd.dll</Name>
          <Location>c:\ismg\core\libs</Location>
          <ReferenceCount>1</ReferenceCount>
        - <References>
          <Core />
          </References>
      </Library1>
    - <Library2>
          <Name>rulesd.dll</Name>
          <Location>c:\ismg\PIX\libs</Location>
```

```
      <ReferenceCount>1</ReferenceCount>
    - <References>
      <PIX />
      </References>
  </Library2>
- <Library3>
      <Name>configured.dll</Name>
      <Location>c:\ismg\PIX\libs</Location>
      <ReferenceCount>1</ReferenceCount>
    - <References>
      <PIX />
      </References>
  </Library3>
- <Library4>
      <Name>populated.dll</Name>
      <Location>c:\ismg\PIX\libs</Location>
      <ReferenceCount>1</ReferenceCount>
    - <References>
      <PIX />
      </References>
  </Library4>
- <Library5>
      <Name>sosd.dll</Name>
      <Location>c:\ismg\PIX\libs</Location>
      <ReferenceCount>1</ReferenceCount>
    - <References>
      <PIX />
      </References>
  </Library5>
- <Library6>
      <Name>statusd.dll</Name>
      <Location>c:\ismg\PIX\libs</Location>
      <ReferenceCount>1</ReferenceCount>
    - <References>
      <PIX />
      </References>
  </Library6>
- <Library7>
      <Name>pixd.dll</Name>
      <Location>c:\ismg\PIX\libs</Location>
      <ReferenceCount>1</ReferenceCount>
    - <References>
      <PIX />
      </References>
  </Library7>
- <Library8>
      <Name>translationd.dll</Name>
      <Location>c:\ismg\PIX\libs</Location>
      <ReferenceCount>1</ReferenceCount>
    - <References>
      <PIX />
      </References>
  </Library8>
- <Custom1>
      <Name>CustomResourceOne</Name>
      <Location>d:\ismg\core</Location>
      <ReferenceCount>1</ReferenceCount>
    - <References>
      <Core />
      </References>
  </Custom1>
- <Custom1>
      <Name>CustomResourceTwo</Name>
      <Location>d:\ismg\core</Location>
```

```
            <ReferenceCount>1</ReferenceCount>
          - <References>
            <Core />
            </References>
        </Custom1>
      - <Custom2>
            <Name>CustomResourceThree</Name>
            <Location>d:\ismg\core</Location>
            <ReferenceCount>1</ReferenceCount>
          - <References>
            <Core />
            </References>
        </Custom2>
        </Resources>
      - <Core>
            <Location>d:\ismg\core</Location>
            <Configurations />
            <Libraries />
          - <ExtensionLibraries>
            <Library1 />
            </ExtensionLibraries>
            <InitializationInfo />
      - <Custom>
            <Custom1 />
            <Custom1 />
            <Custom2 />
            </Custom>
        <Java />
        <Notifications />
      - <Logging>
            <Location>d:\ismg\core\log\core.log</Location>
            <axiom priority="DEBUG" />
            <eta priority="DEBUG" />
            <coreagent priority="DEBUG" />
        </Logging>
        </Core>
      - <PIX>
            <Location>d:\ismg\PIX</Location>
            <Configurations />
            <Libraries />
          - <ExtensionLibraries>
            <Library2 />
            <Library3 />
            <Library4 />
            <Library5 />
            <Library6 />
            <Library7 />
            <Library8 />
            </ExtensionLibraries>
            <InitializationInfo />
            <Custom />
            <Java />
            <Notifications />
            <Logging />
        </PIX>
</CCRRoot>
```

CCREntry Data Structure

`CCR::CCREntry` are strings that contain the information to specify an entry as well as string-based custom key value pairs.

```
std::string rootelement;
std::string subelement;
std::string resourcetype;
std::string resourcedata;
std::string resourcename;
std::string resourcelocation;
```

CCRResponse Data Structure

CCR::CCRResponse is a simple data structure that the CCRInterface will return that describes the result of a function execution.

```
int responsecode;
std::string description;
std:list<CCREntry*> returnedValues;

static final int SUCCESS = 0;
static final int FAILURE = 1;
static final int EXISTS = 2;
```

Using the CCR C++ API

This topic describes the functions for these components:

- CCRInterface Functions
- CCREntry Functions
- CCRResponse Functions

For the corresponding Java functions, see the "Using the CCR Java Interface" section on page 13-51.

CCRInterface Functions

The functions and fields of the CCRInterface component are:

`Cn::SharedPtr<CCRProcess> theProcess`

The process to which the interface will talk.

`CCRInterface::CCRInterface()`

Creates an interface. The `CCRProcess:: StartProcess()` function will be called in order to retrieve the process.

`CCRInterface::CCRInterface(std::string fileName)`

Creates an interface. The `CCRProcess::StartProcess(fileName)` function will be called in order to retrieve the process.

CCRInterface::CCRInterface(Cn::SharedPtr<CCRProcess> theProcess)

Creates an interface with an already existing process. As the interface should be the only one connecting to the process, this constructor might not be necessary, desired, or needed.

CCRResponse CCRInterface::addEntry(CCREntry* theEntry)

Adds a new entry to the registration repository. If the entry exists, its reference count gets incremented. It takes the following parameters:

Parameter Name	Type	Purpose
theEntry	CCREntry*	The new entry to be added.
Return	CCRResponse	The response to the addEntry request.

CCRResponse CCRInterface::removeEntry(CCREntry* theEntry)

Removes an entry from the repository. If it is the last reference to that entry, it is removed. Otherwise, the reference count is decremented. It takes the following parameters:

Parameter Name	Type	Purpose
theEntry	CCREntry*	The entry to be removed.
Return	CCRResponse	The response to the removeEntry request.

CCRResponse CCRInterface::updateEntry(CCREntry* theEntry, CCREntry* newEntry)

Updates an entry in the repository. It takes the following parameters:

Parameter Name	Type	Purpose
theEntry	CCREntry*	The entry to be updated.
newEntry	CCREntry*	The new entry information. Exisiting references should be checked and reference counts should be decremented/ incremented accordingly.
Return	CCRResponse	The response to the updateEntry request.

CCRResponse CCRInterface::retrieveEntry(CCREntry* theEntry)

Retrieves the complete info for an entry. It takes the following parameters:

Parameter Name	Type	Purpose
theEntry	CCREntry*	The entry to retrieve.
Return	CCRResponse	The response to the retrieveEntry request.

CCRResponse CCRInterface::retrieveEntriesOfType(CCREntry* theEntry)

Retrieves all entries of the specific type. If there is no rootelement value, it should return the entries from the resources directory. It takes the following parameters:

Parameter Name	Type	Purpose
theEntry	CCREntry*	The entry type to return.
Return	CCRResponse	The response to the retrieveEntriesOfType request.

`CCRResponse CCRInterface::applyTomcatConfiguration(CCREntry* theEntry)`

Will apply the `tomcatconfig` entry to the appropriate Tomcat config file. It takes the following parameters:

Parameter Name	Type	Purpose
theEntry	CCREntry*	The entry that describes the proper `tomcatconfig`.
Return	CCRResponse	The response to the applyTomcatConfiguration request.

`CCRResponse CCRInterface::removeTomcatConfiguration(CCREntry* theEntry)`

Will remove the `tomcatconfig` entry from the appropriate Tomcat config file. It takes the following parameters:

Parameter Name	Type	Purpose
theEntry	CCREntry*	The entry that describes the proper `tomcatconfig`.
Return	CCRResponse	The response to the removeTomcatConfiguration request.

`boolean CCRInterface::entryExists(CCREntry* theEntry)`

Determines whether or not an entry already exists in the repository. It takes the following parameters:

Parameter Name	Type	Purpose
theEntry	CCREntry*	The entry to examine.
Return	boolean	True if it exists. Otherwise, false.

`int CCRInterface::getEntryReferenceCount(CCREntry* theEntry)`

Determines the number of reference counts of a particular entry. It takes the following parameters:

Parameter Name	Type	Purpose
theEntry	CCREntry*	The entry to examine.
Return	int	The number of references of the entry.

`CCRResponse* CCRInterface::addNotification(std::string mdc, std::string location, std::string protocol)`

Adds a notifcation entry to an MDC in the CCR. It takes the following parameters:

Parameter Name	Type	Purpose
mdc	std::string	The MDC that will have the new notification.
location	std::string	The notification source.
protocol	std::string	The protocol that the notification will use.
Return	CCRResponse	The response to the addEntry request.

CCRResponse* CCRInterface::addMDC(CCREntry* theEntry)

Adds a new MDC to the CCR. It takes the following parameters:

Parameter Name	Type	Purpose
theEntry	CCREntry*	The new MDC to be added.
Return	CCRResponse	The response to the addMDC request.

CCRResponse* CCRInterface::setLoggingLocation(std::string mdc, std::string location)

Sets the location and file name for logging messages within an MDC. It takes the following parameters:

Parameter Name	Type	Purpose
mdc	std::string	The MDC that will have the new logging location.
location	std::string	The logging source. Should include the file name and full path.
Return	CCRResponse	The response to the setLoggingLocation request.

CCRResponse* CCRInterface::addLoggingCategory(std::string mdc, std::string name, std::string priority)

Adds a new category for logging within an MDC. It takes the following parameters:

Parameter Name	Type	Purpose
mdc	std::string	The MDC that will have the new logging category.
name	std::string	The name of the new category.
priority	std::string	The priority that the new category will use.
Return	CCRResponse	The response to the addLoggingCategory request.

CCRResponse* CCRInterface::updateLoggingCategory(std::string mdc, std::string name, std::string priority)

Updates a category's priority within an MDC. It takes the following parameters:

Parameter Name	Type	Purpose
mdc	std::string	The MDC that will have the updated logging category.
name	std::string	The name of the category.

Parameter Name	Type	Purpose
priority	`std::string`	The priority to which the category will be updated.
Return	CCRResponse	The response to the updateLoggingCategory request.

`std::string CCRInterface::getClassPath(std::string MDCName)`

Returns a ";" separated classpath for an MDC. It takes the following parameters:

Parameter Name	Type	Purpose
MDCName	`std::string`	The MDC that will have its classpath returned.
Return	`std::string`	A ";" separated list of Java libraries and directories.

`std::string CCRInterface::getLibraryPath(std::string MDCName)`

Returns a ";" separated path for an MDC's libraries. It takes the following parameters:

Parameter Name	Type	Purpose
MDCName	`std::string`	The MDC that will have its library path returned.
Return	`std::string`	A ";" separated list of libraries.

`std::string CCRInterface::getExtensionLibraryNames()`

Returns a ";" separated list of all of the extension libraries, used primarily for loading purposes. It takes the following parameter:

Parameter Name	Type	Purpose
Return	`std::string`	A ";" separated list of libraries.

`std::string CCRInterface::getMDCNames()`

Returns a ";" separated list of the MDC names within the CCR. It takes the following parameter:

Parameter Name	Type	Purpose
Return	`std::string`	A ";" separated list of MDC names.

`std::string CCRInterface::getMDCLoggingCategories(std::string mdc)`

Returns a ";" separated list of all logging categories of an MDC. It takes the following parameters:

Parameter Name	Type	Purpose
mdc	`std::string`	The MDC that will have its logging categories returned.
Return	`std::string`	A ";" separated list of of logging categories for an MDC.

`std::string CCRInterface::getLoggingLocation(std::string mdc)`

Returns the logging location for an MDC. It takes the following parameters:

Parameter Name	Type	Purpose
mdc	`std::string`	The MDC that will have its logging location returned.
Return	`std::string`	A MDC's logging location.

`void CCRInterface::writeTree()`

Writes and saves the CCR.

CCREntry Functions

The functions and fields of CCREntry are:

`private std::string rootElement`

Describes the root element that contains the entry.

`private std::string subElement`

Describes the sub element that contains the entry.

`private std::string resourceType`

Describes the type of the resource.

`private std::string resourceData`

Describes the data that the resource might contain.

`private std::string resourceName`

Describes the name of the resource.

`private std::string resourceLocation`

Describes the location of the resource.

`private CustomTagTable customTags`

A `hashtable` that describes any custom keys and values of the resource.

`private PasswordTable customPwds`

A `hashtable` that describes the passwords for any encrypted custom keys.

6. `private std::string m_dataEncryptPwd`

The password for an encrypted data value.

7. `private std::string m_locationEncryptPwd`

The password for an encrypted location value.

8. `private std::string m_nameEncryptPwd`

The password for an encrypted name value.

9. `CCREntry::CCREntry()`

Default constructor that sets all of the values of the entry to "".

10. `CCREntry::CCREntry(std::string stringForm)`

Constructor that creates a CCREntry from one that was created to a `std::string` using the toString method.

11. `CCREntry::CCREntry(std::string rootelement, std::string subelement, std::string resourcetype, std::string resourcedata)`

Constructor that sets all of the values of the entry to to the argument's values.

Parameter Name	Type	Purpose
rootelement	`std::string`	The new value of the root element.
subelement	`std::string`	The new value of the sub element.
resourcetype	`std::string`	The new value of the resource type.
resourcedata	`std::string`	The new value of the resource data.

12. `std::string CCREntry::getRootElement()`

Getter function for the root element value.

Parameter Name	Type	Purpose
Return	`std::string`	The root element value.

13. `std::string CCREntry::getSubElement()`

Getter function for the sub element value.

Parameter Name	Type	Purpose
Return	`std::string`	Returns the sub element value.

14. `std::string CCREntry::getResourceType()`

Getter function for the resource type value.

Parameter Name	Type	Purpose
Return	`std::string`	Returns the resource type value.

15. `std::string CCREntry::getResourceData()`

Getter function for the resource data value.

Parameter Name	Type	Purpose
Return	`std::string`	The resource data value.

16. `std::string CCREntry::getResourceLocation()`

Getter function for the resource location value.

Parameter Name	Type	Purpose
Return	`std::string`	The resource location value.

17. `std::string CCREntry::getResourceName()`

Getter function for the resource name value.

Parameter Name	Type	Purpose
Return	`std::string`	The resource name value.

18. `CustomTagTable CCREntry::getTagTable()`

Getter function for the resource custom tags and values (in a `hashtable`).

Parameter Name	Type	Purpose
Return	CustomTagTable	The resource `hashtable` of custom tags and values.

19. `bool CCREntry::isDataEncrypted()`

Determines whether the data value has been encrypted.

Parameter Name	Type	Purpose
Return	bool	Indicates whether or not the data value is an encrypted value.

20. `bool CCREntry::isLocationEncrypted()`

Determines whether the location value has been encrypted.

Parameter Name	Type	Purpose
Return	bool	Indicates whether or not the location value is an encrypted value.

21. `bool CCREntry::isNameEncrypted()`

Determines whether the name value has been encrypted.

Parameter Name	Type	Purpose
Return	bool	Indicates whether or not the name value is an encrypted value.

22. `bool CCREntry:: isCustomKeyValueEncrypted (std::string key)`

Determines whether the value reference by the key has been encrypted.

Parameter Name	Type	Purpose
key	std::string	The key of the value to be checked.
Return	bool	Indicates whether or not the value reference by the key has been encrypted

23. `CCREntry::setRootElement(std::string value)`

Setter function for the root element value.

Parameter Name	Type	Purpose
value	`std::string`	The new root element value.

24. `CCREntry::setSubElement (std::string value)`

Setter function for the sub element value.

Parameter Name	Type	Purpose
value	`std::string`	The new sub element value.

25. `CCREntry::setResourceType (std::string value)`

Setter function for the resource type value.

Parameter Name	Type	Purpose
value	`std::string`	The new resource type value.

26. `CCREntry::setResourceData(std::string value)`

Setter function for the resource data value.

Parameter Name	Type	Purpose
value	`std::string`	The new resource data value.

27. `CCREntry::setResourceLocation(std::string value)`

Setter function for the resource location value.

Parameter Name	Type	Purpose
value	`std::string`	The new resource location value.

28. `CCREntry::setResourceName(std::string value)`

Setter function for the resource name value.

Parameter Name	Type	Purpose
value	`std::string`	The new resource name value.

29. `CCREntry::setDataEncrypt(std::string value)`

Setter function for the resource data value encryption password.

Parameter Name	Type	Purpose
value	`std::string`	The new resource data value encryption password.

30. `CCREntry::setLocationEncrypt(std::string value)`

Setter function for the resource location value encryption password.

Parameter Name	Type	Purpose
value	`std::string`	The new resource name value encryption password.

31. `CCREntry::setNameEncrypt(std::string value)`

Setter function for the resource name value encryption password.

Parameter Name	Type	Purpose
value	`std::string`	The new resource name value encryption password.

32. `int CCREntry:: getCustomTagCount()`

Returns the number of custom tags in the resource.

Parameter Name	Type	Purpose
return	int	The number of custom tags.

33. `std::string CCREntry:: getCustomTagKey(int index)`

Returns the custom key at the index.

Parameter Name	Type	Purpose
index	int	The location of the key.
return	`std::string`	The key value.

34. `std::string CCREntry:: getCustomTagEntry(int index)`

Returns the custom value at the index.

Parameter Name	Type	Purpose
index	int	The location of the key.
return	`std::string`	The value.

35. `CCREntry::addCustomTag(std::string key, std::string value, std::string password = "")`

Setter for the resource data value. If the password value is not "", then that value is encrypted.

Parameter Name	Type	Purpose
key	`std::string`	The new resource custom key value.
value	`std::string`	The new resource custom value.

36. `std::string CCREntry::toString ()`

Converts an entry to a string representation. Used for the JNI interface.

Parameter Name	Type	Purpose
return	`std::string`	A string representation.

37. `int CCREntry::encryptName ()`

Encrypts the name value.

Parameter Name	Type	Purpose
return	int	The length of the string that was encrypted.

38. `int CCREntry::encryptData()`

Encrypts the data value.

Parameter Name	Type	Purpose
return	int	The length of the string that was encrypted.

39. `int CCREntry::encryptLocation()`

Encrypts the location value.

Parameter Name	Type	Purpose
return	int	The length of the string that was encrypted..

40. `int CCREntry::encryptCustomKeyValue(std::string key)`

Encrypts the key's value.

Parameter Name	Type	Purpose
key	`std::string`	The key whose value will be encrypted.
return	int	The length of the string that was encrypted..

41. `std::string encrypt(std::string toEncrypt, std::string password)`

Converts a string to an encrypted value based on the password.

Parameter Name	Type	Purpose
toEncrypt	`std::string`	The string to be encrypted.
password	`std::string`	The password to the will be the encryption key.
return	`std::string`	A string representation.

42. `std::string CCREntry::decryptData(std::string toDecrypt, int size)`

Decrypts a string based on the data password key.

Parameter Name	Type	Purpose
toDecrypt	`std::string`	The string to decrypt.
size	int	The intended size of the decrypted string.
return	`std::string`	A decrypted string.

43. `std::string CCREntry::decryptName(std::string toDecrypt, int size)`

Decrypts a string based on the name password key.

Parameter Name	Type	Purpose
toDecrypt	`std::string`	The string to decrypt.
size	int	The intended size of the decrypted string.
return	`std::string`	A decrypted string.

44. `std::string CCREntry::decryptLocation(std::string toDecrypt, int size)`

Decrypts a string based on the location password key.

Parameter Name	Type	Purpose
toDecrypt	`std::string`	The string to decrypt.
size	int	The intended size of the decrypted string.
return	`std::string`	A decrypted string.

45. `std::string CCREntry::decryptKeyValue(std::string toDecrypt, std::string key, int size)`

Decrypts a string based on the key's password key.

Parameter Name	Type	Purpose
toDecrypt	`std::string`	The string to decrypt.
key	`std::string`	The key whose password key will be used.
size	int	The intended size of the decrypted string.
return	`std::string`	A decrypted string.

CCRResponse Functions

The functions and fields of CCRResponse are:

`private std::string description`

Provide the response's description.

`private id responseID`

Describes the ID of the reponse. Possible values: SUCCESS, FAILURE, or EXISTS.

`std:vector<CCREntry*> returnedValues;`

Describes the entry or entries that returned with the response.

`static final int SUCCESS = 0;`

The ID of a successful response.

`static final int FAILURE = 1;`

The ID of an unsuccessful response.

`static final int EXISTS = 2;`

The ID of a response where the entry exists.

`CCRResponse::CCRResponse()`

Default constructor. The response ID and description are not set; use the setter functions.

`CCRResponse::CCRResponse(int type, std::string description)`

Constructor that sets the ID and description.

Parameter Name	Type	Purpose
type	int	The response's type. Possible values: SUCCESS, FAILURE, or EXISTS.
description	`std::string`	The response's description. Explains the type.

CCRResponse::CCRResponse(std::string stringForm)

Constructor that creates a response object from one that was converted into a string.

Parameter Name	Type	Purpose
stringForm	std::string	The representation of a CCRResponse object in a string form.

std::string CCRResponse::getDescription()

The getter function for the description value.

Parameter Name	Type	Purpose
Return	std::string	Returns the description value.

46. `int CCRResponse::getResponseID()`

The getter function for the response's type value.

Parameter Name	Type	Purpose
Return	int	Returns the type value.

47. `std::vector<CCREntry*> CCRResponse::getReturnedValues()`

The getter function for the response's returned entries.

Parameter Name	Type	Purpose
Return	std::vector<CCREntry*>	Returns the entries that are part of the response.

48. `CCRResponse::setDescription(std::string description)`

The setter function for the description value.

Parameter Name	Type	Purpose
description	std::string	The new value of the description.

49. `CCRResponse::setResponseID(int type)`

The setter function for the type value.

Parameter Name	Type	Purpose
type	int	The new value of the type.

50. `CCRResponse::setReturnedValues(std::vector<CCREntry*> values)`

The setter function for the type value.

Parameter Name	Type	Purpose
values	`std::vector<CCREntry *>`	The new value of the entries that belong with the response.

51. `CCRResponse::addReturnedValue(CCREntry* value)`

The setter function for the type value.

Parameter Name	Type	Purpose
value	CCREntry*	A new CCREntry to be added with the response.

52. `boolean CCRResponse::success()`

A convenience function that will tell the caller whether the response is a success response or not.

Parameter Name	Type	Purpose
Return	boolean	True if the response type is SUCCESS. Otherwise, false.

53. `boolean CCRResponse::failure()`

A convenience function that will tell the caller whether the response is a failure response or not.

Parameter Name	Type	Purpose
Return	boolean	True if the response type is FAILURE. Otherwise, false.

54. `boolean CCRResponse::exists()`

A convience function that will tell the caller whether the response is a exists response or not.

Parameter Name	Type	Purpose
Return	boolean	True if the response type is EXISTS. Otherwise, false.

55. `std::string CCRResponse::toString()`

Converts a CCRResponse object to a string form. Can be reconstituted using the constructor that takes a string as an argument.

Parameter Name	Type	Purpose
Return	`std::string`	The string form of the CCRResponse object.

Using the CCR API: Example

The following procedure shows how you can use the CCR API to modify and access the CCR.

Step 1 Initialize the CCR.

To access and modify the CCR, you must create a CCRInterface object. When this object is constructed, either an existing CCR will be loaded, or a new one will be created.

```
CCRInterface* theCCRInterface = new CCRInterface();
```

Step 2 Create an MDC.

In order to create a new MDC, first a CCREntry object should be created where the root element is the name of the MDC, and the location value is where the MDC is located. Then the addMDC method is called on the CCRInterface object.

```
CCREntry* newMDCEntry = new CCREntry();
newMDCEntry->setRootElement("NewMDCName");
newMDCEntry->setResourceLocation("d:\mdclocation");
theCCRInterface->addMDC(newMDCEntry);
```

Step 3 Create entries for extension libraries.

When the MDC is created, extension libraries pertaining to the MDC must be added. They are located in the MDC element and under the ExtensionLibraries element. The name of each entry will be the name of the library, and the location of the entry is where the library is located. Also, the entry type will be set to Library.

```
CCREntry* newExtensionLibrary = new CCREntry();
newExtensionLibrary->setRootElement("NewMDCName");
newExtensionLibrary->setSubElement("ExtensionLibraries");
newExtensionLibrary->setResourceName("mylibrary.dll");
newExtensionLibrary->setResourceType("Library");
theCCRInterface->addEntry(newExtensionLibrary);
```

Step 4 Add a new logging location for the MDC.

You must specify the MDC that will be changed as well as the complete path of the file in which you want to track the logs.

```
theCCRInterface->setLoggingLocation("TheMDC", "d:\mymdc\logs\mymdc.log");
```

Step 5 Add a new logging entry for the MDC.

You must add the MDC name, the category name, and the priority level. Priority levels can be DEBUG, INFO, WARN, ERROR, or FATAL.

```
theCCRInterface->addLoggingCategory("TheMDC", "NewLoggingCategory", "DEBUG");
```

Using the CCRAccess Client

CCRAccess is a command-line client that allows easy initialization and manipulation of the CCR. This is an executable and should be the primary way to enter information into CCR. To use CCRAccess, enter the following on the command line:

```
CCRAccess [-options] [-commands]
```

Where:

`[-options]` is one or more of the options or wildcards shown in Table 13-1.

`[-commands]` is one of the commands shown in Table 13-2.

Note that this client is also available as a DLL (see "Using the CCRAccess DLL" section on page 13-50).

Table 13-1 CCRAccess Options and Wildcards

Option/Wildcard	Description
`? -help`	Print this help message.
`-new`	Create a new Registration Daemon (CCR).
`-location <file>`	Specify an existing CCR saved in a file.
`-script <file>`	Load commands for the CCR from a script file.
`/e:<encryptionkey`	Encrypts data, name, location, or custom tag value of resource. If the key is in a file, `encryptionkey` should be `file:<filename>`. Otherwise, enter the string. This should be directly before the data, location, or name that you want to encrypt or before the tag of the custom value that you want to encrypt.
`/d:<encryptionkey`	Decrypts data, name, location, or custom tag value of resource. If the key is in a file, `encryptionkey` should be `file:<filename>`. Otherwise, enter the string. This should be directly before the data, location, or name that you want to decrypt or before the tag of the custom value that you want to decrypt.

Table 13-2 CCRAccess Commands

Command	Description
`-addResource`	Adds a resource. This command must be followed by `<RootElement> <SubElement> <ResourceType> <ResourceData> <ResourceLocation> <ResourceName>`. Custom values and keys can be added at the end. They must be added in pairs. Empty strings should be entered as "".
`-addNotification`	Adds a new notification entry to the CCR. This command must be followed by `<MDCName> <Location> <Protocol>`. No empty strings are allowed.
`-addLog`	Adds a new logging entry to the CCR. This command must be followed by `<MDCName> <Name> <Priority>`. No empty strings are allowed.
`-addLogLocation`	Adds the log location for an MDC. This command must be followed by `<MDCName> <Location>`. No empty strings are allowed.
`-updateLog`	Updates a new logging entry to the CCR. This command must be followed by `<MDCName> <Name> <Priority>`. No empty strings are allowed.
`-addMDC`	Adds a new MDC to the CCR. This command must be followed by `<MDCName> <MDCLocation>`. No empty strings are allowed.

Table 13-2 CCRAccess Commands (continued)

Command	Description
-removeMDC	Removes an MDC from the CCR. This command must be followed by <MDCName>. No empty strings are allowed.
-removeResource	Removes a resource. This command must be followed by <RootElement> <SubElement> <ResourceType> <ResourceData> <ResourceLocation> <ResourceName>. Custom values and keys can be added at the end. They must be added in pairs. Empty strings should be entered as "".
-getClasspath	Gets the classpath of the MDC. This command must be followed by <MDCName>. No empty strings are allowed.
-getLibraryPath	Gets the library path of the MDC. This command must be followed by <MDCName>. No empty strings are allowed.
-getLogCategories	Gets the logging categories of the MDC. This command must be followed by <MDCName>. No empty strings are allowed.
-getLogLocation	Gets the logging file location of the MDC. This command must be followed by <MDCName>. No empty strings are allowed.
-getELNames	Gets the extension libraries in CCR.
-getMDCNames	Gets the name of the MDC in CCR.
-getResource	Retrieves a resource. This command must be followed by <RootElement> <SubElement> ResourceType> <ResourceData> <ResourceLocation> <ResourceName>. Custom values and keys can be added at the end. They must be added in pairs. Empty strings should be entered as "".
-getData	Retrieves a resource's data. This command must be followed by <RootElement> <SubElement> <ResourceType> <ResourceData> <ResourceLocation> <ResourceName>.Custom values and keys can be added at the end. They must be added in pairs. Empty strings should be entered as "".
-getLocation	Retrieves a resource's location. This command must be followed by <RootElement> <SubElement> <ResourceType> <ResourceData> <ResourceLocation> <ResourceName>. Custom values and keys can be added at the end. They must be added in pairs. Empty strings should be entered as "".
-getName	Retrieves a resource's name. This command must be followed by <RootElement> <SubElement> <ResourceType> <ResourceData> <ResourceLocation> <ResourceName>. Custom values and keys can be added at the end. They must be added in pairs. Empty strings should be entered as "".
-getCustom	Retrieves a resource's custom tags and values. This command must be followed by <RootElement> <SubElement> <ResourceType> <ResourceData> <ResourceLocation> <ResourceName>. Custom values and keys can be added at the end. They must be added in pairs. Empty strings should be entered as "".
-getCustomTagValue	Retrieves a resource's tag's value. This command must be followed by <TagName> <RootElement> <SubElement> <ResourceType> <ResourceData> <ResourceLocation> <ResourceName>. Custom values and keys can be added at the end. They must be added in pairs. Empty strings should be entered as "".

Table 13-2 CCRAccess Commands (continued)

Command	Description
`-getResourcesOfType`	Retrieves a group of resources. This command must be followed by `<RootElement> <SubElement> <ResourceType> <ResourceData> <ResourceLocation> <ResourceName>`. Custom values and keys can be added at the end. They must be added in pairs. Empty strings should be entered as "".
`-entryExists`	Checks if a resource exists. This command must be followed by `<RootElement> <SubElement> <ResourceType> <ResourceData> <ResourceLocation> <ResourceName>`. Custom values and keys can be added at the end. They must be added in pairs. Empty strings should be entered as "".
`-getEntryRefCount`	Gets the number of MDCs that reference the resource. This command must be followed by `<RootElement> <SubElement> <ResourceType> <ResourceData> <ResourceLocation> <ResourceName>`. Custom values and keys can be added at the end. They must be added in pairs. Empty strings should be entered as "".

Scripting CCRAccess

CCRAccess can load and run scripts, allowing you to perform “bulk” manipulation of the CCR. Scripts are simple, separated lines, with each line a CCRAccess command or option.

For example:

```
-addMDC NewMDCName d:\mymdclocation;
-addResource NewMDCName ExtensionLibraries Library EMPTYSTRING d:\mymdclocation\libs
mylibrary.dll;
-addLogLocation TheMDC d:\mymdc\logs\mymdc.log;
-addLog TheMDC NewLoggingCategory DEBUG
```

This information can be stored in a file called `mdcsetup.script`. The following procedure is an example illustrating how you can use the CCRAccess command line interface and its scripting capabilities.

Step 1 Use CCRAccess commands. For example:

```
CCRAccess -addMDC NewMDCName d:\mymdclocation;
CCRAccess -addResource NewMDCName ExtensionLibraries Library EMPTYSTRING
d:\mymdclocation\libs mylibrary.dll;
CCRAccess -addLogLocation TheMDC d:\mymdc\logs\mymdc.log;
CCRAccess -addLog TheMDC NewLoggingCategory DEBUG
```

Step 2 Load a script. For example:

```
CCRAccess -script mdcsetup.script
```

Using the CCRAccess DLL

CCR includes ccraccess.dll, a dynamic link library that provides all of the same functionality as the ccraccess.exe client described in the "Using the CCRAccess Client" section on page 13-47. The DLL version is significantly quicker to call (by approximately four seconds per call) than the EXE version. CCRAccess is often called during product installations, so developers who need to trim install times will find it useful to replace existing ccraccess.exe calls with ccraccess.dll calls.

The CCRAccess DLL interface is defined with the following prototype:

```
int EXPORTED ccraccess(char* commandLineInputString, char* resultFileName);
```

Where:

- *commandLineInputString* is one or more of the command-line arguments available for ccraccess.exe, which are listed in Table 13-2 on page 13-47. Multiple commands can be distinguished using the '|' variable separator
- *resultFileName* is the name of the file where the result of execution is to be dumped.

Calls to ccraccess.dll always return a zero on success and non-zero on failure.

Follow the steps below to call ccraccess.dll from a Windows-platform install (rul) file. Example 13-1 shows sample code that follows all of these steps.

Step 1 Prepare a `commandLineInputString` parameter by setting it to a string containing the previously used ccraccess.exe's command line parameters. Concatenate each parameter with '|' in place of the space separator. For example: `"-addResource|Core|Custom|Custom|…. "`

Step 2 Prepare a `resultFileName` parameter by setting it to an empty string (if you are writing into CCR using `-addResource`) or to a valid file name (if you will be reading from CCR using `-getResource`).

Step 3 Load the library by calling the install library's utility function `loadCcrDll()`. Check for a zero return value to confirm that CCRlibrary is loaded properly.

Step 4 Make the call : `return_value = ccraccess.ccraccess(commandLineInputSting , resultFileName);`, where `return_value` holds the status of the call.

Step 5 If you are reading from CCR using `-getResource`, parse the output file to retrieve the results.

Step 6 Unload the library using the install library's utility function `UnloadCcrDll( )`.

Example 13-1 Using ccraccess.dll

```
function MODULENAME_postinstall()
{

#define CMD_CCR_CUSTOM_PREFIX_DLL       "-addResource|Core|Custom|Custom"
#define CMD_CCR_REG_HTTP_PORT_DLL
CMD_CCR_CUSTOM_PREFIX_DLL+"|"+G_Port+"|"+"EMPTYSTRING|HttpPort"
STRING resultFileName;

//loading CCRDLL and reporting in case of error

if (loadCcrDll() != 0) then
                szTit="CCRaccess Dll's Not Found";
                SetDialogTitle (DLG_MSG_INFORMATION,szTit);
```

```
                MessageBox("ccraccess Dll's are not found. Installation will
abort",SEVERE);
                SureDeleteFile(WINDISK+"\\CMFLOCK.TXT");
                abort;
 endif;

// Calling CCRDLL  API to set
nCcrVal=ccraccess.ccraccess(CMD_CCR_REG_HTTP_PORT_DLL, resultFileName);

// More lines using the ccraccess.dll.

UnloadCcrDll( );

}
```

Using the CCR Java Interface

The Java interface for CCR is virtually identical to the functions and fields described in the "Using the CCR C++ API" section on page 13-31. However, it does rely heavily on the `toString()` functionality of CCRResponse and CCREntry. Java users should use `com.cisco.core.ccr.CCRInterface` when manipulating CCR.

`com.cisco.core.ccr.CCREntry`

This class is identical to the CCREntry C++ class, with obvious language differences. The changes are as follows:

- `std::string` changes to `java.lang.String`
- `CustomTagTable` and `PasswordTable` change to `java.util.Hashtable`
- `std:vector` changes to `java.util.Vector`

Encryption is slightly different as the algorithms for this are on the C++ side. CCRInterface provides methods to handle this.

`com.cisco.core.ccr.CCRInterface`

This class is identical to the CCRInterface C++ class with obvious language differences.

The changes are as follows:

- `std::string` changes to `java.lang.String`
- `CustomTagTable` and `PasswordTable` change to `java.util.Hashtable`
- `std:vector` changes to `java.util.Vector`

Access to the CCR via Java should be done through this class.

There are some additional methods to handle decryption of a CCREntry:

- `public CCREntry decryptData(CCREntry entry, int size)`

  This method will decrypt the data value based on the entry's data password. You will need to know the size of the original data value.

- `public CCREntry decryptName(CCREntry entry, int size)`

  This method will decrypt the name value based on the entry's name password. You will need to know the size of the original name value.

- `public CCREntry decryptLocation(CCREntry entry, int size)`

  This method will decrypt the location value based on the entry's location password. You will need to know the size of the original location value.

- `public CCREntry decryptKeyValue(CCREntry entry, String key, int size)`

  This method will decrypt the key's value based on the entry's key password. You will need to know the size of the original key value.

`com.cisco.core.ccr.CCRResponse`

This class is identical to the CCRResponse C++ class, with obvious language differences. The changes are as follows:

- `std::string` changes to `java.lang.String`
- `CustomTagTable` and `PasswordTable` change to `java.util.Hashtable`
- `std:vector` changes to `java.util.Vector`

`JNICCRInterface`

`com.cisco.core.ccr.CCRInterface` uses this class to bridge the Java/C++ gap via JNI. It is possible to use this class for the interface, but it is recommended that you use the CCRInterface class in `com.cisco.core.ccr`.

Encrypting and Decrypting CCREntry Values

You can encrypt and decrypt CCREntry name, data, location, and custom entry values. You must provide a key in the form of a file name or string value in order to encrypt or decrypt a value. The following topics show how to perform encryption and decryption using all of the available access methods:

- Encrypting Entry Names
- Decrypting Entry Names
- Encrypting Entry Data
- Decrypting Entry Data
- Encrypting Entry Locations
- Decrypting Entry Locations
- Encrypting Custom Entries
- Decrypting Custom Entries

Encrypting Entry Names

You can encrypt entry names via the C++, Java, or CCRAccess interfaces.

To encrypt an entry name using the C++ interface:

Step 1 Create a valid CCREntry.

Step 2 Call `setNameEncrypt` with the key (password) with which you want to encrypt the name value.

Step 3 Call `addEntry` via CCRInterface.

The entry will be added to the CCR with its name value encrypted.

To encrypt an entry name using the Java interface:

Step 1 Create a valid CCREntry.

Step 2 Call `setNameEncrypt` with the key (password) with which you want to encrypt the name value.

Step 3 Call `addEntry` via CCRInterface.

The entry will be added to the CCR with its name value encrypted.

To encrypt an entry name using CCRAccess:

Step 1 Consider your key to be “mykey”.

Step 2 Call:

```
CCRAccess –addResource MyMDC MyResourceSubdirectory MyResourceType MyResourceData
MyResourceLocation /e:mykey MyResourceName <Any custom tag pairs>
```

The entry will be added to the CCR with its name value encrypted.

Decrypting Entry Names

You can decrypt entry names via the C++, Java, or CCRAccess interfaces.

To decrypt an entry name using the C++ interface:

Step 1 Create a valid CCREntry.

Step 2 Call `setNameEncrypt` with the key (password) with which you want to encrypt the name value. The name value should be “”.

Step 3 Call `retrieveEntry` via CCRInterface.

The entry retrieved will have the decrypted name value if the password was correct.

To decrypt an entry name using the Java interface:

Step 1 Create a valid CCREntry.

Step 2 Call `setNameEncrypt` with the key (password) with which you want to encrypt the name value. The name value should be “”.

Step 3 Call `retrieveEntry` via CCRInterface.

The entry retrieved will have the decrypted name value if the password was correct.

To decrypt an entry name using CCRAccess:

Step 1 Consider your key to be “mykey”.

Step 2 Call:

```
CCRAccess -getResource MyMDC MyResourceSubdirectory MyResourceType MyResourceData
MyResourceLocation /d:mykey “” <Any custom tag pairs>
```

The entry retrieved will have the decrypted name value if the password was correct.

Encrypting Entry Data

You can encrypt entry data via the C++, Java, or CCRAccess interfaces.

To encrypt an entry’s data using the C++ interface:

Step 1 Create a valid CCREntry.

Step 2 Call `setDataEncrypt` with the key (password) with which you want to encrypt the data value.

Step 3 Call `addEntry` via the CCRInterface.

The entry will be added to the CCR with its data value encrypted.

To encrypt an entry’s data using the Java interface:

Step 1 Create a valid CCREntry.

Step 2 Call `setDataEncrypt` with the key (password) with which you want to encrypt the data value.

Step 3 Call `addEntry` via CCRInterface.

The entry will be added to the CCR with its data value encrypted.

To encrypt an entry’s data using CCRAccess:

Step 1 Consider your key to be “mykey”.

Step 2 Call:

```
CCRAccess -addResource MyMDC MyResourceSubdirectory MyResourceType /e:mykey MyResourceData
MyResourceLocation MyResourceName <Any custom tag pairs>
```

The entry will be added to the CCR with its data value encrypted.

Decrypting Entry Data

You can decrypt an entry's data via the C++, Java, or CCRAccess interfaces.

To decrypt an entry's data using the C++ interface:

Step 1 Create a valid CCREntry.

Step 2 Call `setDataEncrypt` with the key (password) with which you want to encrypt the data value.

The data value should be "".

Step 3 Call `retrieveEntry` via CCRInterface.

The entry retrieved will have the decrypted data value if the password was correct.

To decrypt an entry's data using the Java interface:

Step 1 Create a valid CCREntry.

Step 2 Call `setDataEncrypt` with the key (password) with which you want to encrypt the data value.

The data value should be "".

Step 3 Call `retrieveEntry` via CCRInterface.

The entry retrieved will have the decrypted data value if the password was correct.

To decrypt an entry's data using CCRAccess:

Step 1 Consider your key to be "mykey".

Step 2 Call:

```
CCRAccess -getResource MyMDC MyResourceSubdirectory MyResourceType /d:mykey ""
MyResourceLocation MyResourceName <Any custom tag pairs>
```

The entry retrieved will have the decrypted data value if the password was correct.

Encrypting Entry Locations

You can encrypt an entry's location data via the C++, Java, or CCRAccess interfaces.

To encrypt an entry's location data using the C++ interface:

Step 1 Create a valid CCREntry.

Step 2 Call `setLocationEncrypt` with the key (password) with which you want to encrypt the location value.

Step 3 Call `addEntry` via CCRInterface.

The entry will be added to the CCR with its location value encrypted.

To encrypt an entry's location data using the Java interface:

Step 1 Create a valid CCREntry.

Step 2 Call `setLocationEncrypt` with the key (password) with which you want to encrypt the location value.

Step 3 Call `addEntry` via CCRInterface.

The entry will be added to the CCR with its location value encrypted.

To encrypt an entry's location data using CCRAccess:

Step 1 Consider your key to be "mykey".

Step 2 Call:

```
CCRAccess –addResource MyMDC MyResourceSubdirectory MyResourceType MyResourceData /e:mykey
MyResourceLocation MyResourceName <Any custom tag pairs>
```

The entry will be added to the CCR with its location value encrypted.

Decrypting Entry Locations

You can decrypt an entry's location data via the C++, Java, or CCRAccess interfaces.

To decrypt an entry's location data using the C++ interface:

Step 1 Create a valid CCREntry.

Step 2 Call `setLocationEncrypt` with the key (password) with which you want to encrypt the location value.

The location value should be "".

Step 3 Call `retrieveEntry` via CCRInterface.

The entry retrieved will have the decrypted location value if the password was correct.

To decrypt an entry's location data using the Java interface:

Step 1 Create a valid CCREntry.

Step 2 Call `setLocationEncrypt` with the key (password) with which you want to encrypt the location value.

The location value should be "".

Step 3 Call `retrieveEntry` via CCRInterface.

The entry retrieved will have the decrypted location value if the password was correct.

To decrypt an entry's location data using CCRAccess:

Step 1 Consider your key to be "mykey".

Step 2 Call:

```
CCRAccess –getResource MyMDC MyResourceSubdirectory MyResourceType MyResourceData /d:mykey
"" MyResourceName <Any custom tag pairs>
```

The entry retrieved will have the decrypted location value if the password was correct.

Encrypting Custom Entries

You can encrypt custom entry data via the C++, Java, or CCRAccess interfaces.

To encrypt custom entry data using the C++ interface:

Step 1 Create a valid CCREntry.

Step 2 Call `addCustomTag` with the key (password) with which you want to encrypt the key value as the third argument.

Step 3 Call `addEntry` via CCRInterface.

The entry will be added to the CCR with the custom tag value encrypted.

To encrypt custom entry data using the Java interface:

Step 1 Create a valid CCREntry.

Step 2 Call `addCustomTag` with the key (password) with which you want to encrypt the key value as the third argument.

Step 3 Call `addEntry` via CCRInterface.

The entry will be added to the CCR with the custom tag value encrypted.

To encrypt custom entry data using CCRAccess:

Step 1 Consider your key to be "mykey", the custom tag to be "MyCustomKey", and the value to be "MyCustomKeyValue".

Step 2 Call:

```
CCRAccess –addResource MyMDC MyResourceSubdirectory MyResourceType MyResourceData
MyResourceLocation MyResourceName /e:mykey MyCustomKey MyCustomKeyValue
```

The entry will be added to the CCR with its custom tag value encrypted.

Decrypting Custom Entries

You can decrypt custom entry data via the C++, Java, or CCRAccess interfaces.

To decrypt custom entry data using the C++ interface:

Step 1 Create a valid CCREntry.

Step 2 Call `addCustomTag` with the key (password) with which you want to decrypt the key value as the third argument.

The custom tag's value should be "".

Step 3 Call `retrieveEntry` via CCRInterface.

The entry will be retrieved from the CCR with the custom tag value decrypted.

To decrypt custom entry data using the Java interface:

Step 1 Create a valid CCREntry.

Step 2 Call `addCustomTag` with the key (password) with which you want to decrypt the key value as the third argument.

The custom tag's value should be "".

Step 3 Call `retrieveEntry` via CCRInterface.

The entry will be retrieved from the CCR with the custom tag value decrypted.

To decrypt custom entry data using CCRAccess:

Step 1 Consider your key to be "mykey" and the custom tag to be MyCustomKey MyCustomKeyValue.

Step 2 Call:

```
CCRAccess -getResource MyMDC MyResourceSubdirectory MyResourceType MyResourceData
MyResourceLocation MyResourceName /d:mykey MyCustomKey
""
```

The entry retrieved will have the decrypted key value if the password was correct.