Using DCR Service

The Device Credentials Repository (DCR) is a common repository of devices, their attributes and their credentials. A DCR system consists of:

- One or more DCR Servers that store and distribute device information
- Applications that use DCR APIs to access device information
- Device and Credentials Admin GUI accessible from the CiscoWorks Home Page.

When implemented, DCR provides:

- Easier, centralized access to device and credentials data.
- Secure device-data persistence, access and transport.
- Rationalized and controlled replication of device information, with less user-level reconciliation.
- Better integration with third-party and Cisco network-management applications.

DCR stores device information. It does not communicate with devices directly. The code that interacts directly with devices and fetches their data for storage in DCR is the responsibility of your application.

For more information about DCR, see:

- Device List and Credentials Repository Server Functional Specification
- DCR Deployment Scenarios, Use Cases and Guidelines for Applications
- Device List and Credentials Repository Master/Slave Functional Specification
- Device List and Credentials Repository Import Export Software Functional Specification

CiscoWorks users know DCR by the acronym DCA, after the Device and Credentials Admin GUI they use to update DCR device lists. Some developers also use the DCA acronym to refer to DCR.

The following sections describe how DCR works and the tasks you can perform in DCR Service:

- Understanding DCR
- Integrating Applications with DCR
- Common Classes Available in DCR NBAPI
- DCR Service Related Class
- APIs Available in DCR
- DCR Events
Understanding DCR

All network management applications need to store basic attributes of the devices they manage. This includes device credentials, such as SNMP community strings consisting of a user ID-and-password pair along with SNMPv3 AuthPriv support.

Devices and their credentials represent a special data management problem. They must be both:

- Shared
  
  Multiple copies of device data are wasteful. If there is no single device data store that can be shared, users of multiple network management applications must waste resources reconciling the data, either manually or by using automated tools that they must build and maintain.

- Secure
  
  Independent copies of credentials are insecure by definition since they must be manually copied among applications, or transmitted using insecure means. Insecure credentials are dangerous, as anyone with access to credentials may reconfigure devices in destructive ways.

The DCR solution enables multiple applications to share device lists and credentials using a client-server mechanism, with secure storage and communications.

**Note**

All DCR APIs are accessible only if DCRServer is running. All DCR access involving credentials information must be user-authenticated and task-authorized. All credentials data is encrypted while stored and during transport.

About DCR Features and Benefits

The features of DCR are that it:

- Stores device attributes and credentials. It permits users to attach custom data to devices, and permits default grouping of devices.

- Supports proxy device attributes, storage of IPv6 and SNMP v3 device with AuthPriv support and credential information, and assigns a unique, internally generated DCR Device ID to every device.

- Supports “unreachable devices”. These are devices known to be “on the shelf” and not yet deployed, or “phone home” devices in transit to their location during initial deployment.

  DCR allows users to import these devices and retain credentials for them although some critical information about them may not be available, initially.

- Supports device pre-provisioning. Applications are sometimes not aware of essential attributes of pre-provisioned devices, such as their host names or IP addresses. Using the DCR device_identity attribute, applications can uniquely identify pre-provisioned devices.

- Allows clients to add, modify, and delete DCR devices using the DCA GUI. DCR also permits users to populate the database by importing from many sources, and to export device data for use with third-party products such as HP OpenView and Netview.

- Uses several attributes to detect and stop the creation of duplicate devices. If a new device has (or an update causes an existing device to change to) a display_name or fully-qualified DNS name (i.e., host_name + domain_name) that is the same as an existing device, the operation will fail.

Keeps data secure. DCR encrypts credential data using a static encryption scheme that must be hard-coded into the methods used for encryption (considered acceptable by the security team).

Keeps communications secure. Access to device data using API calls or the DCA GUI is performed only by secured channel (HTTPS) and authorized clients. DCR sends events to applications using ESS (see section “Using Event Services Software” in SDK Developer's Guide for CiscoWorks Common Services 3.2 available at the location http://preview.cisco.com/en/US/docs/net_mgmt/ciscoworks_common_services_software/3.1/sdk/developer/guide/ess.html#wp999313.

Supports a full range of administrative features, including the DCA GUI, change logs and reports, and full compatibility with CWCS backup and restore.

Integrating Applications with DCR

- **DCR Tasks to Perform During Application Startup**
- **Common Classes Available in DCR NBAPI**

**DCR Tasks to Perform During Application Startup**

Every application using DCR should be sure to complete the following tasks at startup:

After start-up, your application should not process DCR events sent before the DCR_SERVER_START event.

You must get the current DCR transaction ID stored in the application database.

If the transaction ID is zero, then this is the first time the application has been started. In this case:

**Step 1**
Call the getNewDevices(long transactionId) API and get the device list from DCR.

**Step 2**
Update the application-managed device list from the DCR device list.

**Step 3**
Get the DCR Domain ID and store it in the application database.

**Step 4**
Derive the maximum transaction ID from the device list and update it in the application database.

If the transaction ID is not zero, fetch the current DCR Domain ID from the DCR Server and compare it with the DCR Domain ID stored in the application database.

If the Server and application DCR Domain IDs are different:

**Step 1**
Clean the application-managed device list, or perform any other required application-specific action, if necessary.

**Step 2**
Set the application transaction ID to zero.
### Common Classes Available in DCR NBAPI

Table 6-1 lists the common classes available in DCR NBAPI.

<table>
<thead>
<tr>
<th>Class</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Device</td>
<td>This class represents device objects. The class contains the commonly used attributes and DeviceIdentification. DeviceIdentification is a direct member. All other attributes are encapsulated as DeviceAttributeCollection. You can assign values to direct members (mostly used attributes) and DeviceAttributeCollection, if you have assigned different values to the same attribute. That is, if there are direct members as well as the attribute in DeviceAttributeCollection, the direct member will be given higher priority. DeviceIdentification value will not be taken into account when you perform Add operations.</td>
</tr>
<tr>
<td>DeviceAttributeType</td>
<td>This class is an enumerated class that contains all the attributes (Device Attributes) available in DCR</td>
</tr>
<tr>
<td>DeviceAttributeTypeCollection</td>
<td>This class contains an array of DeviceAttributeType (Device Attributes) and User Defined attributes as array of strings.</td>
</tr>
<tr>
<td>DeviceAttribute</td>
<td>This class represents individual attributes. The members of this object are attributeName, attributeValue, and description. Of these,</td>
</tr>
<tr>
<td></td>
<td>• attributeName is the DeviceAttributeType</td>
</tr>
<tr>
<td></td>
<td>• attributeValue is the String containing the value to attributes</td>
</tr>
<tr>
<td></td>
<td>• description is the String containing description about this attribute</td>
</tr>
<tr>
<td></td>
<td>The description is read only. That is if you set description field, your changes are ignored.</td>
</tr>
<tr>
<td>UserDefinedDeviceAttribute</td>
<td>This class represents user defined attributes. The members of this object are attributeName, attributeValue and description. Of these,</td>
</tr>
<tr>
<td></td>
<td>• attributeName is the String representing the user defined name</td>
</tr>
<tr>
<td></td>
<td>• attributeValue is the String containing the value of the attribute</td>
</tr>
<tr>
<td></td>
<td>• description is the String containing description about this attribute</td>
</tr>
<tr>
<td>DeviceAttributeCollection</td>
<td>This class represents collection of attributes. That is, collection of device attributes (DeviceAttribute) and user defined attributes (UserDefinedDeviceAttribute)</td>
</tr>
<tr>
<td>DeviceCollection</td>
<td>This class contains an array of devices.</td>
</tr>
</tbody>
</table>
DCR Service Related Class

- Creating the DCR Service Object
- Adding a Device to DCR
- Updating a DCR Device
- Adding and Updating Devices in Bulk
- Retrieving DCR Device Objects

Creating the DCR Service Object

DCRService is the main service class. An object of class DCRService:
- Represents a DCR Server instance to the application.
- Provides all the methods needed to access and modify data in the DCR Server instance.

To start using DCR, your application should instantiate a single DCRService object, as shown below:

```java
com.cisco.nm.nbi.cwcs.dcr.AxisServletSoapBindingStub
dcrLocator=new DCRServiceServiceLocator();
com.cisco.nm.nbi.cwcs.dcr.AxisServletSoapBindingStub
dcrService=(com.cisco.nm.nbi.cwcs.dcr.AxisServletSoapBindingStub)dcrLocator.getAxisServlet();
if(dcrService!=null) {
    System.out.println("Inside If Part ");
    ((javax.xml.rpc.Stub)dcrService)._setProperty("javax.xml.rpc.session.maintain", Boolean.TRUE);
    ((javax.xml.rpc.Stub)dcrService)._setProperty(HTTPConstants.HEADER_COOKIE, "JSESSIONID="+coreId +","+cookie.getCookie()); // Core ID and cookie.getCookie() are obtained from login method of SecurityService(Refer SecurityService, possible hyperlink here)
```

<table>
<thead>
<tr>
<th>Class</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DeviceIdentification</td>
<td>This class contains the unique String Identifier to Device. When you add a Device to DCR it is assigned one unique identifier (DeviceIdentification).</td>
</tr>
<tr>
<td>DeviceIdentificationColl</td>
<td>This class contains an array of DeviceIdentification.</td>
</tr>
</tbody>
</table>
Adding a Device to DCR

This section contains sample codes for adding a standard (non-proxy) device to DCR using the addDevice() call.

- Adding a Standard Device
- Adding an AUS Device
- Adding an AUS-Managed Device
- Adding a DSBU Cluster
- Adding a DSBU Cluster Member
- Adding a CNS Configuration Engine (CNS Server)
- Adding a CNS Managed Device

Adding a Standard Device

This section provides sample code used for adding a Standard Device.

```java
DeviceAttribute[] devAt= new DeviceAttribute[10];
devAt[0]= new DeviceAttribute();
devAt[0].setAttributeName(DeviceAttributeType.dcr_device_type);
devAt[0].setAttributeValue("0");

DeviceAttributeCollection devAtrCol=new DeviceAttributeCollection();
devAtrCol.setAttributes(devAt);

//Set other attributes
Device device = new Device();
device.setAttributeCollection(devAtrCol);
//Assign most used/mandatory attribute
device.setDisplayName("router1");
device.setManagementIPAddress("1.2.3.2");
device.setSysObjectID("1234567");

// set other attributes
DeviceIdentification id = null;
try
{
    id = dcrService.addDevice(device);
}
catch (DCRException de)
{
    System.out.println("Error in adding Device "+
de.getFaultString());
    // Handle any error from DCR
}
catch (Exception e)
{
    System.out.println("Error in adding Device "+
e.getMessage());
    // Do application-specific things
}
System.out.println("ID for New Device = "+ id.getDeviceIdentifier ());
// This is the value that the application will retain
```

To add other types of devices using the same call, simply change the dcr_device_type value appropriately and set the corresponding attribute for each device, as shown in the following examples.
Adding an AUS Device

This section provides sample code used for adding an AUS Device.

```java
DeviceAttribute[] devAt = new DeviceAttribute[10];
devAt[0] = new DeviceAttribute();
devAt[0].setAttributeName(DeviceAttributeType.dcr_device_type);
devAt[0].setAttributeValue("2");

DeviceAttributeCollection devAtrCol = new DeviceAttributeCollection();
devAtrCol.setAttributes(devAt);

// Set other attributes

Device device = new Device();
device.setAttributeCollection(devAtrCol);
// Set the mandatory attributes
...
```

Adding an AUS-Managed Device

This section provides sample code used for adding an AUS-Managed Device.

```java
DeviceAttribute[] devAt = new DeviceAttribute[10];
devAt[0] = new DeviceAttribute();
devAt[0].setAttributeName(DeviceAttributeType.dcr_device_type);
devAt[0].setAttributeValue("1"); // Standard

devAt[1].setAttributeName("parent_aus_id");
devAt[1].setAttributeValue("999"); // Value of Parent AUS ID.

devAt[2].setAttributeName("device_identity");
devAt[2].setAttributeValue("999"); // This attribute is a string uniquely identifying the AUS-managed device

DeviceAttributeCollection devAtrCol = new DeviceAttributeCollection();
devAtrCol.setAttributes(devAt);

// Set other attributes

Device device = new Device();
device.setAttributeCollection(devAtrCol);
// Set the mandatory attributes
...

DeviceId id = null;
...
```

Adding a DSBU Cluster

This section provides sample code used for adding a DSBU Cluster.

```java
DeviceAttribute[] devAt = new DeviceAttribute[10];
devAt[0] = new DeviceAttribute();
devAt[0].setAttributeName(DeviceAttributeType.dcr_device_type);
devAt[0].setAttributeValue("1");

DeviceAttributeCollection devAtrCol = new DeviceAttributeCollection();
devAtrCol.setAttributes(devAt);

// Set other attributes

Device device = new Device();
device.setAttributeCollection(devAtrCol);
// Set the mandatory attributes
```
Adding a DSBU Cluster Member

This section provides sample code used for adding a DSBU Cluster Member.

```java
DeviceAttribute[] devAt = new DeviceAttribute[10];
devAt[0]=new DeviceAttribute();
devAt[0].setAttributeName(DeviceAttributeType.dcr_device_type);
devAt[0].setAttributeValue("1");

devAt[1].setAttributeName("parent.dsbu_id");
devAt[1].setAttributeValue("999"); //Value of Parent DSBU Cluster ID.

devAt[2].setAttributeName("dsbu_member_number");
devAt[2].setAttributeValue("1"); // The number of the DSBU Cluster member. This number
represents the order in which the device was added to the cluster.
DeviceAttributeCollection devAtrCol=new DeviceAttributeCollection();
devAtrCol.setAttributes(devAt);

//Set other attributes
Device device = new Device();
device.setAttributeCollection(devAtrCol);
//Set the Mandatory attributes
...
DeviceId id = null;
...
```

Adding a CNS Configuration Engine (CNS Server)

This section provides sample code used for adding a CNS Configuration Engine (CNS Server).

```java
DeviceAttribute[] devAt= new DeviceAttribute[10];
devAt[0]=new DeviceAttribute();
devAt[0].setAttributeName(DeviceAttributeType.dcr_device_type);
devAt[0].setAttributeValue("4");

DeviceAttributeCollection devAtrCol=new DeviceAttributeCollection();
devAtrCol.setAttributes(devAt);

//Set other attributes
Device device = new Device();
device.setAttributeCollection(devAtrCol);
//Set the mandatory attributes
...
```

Adding a CNS Managed Device

This section provides sample code used for adding a CNS Managed Device.

```java
DeviceAttribute[] devAt= new DeviceAttribute[10];
devAt[0]=new DeviceAttribute();
devAt[0].setAttributeName(DeviceAttributeType.dcr_device_type);
devAt[0].setAttributeValue("1");

devAt[1].setAttributeName("parent.cns_id");
devAt[1].setAttributeValue("999"); //Value of Parent CNS Server ID

devAt[2].setAttributeName("cns.config_id");
devAt[2].setAttributeValue("1"); // CNS Config ID of the device

devAt[3].setAttributeName("cns_image_id");
devAt[3].setAttributeValue("1"); // CNS Config ID of the device
```
Chapter 6      Using DCR Service

**CISCO CONFIDENTIAL**

```
devAt[4].SetAttributeName("cns_event_id");
devAt[4].SetAttributeValue("1"); // CNS Config ID of the device

DeviceAttributeCollection devAtrCol=new DeviceAttributeCollection();
devAtrCol.setAttributes(devAt);
//Set other attributes

Device device = new Device();
device. setAttributeCollection(devAtrCol);
//Set the Mandatory attributes
...
DeviceId id = null;
...
```

### Updating a DCR Device

You can use code such as the following to update attributes or credentials for any existing DCR Device.

```
DeviceIdentification deviceID = new DeviceIdentification("<known-device-id>");
Device device = new Device(deviceID);
// Set new attribute values
try {
    dcrService.updateDevice(device);
}
catch (DCRException de) {
    System.out.println("Error in updating Device " +
    de.getMessage());
    // Do application specific things
}
catch (Exception e) {
    System.out.println("Error in updating Device " +
    e.getMessage());
    // Do application specific things
}
```

### Adding and Updating Devices in Bulk

These APIs allow you to add or update more than one device object at a time:

- public DCRReturnValue addDevices(DeviceCollection devices) displays DCRException
- public DCRReturnValue updateDevices(DeviceCollection devices) displays DCRException

Both APIs return an object of class DCRReturnValue, which holds all information about the operation on each device.

The information is stored in two arrays. One for the DeviceID objects, and the other for Operation error codes.

The length of each array is always same as the number of Device objects you pass in the API call. The objects in these arrays correspond to each Device object in the devices array that you pass.
To get this information from the appropriate arrays in DCRReturnValue, use the following methods:

- public DeviceIdentification getDevIds(int index) returns the DeviceIdentification object at the index you pass. Use this method after the addDevices call to retrieve and process all the newly created Device IDs.

- public int getErrorCode(int index) returns the error code associated with the operation on the Device object at the index you pass.

The following code is typical for a bulk addition. A bulk update would use another method but have essentially the same structure.

```java
DCRReturnValues drv = null;

// Call add API and collect its output in above DCRReturnValues object
...

int nErrorCode;
if(drv != null)
{
    for(int nLoop = 0; nLoop < numberOfDevicesAdded; nLoop ++)
    {
        // check if device was added successfully
        // If return array contains valid DeviceId, this means that device was added successfully
        if(drv.getDevIds(nLoop) != null) continue; // Valid DeviceId returned
        // Otherwise retrieve error code
        nErrorCode = drv.getErrorCodes(nLoop);
    }
}
// Handle the error here ...
```  

Retrieving DCR Device Objects

The following example demonstrates how to retrieve selected DCR Device Objects and their data using a list of DCR Device IDs supplied by an update event.

```java
DeviceIdentification[] deviceIDs = <populate device ids...>
DeviceIdentificationCollection devIdCollection=new DeviceIdentificationCollection(deviceIDs);

DeviceAttributeType[] devAtrType=
    DeviceAttributeType.fromString("display_name"),
    DeviceAttributeType.fromString("host_name"),
    DeviceAttributeType.fromString("http_password"),
    DeviceAttributeType.fromString("snmp_v3_user_id"),
    DeviceAttributeType.fromString("management_ip_address"),
    
String[] usrAtrStrings=new String[3];
usrAtrStrings[0]=new String("user_defined_field_0");
usrAtrStrings[1]=new String("user_defined_field_2");
usrAtrStrings[2]=new String("user_defined_field_1");

DeviceAttributeTypeCollection devAtrTypeColl=new DeviceAttributeTypeCollection();
devAtrTypeColl.setAttributeNames(devAtrType);
devAtrTypeColl.setUdfAttributeNames(usrAtrStrings);
```
try {
    Device[] devices=dcrService.getDevices(devIdCollection, devAtrTypeColl);
} catch (DCRException de) {
    System.out.println("Error in getting Devices " + e.getFaultMessage());
    // Do application-specific things
} catch (Exception e) {
    System.out.println("Error in getting Device " + e.getMessage());
    // Do application specific things
}

**APIs Available in DCR**

Table 6-2 lists the APIs available in DCR Service.

<table>
<thead>
<tr>
<th>Returns</th>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DeviceIdentification</td>
<td>addDevice(Device dev)</td>
<td>Adds a given device into DCR</td>
</tr>
<tr>
<td>DCRReturnValue</td>
<td>addDevices(DeviceCollection devices)</td>
<td>Adds the given devices into DCR</td>
</tr>
<tr>
<td>DeviceAttributeCollection</td>
<td>getAttributeTypes()</td>
<td>Returns all the attributes defined in DCR</td>
</tr>
<tr>
<td>java.lang.String</td>
<td>getDCRDomainID()</td>
<td>Returns the DCR Domain ID</td>
</tr>
<tr>
<td>DeviceIdentificationCollection</td>
<td>getDeletedDevices(DeviceIdentificationCollection managedDeviceIds)</td>
<td>Gets the list of deleted devices from the given Deviceld list.</td>
</tr>
<tr>
<td>Device</td>
<td>getDevice(DeviceIdentification id)</td>
<td>Gets complete data for the specified device</td>
</tr>
<tr>
<td>Device</td>
<td>getDevice(DeviceIdentification id, DeviceAttributeTypeCollection attList)</td>
<td>Gets the details of given attributes and credentials from the DCR Database for given device ID.</td>
</tr>
<tr>
<td>DeviceIdentificationCollection</td>
<td>getDeviceIdentifiers()</td>
<td>Gets all device identifiers</td>
</tr>
<tr>
<td>DeviceCollection</td>
<td>getDevices()</td>
<td>Gets complete device data for all devices</td>
</tr>
<tr>
<td>DeviceCollection</td>
<td>getDevices(DeviceIdentificationCollection id)</td>
<td>Gets complete device details from the DCR Database for given device IDs</td>
</tr>
<tr>
<td>DeviceCollection</td>
<td>getDevices(DeviceIdentificationCollection id, DeviceAttributeTypeCollection attList)</td>
<td>Gets the details of given attributes and credentials from the DCR Database for given device IDs.</td>
</tr>
<tr>
<td>DeviceIdentificationCollection</td>
<td>getMatchingDevices(Device device)</td>
<td>Gets complete device details from the DCR Database of all the devices that match given device attribute values.</td>
</tr>
<tr>
<td>long</td>
<td>getMaxTransactionID()</td>
<td>Gets maximum value of transaction ID that is being used for DCR records.</td>
</tr>
</tbody>
</table>
CISCO CONFIDENTIAL

Table 6-2 APIs available in DCR Service (continued)

<table>
<thead>
<tr>
<th>Returns</th>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DeviceCollection</td>
<td>getMissingDCRDevicesInACS()</td>
<td>Gets devices not configured in ACS</td>
</tr>
<tr>
<td>DeviceCollection</td>
<td>getNewDevices(long transactionId)</td>
<td>Gets complete device details for devices from the DCR database that were added after the given transaction ID.</td>
</tr>
<tr>
<td>DeviceCollection</td>
<td>getUpdatedDevices(long transactionId)</td>
<td>Gets complete device details for devices from the DCR database that were updated after the given transaction ID.</td>
</tr>
<tr>
<td>DeviceAttributeCollection</td>
<td>getUserDefinedAttributes()</td>
<td>Gets user-defined attributes in DCR</td>
</tr>
<tr>
<td>java.lang.String</td>
<td>getVersion()</td>
<td>Gets version of DCR Webservice</td>
</tr>
<tr>
<td>boolean</td>
<td>isMasterDCR()</td>
<td>Checks if DCR being accessed is Master DCR</td>
</tr>
<tr>
<td>boolean</td>
<td>isMasterDCRRunning()</td>
<td>Checks if Master DCR is up and running</td>
</tr>
<tr>
<td>boolean</td>
<td>isRunning()</td>
<td>Checks if DCR being accessed is up and running</td>
</tr>
<tr>
<td>void</td>
<td>updateDevice(Device device)</td>
<td>Updates a given device into DCR</td>
</tr>
<tr>
<td>DCRReturnValue</td>
<td>updateDevices(DeviceCollection deviceCollection)</td>
<td>Updates given devices in DCR</td>
</tr>
</tbody>
</table>

DCR Events

Through EventService you need to subscribe for the DCREvent Topic. The topic of the DCREvent is cisco.mgmt.cw.cmf.dcr. The Events are populated as DCREventObject.

The members of the DCREventObjects are:

- long dcrTransactionId;
- String dcrDomainId;
- String udfName;
- String topic;
- String eventType;
- String sourceMachineName;
- String sourceApplication;
- String sourceApplicationVersion;
- String eventID;

See JavaDoc for more information about these fields.

The DCR Server does not listen to events broadcast by applications. DCR generates:

- Device events whenever there is a change in the attributes or credentials of devices stored in the DCR repository (e.g., device additions, imports, etc.) For details, see the About DCR Device Events section in SDK Developers Guide for CiscoWorks Common Services 3.2 at http://preview.cisco.com/en/US/docs/net_mgmt/ciscoworks_common_services_software/3.1/sdk/developer/guide/dcr.html#wp1149524
Chapter 6      Using DCR Service

DCR Events


- Restore events whenever DCR data is restored from backup. For details, see the About DCR Restore Events” section in SDK Developers Guide for CiscoWorks Common Services 3.2 at http://preview.cisco.com/en/US/docs/net_mgmt/ciscoworks_common_services_software/3.1/sdk/developer/guide/dcr.html#wp1149574

The normal process of database backup and restore can also generate combinations of these events and accompanying notifications. For a summary of these, see the “About DCR Events During Backup and Restore” section.

These notifications are not broadcast to DCR client applications. The events include all significant repository changes, such as device additions, deletions, credential updates, etc.

The details of event generation for DCR Slaves are the same as in DCR Master. From the application’s perspective:

- It will receive events from DCR when there is an update in DCR. The events can be for new, existing, or deleted devices in DCR.

- The application can fetch device lists and credentials data from DCR as needed.

When a new application (that does not contain any device credential data) is installed, the application can get the device list available from DCR and present the devices in a pick list. The user can then select the devices from the list for management.

Understanding DCR Domain ID and Transaction ID

The Master DCR Server in every Master/Slave DCR setup has its own DCR Domain ID. The DCR Domain ID identifies the logical DCR domain in which the Master and its Slaves participate. Your application must:

- Maintain the DCR Domain ID as part of its data
- Check to see if the DCR Domain ID has changed:

At application startup. You can do this by calling getDCRDomainID() during startup.

- Whenever the application receives a DCR_DATA_RESTORED or DCR_DATA_RESTORED_FROM_DIFFERENT_DOMAIN event.

You can do this by comparing the DCR Domain ID contained in the event with the DCR Domain ID stored in the application database.

- If the DCR Domain ID has changed with a DCR_DATA_RESTORED event, update the DCR Domain ID in the application database.
- If the DCR Domain ID has changed with a DCR_DATA_RESTORED_FROM_DIFFERENT_DOMAIN event, update the DCR Domain ID in the application database.

Clean up and refresh the device list.

Every DCR Server update of any kind (e.g., device additions, deletions, updates, etc.) has a transaction ID, which is a serial number. The DCR Transaction ID is the ID of the last transaction (e.g., device update, deletion, addition, etc.) an application conducted with its DCR Server.
Chapter 6  Using DCR Service

DCR Events

This transaction ID is maintained in the application database, and allows the application to determine whether updates occurred while the application was offline. To make use of it, your application must:

- Maintain the DCR Transaction ID in its own database.
- Check at application startup whether the last (or maximum) transaction ID recorded on the DCR Server is greater than the DCR Transaction ID recorded in the application database. You can get the last transaction ID for the Server by calling getMaxDcrTransactionID() during startup.
- Update the DCR Transaction ID in the application database, and get all the device updates since the last transaction recorded in the application, if the transaction ID on the server is higher than the DCR Transaction ID stored in the application.

DCR Device Events

DCR broadcasts events whenever devices are updated (including addition or deletion). Table 6-3 shows the data for all types of device events. Applications can use this data to decide whether to process the event or not, and to query DCR for details about the target devices.

<table>
<thead>
<tr>
<th>Device Change</th>
<th>Event Data</th>
</tr>
</thead>
</table>
| Added          | eventType=DEVICES_ADDED  
sourceApplication=DCRUI  
sourceApplicationVersion=1.0  
sourceMachineName=server1  
dcrTransactionId=556598     |
| Added in Bulk  | eventType=BULK_DEVICES_ADDED  
sourceApplication=DCRUI  
sourceApplicationVersion=1.0  
sourceMachineName=server1  
dcrTransactionId=556598     |
| Updated        | eventType=DEVICES_UPDATED  
sourceApplication=DCRUI  
sourceApplicationVersion=1.0  
sourceMachineName=server1  
dcrTransactionId=556598     |
| Updated in Bulk| eventType=BULK_DEVICES_UPDATED  
sourceApplication=DCRUI  
sourceApplicationVersion=1.0  
sourceMachineName=server1  
dcrTransactionId=556598     |

DCR Process Events

DCR broadcasts process events whenever the DCR Server starts, which is a common event. It also broadcasts events when the server changes modes, which happens much less often. A DCR process event consists of the events:

- Subject (also treated as the event ID).
  For example: Event Subject Name = "cisco.mgmt.cw.cmf.dcr".
- Data, which consists of:
  - The type of change. For example: MASTER_CHANGED_TO_SLAVE.
  - Identifying details for the DCR affected by the event. For example: "DCR Server 1".
The following example shows the event data for DCR Server start which is a normal process event.

Example: DCR Server Start Event Data

```plaintext
eventType= DCR_SERVER_START
sourceApplication= DCR Server
sourceApplicationVersion=1.0
sourceMachineName=server1
```

Table 6-4 shows the event data for the uncommon DCR mode-change process events. All of these events are broadcast because applications (such as OGS Server) need to know about them.

Most applications can ignore process events unless they are also accompanied by a "data restored" event, in which case applications must receive and process them (see the "About DCR Restore Events" section).

### Table 6-4 Data for Device Events

<table>
<thead>
<tr>
<th>Mode Change</th>
<th>Event Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>Standalone to Master</td>
<td>eventType=STANDALONE_CHANGED_TO_MASTER</td>
</tr>
<tr>
<td></td>
<td>dcrDomainId=Group123456</td>
</tr>
<tr>
<td></td>
<td>sourceApplication= DCR Server</td>
</tr>
<tr>
<td></td>
<td>sourceApplicationVersion=1.0</td>
</tr>
<tr>
<td></td>
<td>sourceMachineName=server1</td>
</tr>
<tr>
<td>Standalone to Slave</td>
<td>eventType=STANDALONE_CHANGED_TO_SLAVE</td>
</tr>
<tr>
<td></td>
<td>dcrDomainId=Group123456</td>
</tr>
<tr>
<td></td>
<td>sourceApplication= DCR Server</td>
</tr>
<tr>
<td></td>
<td>sourceApplicationVersion=1.0</td>
</tr>
<tr>
<td></td>
<td>sourceMachineName=server1</td>
</tr>
<tr>
<td>Slave to Master</td>
<td>eventType=SLAVE_CHANGED_TO_MASTER</td>
</tr>
<tr>
<td></td>
<td>dcrDomainId=Group123456</td>
</tr>
<tr>
<td></td>
<td>sourceApplication= DCR Server</td>
</tr>
<tr>
<td></td>
<td>sourceApplicationVersion=1.0</td>
</tr>
<tr>
<td></td>
<td>sourceMachineName=server1</td>
</tr>
<tr>
<td>Slave to Standalone</td>
<td>eventType=SLAVE_CHANGED_TO_STANDALONE</td>
</tr>
<tr>
<td></td>
<td>dcrDomainId=Group123456</td>
</tr>
<tr>
<td></td>
<td>sourceApplication= DCR Server</td>
</tr>
<tr>
<td></td>
<td>sourceApplicationVersion=1.0</td>
</tr>
<tr>
<td></td>
<td>sourceMachineName=server1</td>
</tr>
<tr>
<td>Master to Slave</td>
<td>eventType= MASTER_CHANGED_TO_SLAVE</td>
</tr>
<tr>
<td></td>
<td>dcrDomainId=Group123456</td>
</tr>
<tr>
<td></td>
<td>sourceApplication= DCR Server</td>
</tr>
<tr>
<td></td>
<td>sourceApplicationVersion=1.0</td>
</tr>
<tr>
<td></td>
<td>sourceMachineName=server1</td>
</tr>
<tr>
<td>Master to Standalone</td>
<td>eventType= MASTER_CHANGED_TO_STANDALONE</td>
</tr>
<tr>
<td></td>
<td>dcrDomainId=Group123456</td>
</tr>
<tr>
<td></td>
<td>sourceApplication= DCR Server</td>
</tr>
<tr>
<td></td>
<td>sourceApplicationVersion=1.0</td>
</tr>
<tr>
<td></td>
<td>sourceMachineName=server1</td>
</tr>
<tr>
<td>Slave to New Master</td>
<td>eventType= SLAVE_CHANGED_TO_NEW_MASTER</td>
</tr>
<tr>
<td></td>
<td>dcrDomainId=Group123456</td>
</tr>
<tr>
<td></td>
<td>sourceApplication= DCR Server</td>
</tr>
<tr>
<td></td>
<td>sourceApplicationVersion=1.0</td>
</tr>
<tr>
<td></td>
<td>sourceMachineName=server1</td>
</tr>
</tbody>
</table>
DCR Restore Events

DCR sends the restore events shown in following Table when DCR data is changed due to specific DCR mode changes or a data restore from backup. The restore events are sent in different conditions:

DCR DATA RESTORED FROM DIFFERENT DCR DOMAIN: DCR will send this event when:

- A DCR Server in Standalone or Master mode changes to Slave mode. The new Slave-mode DCR Server receives this event before it begins receiving DCR DATA RESTORED events.
- A DCR Server in Slave mode is reassigned to a DCR Master in a different domain.
- A DCR Server in Slave mode has its DCR Master change its host name.
- A CWCS restore is performed and the restore contains different domain data.

Applications receiving this event must:

- Alert the application administrator to act appropriately.
- Clean up the application-managed device list.
- Call the following methods to refresh the application device list.
  - `getNewDevices(long transactionId)`
    Gets complete device details for devices from the DCR database that were added after the given transaction ID.
  - `getUpdatedDevices(long transactionId)`
    Gets complete device details for devices from the DCR database that were updated after the given transaction ID.
  - `getDeletedDevices(DeviceIdentificationCollection managedDeviceIDs)`
    Gets the list of deleted devices from the given DeviceId list.

DCR DATA RESTORED: DCR will send this event when:

- A DCR Server in Standalone or Master mode changes to Slave mode. The new Slave-mode DCR Server receives this event after the DCR_DATA_RESTORED_FROM_DIFFERENT_DCR_DOMAIN event, and continues to receive it until the Slave-mode Server synchronizes with its new Master.
- When a DCR Server in Master mode changes its port and the user re-registers with a Slave with this Master.
- When a DCR Server in Slave mode changes its port and the user re-registers with this Slave with its Master.
- When DCR Server in Slave mode changes to Standalone or Master mode.

Applications receiving this event are expected to call the following APIs:

- `getNewDevices(long transactionId)`
  Gets complete device details for devices from the DCR database that were added after the given transaction ID.
- `getUpdatedDevices(long transactionId)`
  Gets complete device details for devices from the DCR database that were updated after the given transaction ID.
- `getDeletedDevices(DeviceIdentificationCollection managedDeviceIDs)`
  Gets the list of deleted devices from the given DeviceId list.
Table 6-5 lists the Event Date for DCR restore events.

<table>
<thead>
<tr>
<th>Restore Event</th>
<th>Event Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data restored from different domain</td>
<td>eventType= DC R_DATA_RESTORED_FROM_DIFFERENT_DCR_DOMAIN</td>
</tr>
<tr>
<td></td>
<td>dcrDomainId=Group123456</td>
</tr>
<tr>
<td></td>
<td>sourceApplication= DCR Server</td>
</tr>
<tr>
<td></td>
<td>sourceApplicationVersion=1.0</td>
</tr>
<tr>
<td></td>
<td>sourceMachineName=server1</td>
</tr>
</tbody>
</table>

| Data restored from same domain | eventType= DC R_DATA_RESTORED                                               |
|                               | dcrDomainId=Group123456                                                   |
|                               | sourceApplication= DCR Server                                             |
|                               | sourceApplicationVersion=1.0                                              |
|                               | sourceMachineName=server1                                                 |