



CHAPTER 11

Using Replication

Revised: March 20, 2009, OL-17222-03

This chapter provides information about how to use the replication feature in Cisco Access Registrar (CAR) and includes the following sections:

- [Replication Overview, page 11-1](#)
- [How Replication Works, page 11-2](#)
- [Replication Configuration Settings, page 11-6](#)
- [Setting Up Replication, page 11-9](#)
- [Replication Example, page 11-11](#)
- [Full Resynchronization, page 11-15](#)
- [Frequently Asked Questions, page 11-17](#)
- [Replication Log Messages, page 11-18](#)



Note

When using replication, use the **aregcmd** command-line interface to make configuration changes to the CAR server. Replication is not supported when using the GUI.

Replication Overview

CAR replication feature can maintain identical configurations on multiple machines simultaneously. When replication is properly configured, changes an administrator makes on the primary or *master* machine are propagated by CAR to a secondary or *slave* machine.

Replication eliminates the need to have administrators with multiple CAR installations make the same configuration changes at each of their installations. Instead, only the master's configuration need be changed and the slave is automatically configured eliminating the need to make repetitive, error-prone configuration changes for each individual installation. In addition to enhancing server configuration management, using replication eliminates the need for a hot-standby machine.

Using a hot-standby machine is a common practice to provide more fault-tolerance where a fully-installed and configured system stands ready to takeover should the primary machine fail. However, a system setup for hot-standby is essentially an idle machine only used when the primary system fails. Hot-standby or secondary servers are expensive resources. Employing CAR's replication feature, both servers can perform RADIUS request processing simultaneously, eliminating wasted resources.

The replication feature focuses on configuration maintenance only, not session information or installation-specific information such as Administrator, Interface, Replication or Advanced machine-specific configuration changes. These configuration items are not replicated because they are specific to each installation and are not likely to be identical between master and slave. While changes to Session Managers, Resource Manager, and Remote Servers are replicated to the slave and stored in the slave's configuration database, they are not hot-configured on the slave (see Hot Configuration Detailed below for more information)

Changes should be made only on the master server. Making changes on a slave server will not be replicated and might result in an unstable configuration on the slave. Any changes made using replication will not be reflected in existing **aregcmd** sessions. **aregcmd** only loads its configuration at start up; it is not dynamically updated. For example, if **aregcmd** is running on the slave, and on the master **aregcmd** is used to add a client, the new client, while correctly replicated and hot-configured, will not be visible in the slave's **aregcmd** until **aregcmd** is exited and restarted.

When there is a configuration change, the master server propagates the change set to all member servers over the network. All member servers have to update their configuration after receiving the change set notifications from master server. Propagating the change set to a member server involves multiple packet transfer from the master server to the member because the master server has to convey all the configuration changes to the member. The number of packets to be transferred depends on the size of the change set.

After receiving a change set notification, the member server will go off-line before applying the change set received from master server. This state is indicated by the log message `Radius Server is Off-Line` in **name_radius_1_log** file. When the change set is successfully applied, the member server goes up automatically. This is indicated by the log message `Radius Server is On-Line` in **name_radius_1_log** file. When the member server goes off-line to apply the change set, no incoming packets are processed.

Due to the number of packets to be transferred in the change set and the amount of time the member server will be offline updating its database points, Cisco recommends that you use multiple **save** commands rather than a large configuration change with one **save** command. You can also minimize the number of changes that occur in a replication interval by modifying either the `RepTransactionArchiveLimit` or the `RepTransactionSyncInterval`, or both of these properties. For example, instead of using the default value of 100 for the `RepTransactionArchiveLimit`, you might change it to 20.

How Replication Works

This section describes the flow of a simple replication as it occurs under normal conditions.

Replication Data Flow

The following sections describe data flow on the master server and the slave server.

Master Server

The following describes the data flow for the master server:

-
- Step 1** The administrator makes a change to the master server's configuration using the **aregcmd** command line interface (CLI) and issues a **save** command.
 - Step 2** After the changes are successfully validated, the changes are stored in the CAR database.

-
- Step 3** **aregcmd** then notifies the CAR server executing on the master of the configuration change.
- Step 4** The CAR server then updates its version of the configuration stored in memory. (This is called *hot-config* because it happens while the server is running and processing requests.)
- Step 5** The CAR server first copies the changes pertaining to the **aregcmd save**, also known as a transaction to its replication archive, then transmits the transaction to the slave server for processing.
- Step 6** In **aregcmd**, the prompt returns indicating that the **save** has completed successfully, the transaction has been archived, and the transaction has been transmitted to the slaves.
-

Slave Server

- Step 1** When the slave server receives the transaction, its contents are verified.
- Step 2** After verification, the changes are applied to the slave server's database
- Step 3** The changes are then applied (hot-configured) in the slave server's in-memory configuration.
- Step 4** The transaction is written to the slave server's replication archive.
-

Security

Replication has two primary security concerns:

- Security of the transactions transmitted to the slave server
- Storage of transactions in the replication archive

Both of these concerns use shared secret (MD5) encryption via the shared secret specified in the replication configuration on both master and slave servers. Replication data transmitted between master and slave is encrypted at the source and decrypted at the destination the same way as standard RADIUS packets between CAR's clients and the CAR server. Transactions written to the replication archive are also encrypted in the same manner and decrypted when read from the replication archive.

Replication Archive

The replication archive serves two primary purposes:

1. To provide persistent, or saved, information regarding the last successful transaction
2. To persist transactions in case the slave server requires re synchronization (see Ensuring Data Integrity below for more information on re synchronization).

The replication archive is simply a directory located in `../CSCOar/data/archive`. Each transaction replicated by the master is written to this directory as a single file. The name of each transaction file is of the form `txn#####` where `#####` is the unique transaction number assigned by the master server. The replication archive size, that is the number of transaction files it might contain, is configured in the Replication configuration setting of `TransactionArchiveLimit`. When the `TransactionArchive` limit is exceeded, the oldest transaction file is deleted.

Ensuring Data Integrity

CAR's configuration replication feature ensures data integrity through transaction data verification, transaction ordering, automatic resynchronization and manual full-resynchronization. With the single exception of a manual full-resynchronization, each of the following techniques help to automatically ensure that master and slave servers contain identical configurations. A detailed description of each technique follows.

Transaction Data Verification

When the master prepares a transaction for replication to a slave, the master calculates a 2's complement Cyclic Redundancy Check (CRC) for each element (individual configuration change) in the transaction and for the entire transaction and includes these CRC values in the transmitted transaction. When the slave receives the transaction, the slave calculates a CRC for each transaction element and for the entire transaction and compares its own calculated values with those sent with the message. If a discrepancy occurs from these comparisons, the transaction element or the entire transaction is discarded and a re-transmission of that particular transaction element or the entire transaction is requested by the slave from the master. This process is called automatic resynchronization. (described in more detail below)

Transaction Order

When the master prepares a transaction for replication, it assigns the transaction a unique transaction number. This number is used to ensure the transactions are processed by the slave in exactly the same order as they were processed on the master. Transactions are order dependent. Since the functionality of CAR's configuration replication feature is to maintain identical configurations between master and slave, if transaction order were not retained, master and slave would not contain identical configurations. Consider where two transactions modify the same thing (a defined client's IP address for example). If the first transaction was a mistake and the second was the desired result, the client configuration on the master would contain the second setting; however, if the transactions were processed in the reverse order on the slave, the client configuration on the slave would contain the mistaken IP Address. This example illustrates the critical need for transaction ordering to ensure data integrity.

Automatic Resynchronization

Automatic Resynchronization is the most significant feature with respect to data integrity. This feature ensures the configurations on both the master and slave are identical. If they are not, this feature automatically corrects the problem.

When the master and slave start-up, they determine the transaction number of the last replication transaction from their respective replication archives. The master immediately begins periodic transmission of a TransactionSync message to the slave. This message informs the slave of the transaction number of the transaction that the master last replicated.

If the transaction number in the TransactionSync message does not match the transaction number of the last received transaction in the slave's archive, then the slave will request resynchronization from the master. The resynchronization request sent by the slave will include the slave's last received transaction number.

The master will respond by retransmitting each transaction since the last transaction number indicated by the slave in the resynchronization request. The master obtains these transactions from its replication archive.

Should the slave's last received transaction number be less than the lowest transaction number in the master's replication archive, then automatic resynchronization cannot occur as the master's replication archive does not contain enough history to synchronize the slave. In this case, the slave must be resynchronized with a full-resynchronization.

Full Resynchronization

Full Resynchronization means that the slave has missed more transactions than are stored in the master's replication archive and cannot be resynchronized automatically. There is no automatic full-resynchronization mechanism in CAR's configuration replication feature. To perform a full resynchronization, see the *Cisco Access Registrar User's Guide*.

Understanding Hot-Configuration

Hot-Configuration is the process of reflecting configuration changes made to CAR's internal configuration database in the in-memory configuration of the executing CAR server. Hot-Configuration is accomplished without interruption of RADIUS request processing. For example, if an administrator uses **aregcmd** to configure a new client and issues a **save** command, when the prompt returns, the newly configured client can send requests to CAR.

Hot-Configuration minimizes the down-time associated with having to restart an CAR server to put configuration changes into effect. With the Hot-Configuration feature, a restart is only necessary when a Session Manager, Resource Manager or Remote Server configuration is modified. These configuration elements might not be hot-configured because they maintain state (an active session, for example) and cannot be modified without losing the state information they maintain. Changes to these configuration elements require a restart of CAR to put them into effect.

Hot-Configuration is not associated with the replication feature. Hot-Configuration's only connection to the replication feature is that when a change is replicated to the slave, the slave is hot-configured to reflect the replicated change as if an administrator had used **aregcmd** to make the changes directly on the slave server.

Replication's Impact on Request Processing

The replication feature was designed to perform replication of transactions with minimal impact on RADIUS request processing. When a transaction is received by a slave, RADIUS requests are queued while the transaction is applied to the slave. After the transaction is complete, RADIUS request processing resumes.

The impact on RADIUS request processing is a direct result of the size of a transaction. The smaller the transaction the lesser the impact, and the larger the transaction, the greater the impact. In other words, when making changes to the master, frequent saves are better than making lots of changes and then saving. Each change is one transaction element and all changes involved in a **save** comprise a single transaction with one element per change. Since the replication feature only impacts RADIUS request processing when changes are made, the impact under normal operation (when changes are not being made) is virtually unmeasurable.

Replication Configuration Settings

This section describes each replication configuration setting. In **aregcmd**, replication settings are found in **//localhost/Radius/Replication**.

RepType

RepType indicates the type of replication. The choices available are SMDBR and NONE.

When RepType is set to NONE, replication is disabled. To enable replication, set RepType to SMDBR for Single Master DataBase Replication. RepType must be set to SMDBR on both the master and slave servers.

RepTransactionSyncInterval

Master

On the master server, RepTransactionSyncInterval is the duration between periodic transmission of the TransactionSync message expressed in milliseconds. The default is 60000 or 1 minute.

The purpose of RepTransactionSyncInterval is to indicate how frequently to check for an out-of-sync condition between the master and slave servers. When the slave received the TransactionSync message, it uses its contents to determine if it needs to resynchronize with the master.

The larger the setting for RepTransactionSyncInterval, the longer the period of time between out-of-sync detection. However, if RepTransactionSyncInterval is set too small, the slave can frequently request resynchronization when it is not really out of sync. If the duration is too small, the slave cannot completely receive a transaction before it receives the TransactionSync message. In this case, the servers will remain synchronized, but there will be unnecessary excess traffic that could affect performance.

**Note**

Cisco recommends that you use smaller values for the RepTransactionSyncInterval to limit the time a slave server is offline applying change sets during automatic resynchronization.

Slave

On the slave, RepTransactionSyncInterval is used to determine if the slave has lost contact with the master and to alert administrators of a possible loss of connectivity between the master and slave. If the elapsed time since the last received TransactionSync message exceeds the setting of RepTransactionSyncInterval, the slave writes a log message indicating that it might have lost contact with the master. This log message is repeated each TransactionSyncInterval until a TransactionSync message is received.

RepTransactionArchiveLimit

On both master and slave, the RepTransactionArchiveLimit setting determines how many transactions can be stored in the archive. The default setting is 100. When the limit is exceeded, the oldest transaction file is deleted. If a slave requires resynchronization and the last transaction it received is no longer in the archive, a full resynchronization will be necessary to bring the slave back in sync with the master.

**Note**

The value set for RepTransactionArchiveLimit should be the same on the master and the slave.

An appropriate value for RepTransactionArchiveLimit depends upon how much hard disk space an administrator can provide for resynchronization. If this value is large, say 10,000, then the last 10,000 transactions will be stored in the archive. This is like saying the last 10,000 saves from **aregcmd** will be stored in the archive. Large values are best. The size of each transaction depends upon how many configuration changes were included in the transaction, so hard disk space usage is difficult to estimate.

**Note**

Cisco recommends that you use smaller values for the RepTransactionArchiveLimit to limit the time a slave server is offline applying change sets during automatic resynchronization.

If the slave should go down or otherwise be taken off line, the value of RepTransactionArchiveLimit and the frequency of **aregcmd** saves will determine how long the slave can be off-line before a full-resynchronization will be required.

There are two reasons why a slave server should have an archive:

1. The slave must save the last received transaction for resynchronization purposes (at a minimum).
2. Should the master go down, the slave can then be configured as the master and provide resynchronization services to other slaves.

RepIPAddress

The RepIPAddress value is set to the IP Address of the machine containing the CAR installation.

RepPort

The RepPort is the port used to receive of replication messages. In most cases, the default value (1645) is sufficient. If another port is to be used, the interfaces must exist in the machine.

RepSecret

RepSecret is the replication secret shared between the master and slave. The value of this setting must be identical on both the master and the slave.

ReplIsMaster

The ReplIsMaster setting indicates whether the machine is a master or a slave. On the master, set ReplIsMaster to TRUE. On the slave set it to FALSE. Only the master can have this value set to TRUE and there can be only one master.

RepMasterIPAddress

RepMasterIPAddress specifies the IP Address of the master. On the master, set RepMasterIPAddress to the same value used in RepIPAddress above. On the slave, RepMasterIPAddress must be set to the IP Address of the master.

RepMasterPort

RepMasterPort is the port to use to send replication messages to the master. In most cases, the default value (1645) is sufficient; however, if another is to be used, the interfaces must exist in the machine.

Rep Members Subdirectory

The Rep **Members**\ subdirectory contains the list of slaves to which the master will replicate transactions.

Rep Members/Slave1

Each slave is added much like a client is added. Each slave must have a configuration in the Rep Members directory to be considered part of the *replication network* by the master. The master will not transmit any messages or replications to servers not in this list, and any communication received by a server not in this list will be ignored.

**Note**

Although it is possible to configure multiple slaves with the same master, we have only considered a single-master/single-slave configuration. This is the recommended configuration.

Name

This is the name of the slave. The name must be unique.

IPAddress

This is the IP Address of the slave.

Port

This is the port upon which the master will send replication messages to the slave.


Setting Up Replication

This section provides step-by-step instructions about how to configure replication on both the master and member servers. The following section, “[Replication Example](#)” section on page 11-11, shows an example of replication configuration.

If possible, open an **xterm** window on both the master and member. In each of these windows, change directory to **\$INSTALL/logs** and run **xtail** to watch the logs. This allows you to watch replication log messages as they occur. If you are using a system which had a previous installation of CAR, delete all files located in the **\$INSTALL/data/archive** directory if it is present on either the master or member systems.

Configuring the Master

Use the following steps to configure the master server for replication:

-
- Step 1** On the machine which is to be the master, using **aregcmd**, navigate to **//localhost/Radius/Replication**
 - Step 2** Set the **RepType** to **SMDBR**:
set RepType SMDBR
 - Step 3** Set the **RepIPAddress** to the IP address of the master:
set RepIPAddress 192.168.1.1
 - Step 4** Set the **RepSecret** to **MySecret**:
set RepSecret MySecret
 - Step 5** Set **RepIsMaster** to **TRUE**:
set RepIsMaster TRUE
 - Step 6** Set **RepMasterIPAddress** to the same value used in step 3:
set RepMasterIPAddress 192.168.1.1
 - Step 7** Change directory to **/Radius/Advanced** and set the **MaximumNumberOfRadiusPackets** property to 8192:
cd /Radius/Advanced
set MaximumNumberOfRadiusPackets 8192
 - Step 8** Change directory to **Rep Members**:
cd “rep members”
-  **Note** You must enclose Rep Members in quotes due to the space in the name.
-
- Step 9** Add **member1**:
add member1
 - Step 10** Change directory to **member1**:
cd member1
 - Step 11** Set the **IPAddress** to the IP Address of the machine to be the member:
set IPAddress 192.168.1.2



Note The RepPort and RepMasterPort properties on the Master must correspond to one of the ports configured in **/Radius/Advanced/Ports**, if one is configured. Otherwise, the default values for the RepPort and RepMasterPort properties are sufficient.

Step 12 Save the configuration:

```
save
```

Step 13 Reload the configuration:

```
reload
```

Configuring The Member

Use the following steps to configure the member server for replication:

Step 1 On the machine which is to be the member, using **argcmd**, navigate to **//localhost/Radius/Replication**.

Step 2 Set the RepType to SMDBR.

```
set RepType SMDBR
```

Step 3 Set the RepIPAddress to the IP address of the member.

```
set RepIPAddress 192.168.1.2
```

Step 4 Set the RepSecret to MySecret.

```
set RepSecret MySecret
```

Step 5 Set RepMasterIPAddress to IP Address of the master (the same value used in Step 3 on page 8-1).

```
set RepMasterIPAddress 192.168.1.1
```

Step 6 Change directory to **/Radius/Advanced** and set the **MaximumNumberOfRadiusPackets** property to 8192.

```
cd /Radius/Advanced
```

```
set MaximumNumberOfRadiusPackets 8192
```

Step 7 If the Master has been configured to use a port other than the well-known (and default) RADIUS ports, configure each Member to use the same port.



Note The RepPort and RepMasterPort properties on the Master must correspond to one of the ports configured in **/Radius/Advanced/Ports**, if one is configured. Otherwise, the default values for the RepPort and RepMasterPort properties are sufficient.

Step 8 Save the configuration:

```
save
```

Step 9 Reload the configuration:

```
reload
```

Verifying the Configuration

After both servers have successfully started, use **aregcmd** to make a small change to be replicated to the member server which you can easily verify. We recommend setting the description in **//localhost/Radius** to something like *Test1*. After you issue an **aregcmd save** and the prompt returns, run **aregcmd** on the member server and change directory to **//localhost/Radius**. Ensure that the description is set to *Test1*. If this was successful, then replication is properly configured and functional.

Replication Example

This section provides an example of replication and shows the actions that occur.

Adding a User

On the master server, use **aregcmd** to add a new user to the default user list. To add a new user, perform the following steps:

-
- Step 1** Change directory to **//localhost/Radius/UserLists/Default**.
 - Step 2** Enter the following:
 add testuser
 - Step 3** Change directory to testuser.
 cd testuser
 - Step 4** Set the password for testuser.
 set password testuser
 - Step 5** Confirm the password by entering *testuser* again.
 - Step 6** Enter save to save the configuration.
-

Master Server's Log

The log on the master shows the following:

```
*** ./name_radius_1_log ***
10/23/2008 23:17:07 name/radius/1 Info Server 0 Initiating Replication of Transaction
1 with 2 Elements.
10/23/2008 23:17:07 name/radius/1 Info Server 0 Replication Transaction #1 With 2
Elements Initiated
```

Member Server's Log

The log on the member shows the following:

```
*** ./name_radius_1_log ***
10/23/2008 23:15:18 name/radius/1 Info Server 0 Radius Server is On-Line
10/23/2008 23:17:12 name/radius/1 Info Server 0 Committing Replication of Transaction
1 with 2 Elements.
```

```
10/23/2008 23:17:16 name/radius/1 Info Server 0 Replication Transaction #1 With 2
Elements Committed.
```

Verifying Replication

You can use one of two methods to verify that the new user *testuser* was properly replicated to the member:

- Run **aregcmd** on the member and look at the default userlist to see if it is there.
- Run **radclient** on the member and enter **simple testuser testuser** to create a simple access request packet (p001).

Enter **p001 send** to send it. When it returns with p002, enter **p002** to see if it is an Access Accept packet or an Access Reject packet. If it is an Access Accept, the user was properly replicated to the member. Using **radclient** is the recommended method to validate that a user was properly replicated.

On the Master, use **aregcmd** to delete the user from the default user list and save the user list.

Master Server's Log

The log on the master shows the following:

```
*** ./name_radius_1_log ***
10/23/2008 23:20:48 name/radius/1 Info Server 0 Initiating Replication of Transaction
2 with 1 Elements.
10/23/2008 23:20:48 name/radius/1 Info Server 0 Replication Transaction #2 With 1
Elements Initiated
```

Member Server's Log

The log on the member shows the following:

```
*** ./name_radius_1_log ***
10/23/2008 23:20:53 name/radius/1 Info Server 0 Committing Replication of Transaction
2 with 1 Elements.
10/23/2008 23:20:57 name/radius/1 Info Server 0 Replication Transaction #2 With 1
Elements Committed.
```

Repeat the validation procedure above to ensure the user *testuser* is no longer present on the member.

Using aregcmd -pf Option

CAR's replication feature works well using **aregcmd** input files. An **aregcmd** input file contains a list of **aregcmd** commands. For example, if the initial configuration of CAR were constructed in an input file, the master and member could be configured for replication first, then the input file applied to the master will be automatically replicated to the member.

To illustrate replication using an **aregcmd** input file, do the following:

Step 1 Create a text file called **add5users** with the following commands:

```
add /Radius/UserLists/Default/testuser1
cd /Radius/UserLists/Default/testuser1
```

```
set password testuser1
add /Radius/UserLists/Default/testuser2
cd /Radius/UserLists/Default/testuser2
set password testuser2
add /Radius/UserLists/Default/testuser3
cd /Radius/UserLists/Default/testuser3
set password testuser3
add /Radius/UserLists/Default/testuser4
cd /Radius/UserLists/Default/testuser4
set password testuser4
add /Radius/UserLists/Default/testuser5
cd /Radius/UserLists/Default/testuser5
set password testuser5
save
```

Step 2 On the master server, run the following command:

```
aregcmd -pf add5users
```

Master Server's Log

The log on the master shows the following:

```
*** ./name_radius_1_log ***
10/23/2008 23:27:08 name/radius/1 Info Server 0 Initiating Replication of Transaction
3 with 10 Elements.
10/23/2008 23:27:08 name/radius/1 Info Server 0 Replication Transaction #3 With 10
Elements Initiated
```

Member Server's Log

The log on the member shows the following:

```
*** ./name_radius_1_log ***
10/23/2008 23:27:12 name/radius/1 Info Server 0 Committing Replication of Transaction
3 with 10 Elements.
10/23/2008 23:27:17 name/radius/1 Info Server 0 Replication Transaction #3 With 10
Elements Committed.
```

When the prompt returns, go to the member and use **aregcmd** to view the **/radius/defaults/userlist**. There should be five users there named *testuser1* through *testuser5*.

An Automatic Resynchronization Example

This example will illustrate resynchronization of the member. This will be accomplished by stopping the member, making changes on the master, then restarting the member forcing a resynchronization.

-
- Step 1** At the member, stop the CAR server:
- ```
/etc/init.d/arservagt stop
```
- At the master, run **aregcmd** and change directory to **/radius/userlist/default**.
- ```
cd /radius/userlist/default
```
- Step 2** Enter the following:
- ```
add foouser
```
- Step 3** Change directory to **foouser**.
- ```
cd foouser
```
- Step 4** Set the password for **foouser**.
- ```
set password foouser
```
- Step 5** Confirm the password by entering *foouser* again.
- Step 6** Save the configuration:
- ```
save
```
-

Master Server's Log

The log on the master shows the following:

```
*** ./name_radius_1_log ***
10/23/2008 23:31:02 name/radius/1 Info Server 0 Initiating Replication of Transaction
5 with 2 Elements.
10/23/2008 23:31:02 name/radius/1 Info Server 0 Replication Transaction #5 With2
Elements Initiated
```

On the member, run **/etc/init.d/arservagt start**. Notice the following log messages in the Master's log:

```
*** ./name_radius_1_log ***
10/23/2008 23:33:19 name/radius/1 Info Server 0 Resynchronizing member1.
```

Member Server's Log

The log on the member shows the following:

```
*** ./name_radius_1_log ***
11/07/2008 23:33:14 name/radius/1 Info Server 0 Radius Server is Off-Line
11/07/2008 23:33:14 name/radius/1 Info Server 0 Starting Replication Manager
11/07/2008 23:33:24 name/radius/1 Info Server 0 Master Selected As Partner (DEFAULT)
11/07/2008 23:33:24 name/radius/1 Info Server 0 Radius Server is Off-Line
11/07/2008 23:33:24 name/radius/1 Warning Server 0 Requesting resynchronization from
Master: Last Txn#3
11/07/2008 23:33:24 name/radius/1 Info Server 0 Resynchronization from Master in
progress.
11/07/2008 23:33:24 name/radius/1 Info Server 0 Committing Replication of Transaction
4 with 2 Elements.
11/07/2008 23:33:28 name/radius/1 Info Server 0 Replication Transaction #4 With 2
Elements Committed.
11/07/2008 23:33:28 name/radius/1 Info Server 0 Radius Server is On-Line
```

As the log above shows, when the member started up, it validated its last received transaction number (#3) with the master's last replicated transaction number (#4). They did not match because a replication was initiated by the master which was not received by the member (because the member was stopped). When the member detected this discrepancy, the member made a resynchronization request to the master. The master responded by transmitting the missed transaction (#4) to the member. After it received and processed the retransmitted transaction, the member determined that it was then synchronized with the master and placed itself in an on-line status.

Full Resynchronization

Full Resynchronization means that the member has missed more transactions than are stored in the master's replication archive and can not be resynchronized automatically. There is no automatic full-resynchronization mechanism in CAR's configuration replication feature. If a full resynchronization is required, you must export the master server's database and update the member configuration.



Note

Before beginning, ensure there are no **aregcmd** sessions logged into the master server.

To perform a manual full-resynchronization, perform the following steps:

-
- Step 1** On the master server, stop the CAR server agent using the following command:
- ```
/etc/init.d/arserver stop
```
- Step 2** On the master server, change directory to **\$INSTALL/data/db**.
- Step 3** Create a tarfile made up of the three database files, **mcddb.d01**, **mcddb.d02**, and **mcddb.d03**.
- ```
tar cvf /tmp/db.tar mcddb.d0*
```
- Step 4** Create a tarfile of the archive.
- ```
tar cvf /tmp/archive.tar $INSTALL/data/archive
```
- Step 5** On the master server, start the CAR server agent using the following command:
- ```
/etc/init.d/arserver start
```
- Step 6** On each member server requiring resynchronization, perform the following:
- On the member server, stop the CAR server agent using the following command:


```
/etc/init.d/arserver stop
```
 - Copy the tarfiles (**db.tar** and **archive.tar**) to **/tmp**.
 - Change directory to **\$INSTALL/data/db**, then untar the compressed database files.


```
cd $INSTALL/data/db  
tar xvf /tmp/db.tar
```
 - Rebuild the key files using the following command:


```
$INSTALL/bin/keybuild mcddb
```



Note This step might take several minutes.

- e. Untar the archive.

```
cd $INSTALL/data/archive
```

```
tar xvf /tmp/archive.tar
```

- f. As a safety check, run the following UNIX command to verify the integrity of the database.

```
$INSTALL/bin/dbcheck mcddb
```



Note You must be user **root** to run **dbcheck**.

No errors should be detected.

- g. Start the CAR server agent using the following command:

```
/etc/init.d/arserver start
```



Note After you start the member server with the master server's database, you will probably see messages such as the following:

```
11/07/2008 23:21:23 name/radius/1 Error Server 0 TXN_SYNC: Failed to get master's
socket handle.
11/07/2008 23:21:49 name/radius/1 Warning Server 0 TXN_SYNC Received by Master from
unknown member 10.1.9.74. Validation Failed
```

These messages will likely continue until you complete steps **h** and **i**.

- h. Change directory to **//radius/replication** and change the following attributes:

- Change the RepIPAddress to that of the member.
- Change RepIsMaster to FALSE.
- Remove any entries under Rep Members.

- i. Save and reload the configuration.

```
save
```

```
Validating //localhost...
Saving //localhost...
```

```
reload
```

The member will start up and show on-line status in the log after it has verified it is synchronized with the master.

Frequently Asked Questions

Question: When I do a **save** in **aregcmd** and the validation fails, is anything replicated?

Answer: No; replication does not occur until **aregcmd** successfully saves the changes.

Question: Can I specify multiple masters with the same members?

Answer: No; the replication feature was designed to be used with a single-master. Also, it is not possible to specify more than one master in a member's configuration.

Question: Do I have to configure the master as a client on the member servers?

Answer: No. In-fact, it would be erroneous to do so. With the exception of Administrators, Interfaces, Replication, and Advanced machine-specific settings, the configuration between master and member must be identical. The replication feature's purpose is to maintain that relationship. Altering configuration settings on the member which are managed by the master will likely result in an unstable and possibly non-operational server.

Question: What configuration elements are replicated and what are not?

Answer: With the exception of Administrators, Interfaces, Replication, and Advanced machine-specific settings, all other settings are replicated.

Question: What configuration elements are hot-configured and what are not?

Answer: Session Managers, Resource Managers and Remote servers are not hot-configured because they maintain state, such as an active session, and cannot be manipulated dynamically.

Question: What is an appropriate TransactionSyncInterval setting?

Answer: This depends upon how long you want to allow an out-of-sync condition to persist. The shorter the interval, the more often an out-of-sync condition is checked. However, this results in added network traffic, additional processing by CAR and, if the interval is too small, frequent unnecessary resynchronization requests. The default value of 60,000 milliseconds (1 minute) is usually sufficient; however, values of as little as 10,000 milliseconds (10 seconds) have been tested and have worked well.

Question: What is an appropriate TransactionArchiveLimit setting?

Answer: This depends upon two things:

1. How much hard disk space you are willing to devote to transaction archive storage
2. How often your configuration is changed (a save is issued through Aregcmd).

If you have limited hard disk space, then perhaps smaller values (less than 1000) are appropriate; however if you have sufficient hard disk space, values of 10,000 or greater are better. The primary reason for this preference is to limit the possibility of a full-resynchronization being required. A full-resynchronization is required when the member has missed so many transactions that the master no longer contains all the transaction necessary to resynchronize the member. The greater the limit, the longer the member can be down without requiring a full-resynchronization.

Question: Can I specify a member in the member configuration?

Answer: Yes, and this is recommended. In the member's replication configuration Rep Members list, specify another server, perhaps one which can be used in-case of critical failure of the master. If the master suffers a catastrophic failure (a hard disk crash, for example) the member can be reconfigured to be the master simply by setting the RepIsMaster to TRUE and changing the MasterIPAddress to its own IP Address and the member specified in its Rep Members list will perform as the member. Because the member has an archive of transactions, the new member can be automatically resynchronized. If the archive limit on the new master has been exceeded (the transaction file txn0000000001 is no longer

present in the new master's archive directory), then the new member will require a full-resynchronization. Setting the member up in this manner prevents down-time if the master fails and allows configuration changes to be made on the new master.

Question: How can I prevent a full-resynchronization from ever being necessary?

Answer: You can't, but you can limit the possibility by setting the TransactionArchiveLimit to a large value (greater than 10000). Another technique is to periodically check the archive when the master and member are synchronized. If the number of transaction files is approaching 10,000, then you can stop the master and member servers, delete all files in the replication archive, and restart the master and member. The only side effect is that if the master or member suffers a catastrophic failure, a full resynchronization will be required.

Question: Can I use the member to process RADIUS requests along with the master?

Answer: Yes, and this was one of the goals of the replication feature. Keep in mind that session information is not replicated between master and member. To use session management in this environment, use CAR's central session manager.

Replication Log Messages

This section contains typical replication log messages and explains what each means. This section include the following subsections:

- Information Log Messages
- Warning Log Messages
- Error Log Messages
- Log Messages You Should Never Receive

Information Log Messages

Error Message Starting Replication Manager

Displayed at start-up and indicates the Replication Manager is configured and enabled. (RepType=SMDBR)

Error Message Replication Disabled

Displayed at start-up and indicates that Replication is not enabled. (RepType=NONE)

Error Message Radius Server is On-Line

Displayed by the member at start-up to indicate the member is synchronized with the master and processing RADIUS requests. It is also displayed after a successfully completed resynchronization. This message is never displayed on the master.

Error Message Radius Server is Off-Line

Displayed by the member at start-up to indicate the radius server is not processing RADIUS requests until it can ensure synchronization with the master. When this is displayed after startup, it indicates the member is no longer synchronized with the master and is directly associated with a resynchronization request to the master. This message is never displayed on the master.

Error Message Resynchronizing <member name>

Displayed by the master to indicate that it is resynchronizing the specified member (member).

Error Message Resynchronization from Master in progress.

Displayed by the member to indicate the master is in the process of resynchronizing it.

Error Message Resynchronization complete.

Displayed by the member to indicate the resynchronization has completed successfully.

Error Message Resynchronization did not complete before timeout. Retrying.

Indicates the master did not complete the resynchronization before the member expected it to complete and that the member is re-requesting resynchronization from the master for the remaining missed transactions.

Error Message Master Selected As Partner (DEFAULT)

Displayed by the member to indicate it has successfully connected with the master.

Error Message Initiating Replication of Transaction <transaction #> with <# of elements> Elements.

Displayed by the master to indicate that it is beginning replication of a transaction to the member.

Error Message Replication Transaction #<transaction #> With <# of elements> Elements Initiated

Displayed by the master to indicate that it has completed sending the transaction to the member.

Error Message Committing Replication of Transaction <transaction #> with <# of elements> Elements.

Displayed by the member to indicate that it has received a transaction and is processing it.

Error Message Replication Transaction #<transaction#> With <# of element> Elements Committed

Displayed by the member to indicate that the transaction has been successfully processed.

Error Message Stopping Replication Manager

Displayed at shutdown by both the master and member to indicate the replication manager is being shut down.

Error Message Stopping Replication Manager - waiting for replication to complete...

Displayed by the member when a shutdown is attempted while received replications are being processed. After the replications are complete, the shutdown will complete.

Error Message Replication in progress. Please wait...

Periodically displayed while a shutdown is pending and replications are being completed.

Error Message Replication Manager Stopped

Displayed by both the master and member to indicate the replication manager has been successfully shutdown.

Warning Log Messages

Error Message Transaction Sync not received within configured TransactionSyncInterval. Communication with the Master may not be possible.

The member displays this log messages to indicate that it has not received a TransactionSync message from the master within its configured TransactionSync interval.

Error Message TXN_SYNC Received by Master from unknown member <ip address>. Validation Failed

Displayed by the master when a TransactionSync message is received by the master. Since there can be only one configured master in a replication network, and the master is the only server who can send a TransactionSync message, this indicates there is another configured master in the replication network.

Error Message TXN_SYNC Received from unknown Master <ip address>. Validation Failed

Displayed by the member to indicate that a TransactionSync message was received from a server not configured as its master.

Error Message Requesting resynchronization from Master: Last Txn#<transaction#>

Displayed by the member to indicate that it is requesting resynchronization from the master. The LastTxn# is the last transaction number the member received and processed successfully.

Error Message Resynchronization Request received from unknown member.

Displayed by the master when a resynchronization request is received by a member who is not listed in its `/radius/replication/rep` members configuration.

Error Message Resynchronization of <member name> requires Full Resynchronization.

Displayed by the master to indicate that the member cannot be automatically resynchronized because its last transaction number is not within the configured history length of the archive (TransactionArchiveLimit). A manual resynchronization of the member is required to put the member back in-sync.

Error Message MEMBER_SYNC Received from unknown Master at <ip address>. Validation Failed

Displayed by a member indicating that a master, other than its configured master, is requesting partnership.

Error Message MEMBER_SYNC Received by Master from unknown member <ip address>. Validation Failed

Displayed by the master to indicate a member not listed in its `/radius/replication/rep` members configuration has requested partnership.

Error Message TXN_EXPECT Received by Master from unknown <ip address>.

Displayed by the master to indicate it has received a transaction which originated from another illegal master.

Error Message TXN_EXPECT Received from unknown Master <ip address>.

Displayed by the member to indicate it has received a transaction which originated from a master other than its configured master.

Error Message TXN_EXPECT Broadcast failed.

Indicates that the master could not initiate a replication.

Error Message DATA_SYNC Received by Master from unknown <ip address>

Displayed by the master to indicate that it received a replication transaction from another illegal master.

Error Message DATA_SYNC Received from unknown <ip address>

Displayed by the member to indicate that a transaction was received from a server external to the replication network.

Error Log Messages

Error Message DATA_SYNC Validation failed - CRC Mismatch

Displayed by the member to indicate a received transaction element is invalid.

Error Message TXN_SYNC: Failed To Get Member Socket Handle.
 TXN_SYNC: Failed to get master's socket handle.
 MEMBER_SYNC could not get socket handle
 TXN_EXPECT: Failed to get socket handle.
 DATA_SYNC could not get socket handle.
 These messages indicate an invalid interface configuration in Cisco Access Registrar.
 They could also be the result of specifying an invalid RepPort setting.
 Failed To Create TXN_SYNC packet. (out of packets?)
 Failed To Create TXN_SYNC packet.
 MEMBER_SYNC Failed to create packet.(out of packets?)
 MEMBER_SYNC Failed to create packet.
 TXN_EXPECT Failed to create packet.(out of packets?)
 TXN_EXPECT Failed to create packet.
 DATA_SYNC Create packet failed.(out of packets?)
 DATA_SYNC Create packet failed.

These message indicate that a packet could not be created. This could be the result of a low memory condition or the result of the /Radius/Advanced/ MaximumNumberOfRadiusPackets setting being set too low

Error Message TXN_SYNC validation failed - Internal error (pTxnSync=NULL).
 MEMBER_SYNC validate failed - Internal Error (pMemberSync=NULL)
 DATA_SYNC Validation Failed - Internal (pDataSync = NULL).
 TXN_EXPECT Could not add new datablock to pending transaction queue.
 Replication Member could not be added to member list.
 Replication Member could not be added to member list.

These messages are the result of a failed memory allocation possibly due to an out of memory condition.

Error Message DATA_SYNC Packet creation failed - Invalid ordinal.
 Attempt To Replicate Transaction With Zero Elements.
 Internal Error - Selected member not valid
 Internal Replication Error ChangeType <change type> For <element path>
 Internal error - Replication manager is invalid

These messages indicate an internal application failure.

Error Message Cannot archive transaction datablock
 Could not archive transaction

These messages are the result of a failed archive attempt. This could be the result of a low disk space condition.

Error Message Could not commit transaction to MCD
 Cannot Get Value For Unsupported DataType <data type id>
 MCD Replication Cannot Delete Value <element path>
 MCD Replication Cannot Delete Directory <element path>
 MCD Replication Cannot Delete Value For <element path> With Unsupported DataType
 <data type id>
 MCD Replication Cannot Create Dir For <element path>
 MCD Replication Cannot Set Value For <element path>
 MCD Replication Cannot Set Value For <element path>

```

MCD Replication Cannot Set Value For <element path>
MCD Replication Cannot Set Value For <element path>
MCD Replication Cannot Set Value For <element path> With Unsupported DataType
<data type id>
MCD Replication Cannot Set Value For <element path> With UNKNOWN DataType <data
type id>

```

These messages are the result of a failed replication commit attempt.

Log Messages You Should Never See

The following list contains log messages which you should never see displayed in a log. If any of these messages are displayed in the log, contact CAR technical support for assistance.

```

Error Message <member name> Selected As Partner (DEFAULT)
DATA_SYNC Received from non-partner <ip address>
DATA_RE_SYNC CRC mismatch. Replying with NAK
DATA_RE_SYNC Commit Failed. Replying with NAK
EVAL_SYNC Validation failed. <ip address> is not a Master or Member of the
Replication network
EVAL_SYNC Received from unknown member.
PARTNER_SYNC Received from unknown member <ip address>.
PARTNER_SYNC Received from unknown member <ip address>.
EVAL_SYNC Cannot get socket handle.
EVAL_SYNC Failed to create packet.(out of packets?)
EVAL_SYNC Failed to create packet.
EVAL_SYNC Validation failed - Internal Error (pEvalSync=NULL).
PARTNER_SYNC Failed to get socket handle.
PARTNER_SYNC Failed to create packet. (out of packets?)
PARTNER_SYNC Failed to create packet.
DATA_RE_SYNC Can't get socket handle
DATA_RE_SYNC Failed to create packet (out of packets?)
DATA_RE_SYNC Failed to create packet
DATA_RE_SYNC Failed validation - Internal Error (pReSync = NULL)
DATA_RE_SYNC Cannot Set Value For <element path>
DATA_RE_SYNC Cannot Set Value For <element path>
DATA_RE_SYNC Cannot Set Value For <element path>
DATA_RE_SYNC Cannot Set Value For <element path>
DATA_RE_SYNC Cannot Set Value For <element path> With Unsupported DataType <data
type id>
DATA_RE_SYNC Cannot Set Value For <element path> With UNKNOWN DataType <data type
id>;
DATA_RE_SYNC Received by Master from unknown member <ip address>
DATA_RE_SYNC Received from unknown Master <ip address>DATA_RE_SYNC Reply received
by Master from unknown Member <ip address>
Could not replicate data element to partners.
Could not replicate to partners - Invalid Ordinal.

```

