CHAPTER 12

Using Backup and Restore

This release of CWCS provides a database backup and restore framework based on the classical CMF backup and restore infrastructure. This framework is used by all CWCS-based applications, including RME, Campus, DFM, ~~DFMfh~~ and ACLM.

As of release 3.0, this framework can also restore the backup data from CMF 2.1 and CMF 2.2. Restoring a backup from a CMF release earlier than 2.1 is not supported.

CORE-based backup and restore is no longer supported.

The following topics describe using the CWCS backup and restore framework:

- Using CWCS Backup
- Using CWCS Restore
- CWCS Backup and Restore API Command Reference
- Restoring a Corrupt Database

For more information about CWCS backup and restore features, refer to *CMF2.3 Restore Framework Software Design Specification*, ENG-306868.

Using CWCS Backup

The following topics describe using the CWCS backup framework:

- CWCS Backup
- How CWCS Backups are Stored
- Running CWCS Backups
- Offline Backup

CWCS Backup

There is no change in the backup program for CWCS 3.0. However Campus Manager, ACLM, and DFM applications must change their application backup manifest directory structure as shown in Table 12-1.

Table 12-1 Application Manifest Directory Changes for CWCS 3.0

Application	Backup Directory
Campus Manager	$NMSROOT/backup/manifest/campus **Note** Campus has its own restore adaptor: $NMSROOT/bin/RestoreTools/campus/restore.pm. **Note** If the backup is based on CMF 2.2, access the backup files under TEMP_FOLDER/cmf/campus, TEMP_FOLDER/cmf/ut and TEMP_FOLDER/cmf/ani. **Note** If the backup is based on CWCS 3.0, access the backup files under TEMP_FOLDER/campus.
ACLM	$NMSROOT/backup/manifest/aclm **Note** ACLM has its own restore adaptor: $NMSROOT/bin/RestoreTools/aclm/restore.pm. **Note** If the backup is based on CMF2.2, access the backup files under TEMP_FOLDER/rme/aclm. **Note** If the backup is based on CWCS 3.0, access the backup files under TEMP_FOLDER/aclm.
DFM	$NMSROOT/backup/manifest/dfm **Note** DFM has its own restore adaptor: $NMSROOT/bin/RestoreTools/dfm/restore.pm **Note** Irrespective of the version of CMF or CWCS, access the DFM data from TEMP_FOLDER/dfm. In CWCS 3.0, DFMfh is a module under DFM. **Note** If the backup is based on CMF 2.1 or CMF 2.2, access the DFMfh data under TEMP_FOLDER/dfmfh. **Note** If the backup is based on CWCS 3.0, access the DFMfh data under TEMP_FOLDER/dfm/dfmfh.

How CWCS Backups are Stored

The CWCS backup and restore framework creates backup locations of the form
backupdir/generationnumber/suite/directory/filename

Where:

- *backupdir* is the root directory where all backups are to be stored.
- *generationnumber* is the number of the backup. Directories are created in serial order, with the highest number representing the latest backup. For example: 1, 2, and 3, where 3 is the latest database backup.
- *suite* is the name of your application or suite. Often, this is the same as the data source (database) name. For example: For CWCS, the *suite* and database name are both “cmf”, but for Campus Manager, the *suite* is “campus” and the database name is “ani”.

- *directory* is the the database directory being backed up. Directories include the database directory and any suite application's data directories. It can also include the database template file, the CWCS (or CMF) version information, and filebackup.tar (which is the archive of all application configuration data files listed in the application backup manifest file datafiles.txt).
- *filename* is the name of the files that have been backed up. These files include database (.db), log (.log), the *dsn*.txt file, and database version file (*dsn*_DbVersion.txt). Application directories will contain only the copy of datafiles.txt from the backup manifest-specified locations.

Running CWCS Backups

To run a CWCS backup, each application must:

Step 1 Create the backup manifest directory structure *$NMSROOT*/backup/manifest/*suite* (see the "Creating the Backup Manifest Files" section on page 11-10).

Step 2 Under that structure, create the /database/orig subdirectory and place the *dsn*.txt file there. The *dsn*.txt file contains the information required to back up the suite database (see the "Creating the Database Backup Manifest File" section on page 11-11).

For example: For the CWCS suite, the database backup manifest file cmf.txt goes in *$NMSROOT*/backup/manifest/cmf/database/orig.

Step 3 Also under the*$NMSROOT*/backup/manifest/*suite*, create the */app*/orig subdirectory for each application module to be backed up and place its matching datafiles.txt file there. Each datafiles.txt file contains the list of files to be backed up for that application module (see the "Creating the Application Backup Manifest File" section on page 11-11)

For example, the configArchive module of Resource Manager Essentials must place its datafiles.txt file under *$NMSROOT*/backup/manifest/rme/configArchive/orig.

Step 4 Configure the database backup manifest by running the following command:

```
$ENV{"NMSROOT"}/bin/perl $ENV{"NMSROOT"}/objects/db/conf/configureDb.pl action=install
dsn=dsn
```

Where *dsn* is your database name (e.g., for CWCS, `cmf`).

The script will copy the suite's *dsn*.txt file from the /orig directory to its parent /database directory, and replace `$ENV{"NMSROOT"}` with the actual directory path. For example: For CWCS, the cmf.txt file will be copied from *$NMSROOT*/backup/manifest/cmf/database/orig to $NMSROOT/backup/manifest/cmf/database.

Step 5 Configure the application backup manifest by running the following command:

```
$ENV{"NMSROOT"}/bin/perl $ENV{"NMSROOT"}/objects/db/conf/configureDb.pl action=install
app=app
```

Where *app* is your application module (e.g., for RME, `configArchive`).

The script will copy the application module's datafiles.txt file from its /orig directory to its parent */app* directory, and replace `$ENV{"NMSROOT"}` with the actual directory path. For example: For RME, the datafiles.txt file will be copied from *$NMSROOT*/backup/manifest/rme/configArchive/orig to *$NMSROOT*/backup/manifest/rme/configArchive

Step 6 The database and applications are now registered for CWCS backup and restore. To back them up, run the backup utility backup.pl (see the "backup.pl" section on page 12-10). The utility will find out what to back up from the database and application backup manifest files.

Related Topics

See the "configureDb.pl" section on page 11-53.

Offline Backup

Applications may require offline backup to avoid inconsistencies between the flat files and databases. If any of the application requires offline backup, then you can configure the backup mode as offline by specifying the value of BACKUP_OFFLINE parameter as YES in the backup.properties file under $NMSROOT/conf/. It should be done in respective applications install flow and should not be exposed to end-users.

If the offline backup is configured, then

- CiscoWorks Home page urgent message will be sent to all the users who are currently logged in before stopping daemon manager
- A JMS Event with the subject **cisco.mgmt.cw.cmf.backup** will be published before stopping daemon manager. Applications can listen for this event and can act accordingly.
- Daemons will be stopped during the backup process and will be started again after the backup is completed.

Using CWCS Restore

The following topics describe using the CWCS restore framework:

CWCS Restore: Changes for CWCS 3.0

For the CWCS 3.0 release:

- In previous releases, CMF applications placed their database files in the <Application>/database folder in the backup archive. Now, the restore framework:
  - Copies the files in the backup archive to the TEMP_FOLDER/tempBackupData/<Application> folder.
  - Extracts the filebackup.tar of each application to TEMP_FOLDER/tempBackupData/<Application>/CSCOpx (irrespective of the original $NMSROOT where it was backed up). This temporary directory structure is the same for Solaris and Windows.
- The Remote Upgrade tools are no longer available because restoring data from previous versions is incorporated into the restore programs. You can use the CWCS backup and restore programs to restore to CWCS 3.0 your data from older CMF 2.1 or 2.2.
- The Per-Product Restore feature is no longer available due to data inconsistencies caused by the replacement of the database during restore.

Understanding the CWCS Restore Framework

In previous versions of the CWCS backup and restore framework, applications registered their files and folders to be backed up and restored in the file, datafiles.txt. The new framework extracts the backup data to a temporary directory, then calls the application's restore adaptors. Using the APIs and facilities provided by the framework, the adaptors handle the data conversion and store it in the runtime location.

Note New restore framework applications are required to write their own restore adaptors.

The new restore framework:

1. Extracts the backup data to a temporary location. For applications installed in the system that have data in the backup archive, it:
   a. Copies the backup data of these applications to TEMP_FOLDER/tempBackupData/<AppName>
   b. Extracts the filebackup.tar of the applications to TEMP_FOLDER/tempBackupData/<AppName>/CSCOpx folder. On Solaris platforms, the non-$NMSROOT files are extracted to the TEMP_FOLDER/tempBackupData/<AppName> folder.

   Note The default location of the temporary directory is $NMSROOT/tempBackupData. You can use the -t option of the restore program to specify a different temporary directory.

2. For each of the following steps, the application restore adaptors are called in this pre-defined order: CMF/CWCS, Campus, RME, ACLM, and DFM.
   a. Run preRestore() functions for all application adapters.
   b. Run doRestore() functions for all application adapters.
   c. Run postRestore() functions for all application adapters.

   Note The application's restore adaptors are loaded from $NMSROOT/bin/RestoreTools/<AppName>/restore.pm.

3. Removes the temporary folder TEMP_FOLDER/tempBackupData.

The applications have knowledge of their data and the conversion logic. Therefore, the application's adaptor (restore.pm) will:

1. Extract the backup data from these locations:
   – TEMP_FOLDER/tempBackupData/<AppName/CSCOpx
   – TEMP_FOLDER/tempBackupData<AppName>

   Note Applications access their archive data from this temporary directory. To find the location of this directory, application adaptors should use the **getTempFolder**() API.

2. Do the conversion if needed.

3. Apply the converted file to the running machine.

Running CWCS Restores

To run a CWCS restore operation, each application must:

Step 1 Write a Perl adaptor that takes care of applying application data to the runtime structure (see the "Guidelines for Writing CWCS Restore Application Adaptors" section on page 12-6).

Step 2 Register the application with CWCS restore by placing the restore adaptor (the Perl module) in

```
$NMSROOT/bin/RestoreTools/<Application Name>
```

Step 3 Run the restore utility (see the "restorebackup.pl" section on page 12-11).

Guidelines for Writing CWCS Restore Application Adaptors

For CWCS 3.0, the new restore framework requires that all applications using the framework write their own application adaptors. When you write an application adaptor, follow these guidelines:

- All application adaptors must beimplemented as Perl modules. To implement a restore program in Java or any other language, supply a Perl module to call the non-Perl application.
- All application adaptors must be named restore.pm.
- Use the APIs provided by the CWCS backup and restore framework (see the "CWCS Backup and Restore API Command Reference" section on page 12-9).
- Place the restore.pm files in subdirectories of the RestoreTools folder in the bin directory. They will be identified based on the directory in which they are placed.

  ```
  $NMSROOT/bin/RestoreTools/<Application>/restore.pm.
  ```

  For example, RME's application adaptor is stored in:

  ```
  $NMSROOT/bin/RestoreTools/rme/restore.pm
  ```

- All application restore modules must contain four functions. These functions are called by the restore framework.
  - preRestore(). The preRestore() function should not make any changes in the running machine that breaks the applications. Follow this principle: if even one application adapter's preRestore() fails, restore the system to the state it was in before the restore program was run.

    Note This function can be empty but must be declared.

  - doRestore(). Applications should use only the doRestore() function to apply data to the machine.
  - postRestore(). The postRestore() function cannot terminate the restore. This function is used to do fine tuning after the restore.

    Note This function can be empty but must be declared.

  - UNLOAD_restore(). The UNLOAD_restore() function removes the functions defined in this restore.pm from memory.

Note **If applications are using any other functions in their restore.pm, they should add an entry for those functions to this list.**

- The last line of the restore.pm must be "1;" to indicate the end of the module.
- When writing the application adapter, use the format shown in Example 12-1.

Example 12-1 CWCS Restore Application Adapter Format

```
sub preRestore
{
    # Code for pre restore of this application;
    return 0; # But on error should return non-zero.
              # Returning non-zero will stop the restore.
}
sub doRestore
{
   # Code for the actual restore.
    return 0; # But on error should return non-zero.
              # Returning non-zero will stop the restore.
}
sub postRestore
{
   # Code for post restore operation
}
sub UNLOAD_restore
{
   # This function removes the functions defined in this restore.pm from memory.
   # The following three entries are required. If applications are using any
   # other functions in their restore.pm, they should add an entry for
   # those functions here.
    undef  &preRestore;
    undef  &doRestore;
    undef  &postRestore;
}
1;
```

- Before calling the application's module, the restore framework redirects the STDOUT and STDERR to the restorebackup.log file. Therefore, applications can print their messages to STDOUT using print statements, which will be captured in restorebackup.log. This is the default behavior.

  However, this means that none of an application's messages will be displayed in the CLI. To add user input to a restore adapter:

  - Before calling user input functions, call the redirectToScreen () API (see the "redirectToScreen" section on page 12-16). This redirects further outputs of STDOUT to print on the screen.
  - After calling the user input functions, call the redirectToLog () API (see the "redirectToLog" section on page 12-17). This will redirect further outputs of STDOUT back to the restore log file.
- When reporting errors, follow these guidelines:

- The modules should not use Perl's exit() function to exit the program. They should return to their respective methods with the appropriate return values.
- The preRestore() functions should return zero for success and non-zero for any error.
- The doRestore() functions should return zero for success and non-zero for any error.
- The return status of the postRestore() function is ignored by the framework.

Related Topics

See the:

- "Sample CWCS Restore Application Adaptor" section on page 12-8.
- "CWCS Backup and Restore API Command Reference" section on page 12-9.

Sample CWCS Restore Application Adaptor

Example 12-2 assumes that the backup archive for CMF2.1, CMF2.2, and CWCS 3.0 contains the following files for this application:

- CMF2.1
  - d:\program Files\CSCOpx\conf\file1.conf
  - d:\Program Files\CSCOpx\etc\file2.data
- CMF2.2
  - d:\CW2000\conf\file1.conf
  - d:\CW2000\etc\file2.data
  - d:\CW2000\etc\file3.data
- CWCS 3.0
  - d:\CW2000\conf\file1.conf
  - d:\CW2000\etc\file2.data
  - d:\CW2000\etc\file3.data

In this example:

- file1.conf is not changed between the three versions.
- file2.data of CMF2.1t needed conversion when restored in CWCS 3.0.
- file3.data was introduced in CMF2.2 and does not need any conversion in CWCS 3.0.

Example 12-2 Sample restore.pm File

```
sub preRestore  {return 0;}
sub doRestore
{
  if (getCMFVersion() eq "2.1")
  {
      if (CopyFileToNMSROOT("conf\file1.conf")!=0) {return -1;}
      if (convertData2()!=0) {return -1;}
  }
  elsif (getCMFVersion() eq "2.2")
  {
```

```
    if (CopyFileToNMSROOT("conf\file1.conf")!=0) {return -1;}
    if (CopyFileToNMSROOT("etc\file2.data")!=0) {return -1;}
    if (CopyFileToNMSROOT("etc\file3.data")!=0) {return -1;}
  }
  elsif (getCMFVersion() eq "3.0")
   {
    if (CopyFileToNMSROOT("conf\file1.conf")!=0) {return -1;}
    if (CopyFileToNMSROOT("etc\file2.data")1=0) {return -1}
    if (CopyFileToNMSROOT("etc\file3.data")1=0) {return -1;}
   }
   return 0;
}
sub convertData2
{
  my $NMSROOT, $d, $APP, $TEMP_FOLDER;
  $NSMROOT= getNMSROOT(); $d= getFolderSeperator();
  $TEMP_FOLDER = getTempFolder(); $APP="cmf";
     SourceFile=TEMP_FOLDER+$d+$APP+$d+"CSCOpx"+$d+"etc"+d$+"file2.dat;
     DestinationFile=$NMSROOT+$d+"CSCOpx"+$d+"etc"+d$+"file2.dat;
     SourceHandler = OpenFile(SourceFile,<Read Mode>);
     DestinationHandler = OpenFile(DestinationFile, <Write Mode>);
# Read content of SourceHandler, convert, write to DestinationHandler
close SourceFileHandler, DestinationFileHandler
Return 0 for success, -1 for any errors.
}

sub postRestore { }
sub UNLOAD_restore
{
   undef &preRestore;  undef &doRestore; undef &convertData2;
   undef &postRestore;
}
1;
```

CWCS Backup and Restore API Command Reference

The following topics describe the utilities and APIs provided by the CWCS framework.

Use these utilities to backup and restore your databases:

- backup.pl
- restorebackup.pl

Use these APIs when you write CWCS restore adaptors for your applications:

- copyFileToNMSROOT
- copyFolderToNMSROOT
- getCMFVersion
- getCMFPatchVersion
- getNMSROOT
- getArchiveNMSROOT
- getFolderSeperator
- getLogFileName
- getTempFolder

backup.pl

```
$NMSROOT/bin/perl $NMSROOT/bin/backup.pl backdir logfilename numberGen
```

Backs up the database file to a specified directory. It also backs up the files that are listed in the backup manifest files into specific directory locations.

The backup script requires the following information:

- The location of the databases

  The database backup manifest file contains a list of database file names. The backup.pl script backs up all specified database files.

- A list of directories containing data files

  The application backup manifest file contains a list of directory names. The backup.pl script backs up all files in the specified directories.

Input Arguments

`backdir`	Backup (target) directory. Must be writable. The full directory path is required. The directory will be created if it does not exist.
`logfilename`	The name of the backup log file. When run from the command line: If specified, the full path is required and the parent directory must exists and be write-enabled (if not, it will *not* create the directory and will throw the message `Error: Cannot save STDERR to the log file`). If not specified, the log will be written to STDOUT. From the GUI: The log is written to $NMSROOT/log/dbbackup.log; there is no option to specify a log file.
`numberGen`	Number of backup archives to retain in the directory. Must be an integer value or blank. If blank, it creates or overwrites archive 0 and keeps all other archives. If specified, it creates a new archive and keeps only that number of archives. For example: You have 100 backups; archive 1 is the oldest and 100 is the newest. If *`numberGen`* is 5, backup.pl creates a new archive 101, deletes archives 1-96, and keeps archives 97-101. On the next run with the same *`numberGen`* value, it creates new archive 102 and deletes archive 97.

Remarks

- You must specify the full path to both perl and the backup.pl script. You must also use the version of Perl supplied with CWCS.
- You can run this script from the CWCS desktop by selecting **Server Configuration > Admin > Backup**. This option lets you run a backup immediately or schedule it for a later date and time.

- If a backup fails due to a previous backup being aborted or interrupted, you will see the following error message

  ```
  "…/backup.LOCK file exists. Most probably another backup process is running"
  ```

  If you are sure that no another backup process is running, you can delete the backup.LOCK file. The purpose of this file is to prevent more than one instance of the backup process.

Related Topics

See the "Creating the Backup Manifest Files" section on page 11-10.

restorebackup.pl

```
perl restorebackup.pl -d backupdir [-t tempdir] [-gen generation]
```

Restores a previous backup. This script is run on demand from the command line with root privileges. The script will verify that no applications are running when it is invoked.

Note Shut down the Daemon Manager before starting this utility.

Runtime Location

$NMSROOT/bin

Switches

`-d backupdir`	Path to the backup directory. Required.
`-gen generation`	Generation to be restored. By default, restores the latest generation. Optional.
`-t tempdir`	Specifies a different temporary directory. Optional. The restore framework uses a temporary directory to extract the contents of the backup archive. By default the temporary directory is NMROOT/tempBackupData.

For example, to restore the fourth backup of Campus Manager from the MyDir directory, enter:

```
perl restorebackup.pl -d /MyDir -gen 4
```

Remarks

- This script restores the database files to the location specified in the appropriate {*dsn*}.txt file in the manifest directory for the database. This ensures that the database is restored to the proper location if the database is moved after a backup.
- This script uses the tar command to untar filebackup.tar for the data files indicated in the datafiles.txt file for the application. The datafiles.txt file is located in the following directory path:

  ```
  $NMSROOT/backup/manifest/<Application Name>/<module Name>
  ```

- For Windows platforms, the untar command gets a “Permission denied” error if the original backup directory and its files do not have read permission to the current user. Therefore, application data files must be readable.

Related Topics

See the “Guidelines for Writing CWCS Restore Application Adaptors” section on page 12-6.

copyFileToNMSROOT

Copies the specified file from the backup archive to *$NMSROOT*.

Note Call this function only when there are no data conversions for the file.

Syntax

```
copyFileToNMSROOT (String <Application>, String <FileName>)
```

Input Arguments

Application	The name of the application suite (for example, “RME”).

FileName	File to be copied from the backup archive.
	Do not include *$NMSROOT*. For example, if the file to be restored is:
	`d:\program files\CSCOpx\cmf\objects\web\conf\ssl.conf`
	<FileName> should contain:
	`cmf\objects\web\conf\ssl.conf`

Return Values

0	Success
non-zero	Failure

copyFolderToNMSROOT

Copies the complete folder from the backup archive to NMSROOT.

Syntax

```
copyFolderToNMSROOT (String <Application>, String <Folder>)
```

Input Arguments

Application	The name of the application suite (for example, “RME”).
Folder	Folder to be copied from the backup archive. Do not include $NMSROOT.

Return Values

0	Success
non-zero	Failure

getCMFVersion

Returns the CMF or CWCS version of the backup data.

Syntax

```
getCMFVersion()
```

Input Arguments

None

Return Values

CMF version	The CMF version of the backup data. Possible values: “2.1” for CMF2.1 “2.2” for CMF2.2 “3.0” for CWCS 3.0

getCMFPatchVersion

Returns the PATCHVER of the CWCS.

Note This API is valid for CWCS 3.0 only; it cannot detect the patch version for CMF 2.1 or CMF 2.2.

Syntax

```
getCMFPatchVersion()
```

Input Arguments

None

Return Values

PATCHVER	The patch version of CWCS. Possible values are “1”, “2”, and so on.

getNMSROOT

Returns the current $NMSROOT.

Syntax

```
getNMSROOT()
```

Input Arguments

None

Return Value

path	The current $NMSROOT (the path where CiscoWorks is installed).

getArchiveNMSROOT

Returns the $NMSROOT of the backup data.

Syntax

```
getArchiveNMSROOT()
```

Input Arguments

None

Return Value

path	The $NMSROOT of the backup data (relative to the path where CiscoWorks is installed).

getFolderSeperator

Returns the folder separator depending on the OS in which CWCS 3.0 is running.

Syntax

```
getFolderSeperator()
```

Input Arguments

None

Return Values

separator	Folder separator. Possible values: "/" for Solaris "\\" for Windows

getLogFileName

Returns the full path of the restorebackup.log.

Syntax

```
getLogFileName()
```

Input Arguments

None

Return Values

filename	The location of the log file, restorebackup.log.

getTempFolder

Returns the temporary directory.

Syntax

```
getTempFolder()
```

Input Arguments

None

Return Values

folder name	By default, the temporary directory for restore is $NMSROOT/tempBackupData. However, if the temporary path has been customized (using the -t parameter), the user-specified temporary path is returned.

isWindows

Returns the OS type.

Syntax

```
isWindows()
```

Input Arguments

None

Return Values

OS type	The operating system type. Possible values are “0” for Windows, “1” for Solaris

redirectToScreen

Reverses the redirectToLog() API, restoring STDOUT and STDERR to their default values. Reverses the redirectToLog() API,

Note Applications should not use this API unless they need user inputs.

Syntax

```
redirectToScreen()
```

Input Arguments

None

Return Values

0	Success
non-zero	Failure

Example

By default, all messages produced by the applications are redirected to restorebackup.log. However, if an application adaptor needs to display some information on the screen to get input from a user, it should use the redirectToScreen() and redirectToLog() APIs. For example:

```
sub doRestore()
{
    ...
    print "<some print message>; # this will be redirected to the log
    redirectToScreen();
    print "<Print the message which should be displayed on the screen,
        which is used for confirmation by users (y/n) >";
    # the above print message will be printed on the screen.
    redirectToLog();
    # Any subsequent print statements will be redirected to the log only
    print "<other print messages>"; # will be redirected to the log
}
```

Related Topics

See the “redirectToLog” section on page 12-17.

redirectToLog

Directs STDOUT and STDERR to the restore log file. Subsequent outputs of STDOUT and STDERR are captured in the log file.

Note Applications should not use this API unless they need user inputs.

Syntax

```
redirectToLog()
```

Input Arguments

None

Return Values

0	Success
non-zero	Failure

Related Topics

See the "redirectToScreen" section on page 12-16.

restoreDatabase

The restoreDatabase API performs the following operations:

- Copies the backup database to the current database.
- Uses the password of the database being restored and updates all files accordingly.
- If the backup database password is not encrypted, it is encrypted and stored during the restore process (provided the current database configuration supports database password encryption).
- If the restore is across CMF versions (and therefore across application versions), the database is rebuilt to the current Sybase file format.

Any other operations must be handled in the application wrapper.

Syntax

```
restoreDatbase(<dsn>,<dmprefix>,[<suite>])
```

Input Arguments

dsn	Database data source name.

dmprefix	The daemon manager prefix. This is the prefix that is used to register the database with the Daemon Manager.
suite	Application suite as registered with the backup framework. If the dsn and the suite are different, then the suite name must be provided. If the suite name is not provided, the dsn is assumed to be the suite name. The suite name and dsn are usually the same, with some exceptions such as Campus Manager, where the suite is “campus” and the dsn is “ani”.

Return Values

0	Success
1	Error

StandardDbRebuild

Upgrades the database file format for a given suite or database data source name. This API unloads the data, creates a new database file, and loads the data back into this file.

This API is called internally in restoreDatabase() API when a restore across versions is detected. Applications can use it as necessary , but are not required to call it separately during a restore operation. For more information, see the “configureDb.pl” section on page 11-53.

Syntax

```
InstallUtility::StandardDbRebuild(<suite>)
```

Input Arguments

suite	Database data source name or the suite name

Return Values

None.

Restoring a Corrupt Database

If the database has been corrupted, choose one of these solutions:

- Option 1: Restoring a Corrupt Database from a Previous Backup
- Option 2: Recovering Part of a Corrupt Database
- Option 3: Abandoning a Corrupt Database

Restoring a Corrupt Database from a Previous Backup

Use the backup and restore utilities for regular database maintenance.

Note Be sure to back up your database regularly.

Related Topics

- See the "backup.pl" section on page 12-10.
- See the "restorebackup.pl" section on page 12-11.

Recovering Part of a Corrupt Database

If only part of a database is corrupted, perform the following steps to save your data. Use the procedure appropriate to the platform:

Recovering Part of a Corrupt Database On Windows Platforms

Step 1 Log in as local administrator.

Step 2 To stop the Daemon Manager, enter:

```
net stop crmdmgtd
```

Step 3 Copy the database files, including any log files, to a save directory. Always back up the original data.

Step 4 If the dbvalid utility reports that one or more tables are corrupted, try replacing just those tables.

For example, assume that the Essentials (RME) syslog tables SLG_MSG_UMGD and SLG_MSG are corrupted, which means that the data in these tables is gone. All you can do is remove the bad tables and replace them with the copies in the /orig directory. Then rerun dbvalid:

```
rm $NMSROOT/objects/db/syslog.db
copy $NMSROOT/objects/db/orig/syslog.dborig $NMSROOT/objects/db/.
dbeng8 -f rme.db
dbvalid -c "uid=dba;pwd=c2kY2k;dbf=rme.db"
```

where *$NMSROOT* is the directory in which the product was installed.

Step 5 If replacing the corrupted tables doesn't work, try extracting the data from the database and reloading it into a new database. Using the same example from Step 4:

```
cd $NMSROOT/databases/rme
dbunload -c "uid=dba;pwd=c2kY2k;dbf=rme.db" savedir
```

Notice the file reload.sql under the current directory and the *.data files under savedir.

Now create the rme.db and syslog.db files, then rerun dbvalid:

```
rm -f rme.db, syslog.db, rme.log
dbinit -p 4096 rme.db
dbisqlc -c "uid=dba;pwd=sql;dbf=rme.db" -q read reload.sql
```

The reload utility reloads the database and restores the old user ID and password values.

Note The reload process might take a long time. Depending on the size of the database, reloading may run for several hours.

Then rerun dbvalid:

```
dbvalid -c "uid=dba;pwd=c2kY2k;dbf=rme.db"
```

Step 6 To restart the Daemon Manager, enter:

```
net start crmdmgtd
```

Step 7 Optional: Use dbreader to examine the contents of the database (see the "Examining the Contents of a Database" section on page 11-31).

Recovering Part of a Corrupt Database On Solaris Platforms

Step 1 Log in as root.

Step 2 To stop the Daemon Manager, enter:

```
/etc/init.d/dmgtd stop
```

Step 3 Copy the database files, including any log files, to a save directory. Always back up the original data.

Step 4 Set the environment variables (see the "Setting Up Your Environment" section on page 11-19).

Step 5 Create a directory to keep the data:

```
mkdir savedir
dbunload -c "uid=$uid;pwd=$pwd;dbf=$dbfile.db" savedir
```

where *$uid* and *$pwd* are the user ID and password for your database.

Notice the file reload.sql under the current directory and the *.data files under savedir.

Step 6 To create the $dbfile.db file, enter:

```
rm -f $dbfile.db $dbfile.log
```

Step 7 To initialize the $dbfile.db file, enter:

```
dbinit -p 4096 $dbfile.db
dbisqlc -c "uid=$uid;pwd=$pwd;dbf=$dbfile.db" -q read reload.sql
```

where *$uid* and *$pwd* are the default user ID and password values (dba and sql).

The reload utility reloads the database and restores the old user ID and password values (used in Step 4).

Step 8 Optional: Use dbvalid to determine if the database is valid (see the "dbvalid" section on page 11-60).

Step 9 To restart the Daemon Manager, enter:

```
/etc/init.d/dmgtd start
```

Step 10 Optional: Use dbreader to examine the contents of the database (see the "Examining the Contents of a Database" section on page 11-31).

Abandoning a Corrupt Database

If the data in the database is not important, or the database is totally corrupted, you can copy a clean database from the orig directory. Assuming that the database file is the only problem, this, at least, will allow you to avoid a reinstallation.

To reinitialize the database, use the dbRestoreOrig.pl script (see the "dbRestoreOrig" section on page 11-59).