# Prime Cable Provisioning Support Tools

This section contains information on, and explains the use of tools that help you maintain Prime Cable Provisioning as well as speed and improve the installation, deployment, and use of this product.

**Note** This section contains several examples of tool use. In many cases, the tool filenames include a path specified as *BPR_HOME*. This indicates the default home directory location.

This section discusses:


- Prime Cable Provisioning Tools, on page 1
- **RDU Export Import Tool** , on page 3
- Using PKCert.sh, on page 8
- Using KeyGen Tool, on page 15
- Using changeNRProperties.sh, on page 17
- Using disk_monitor.sh, on page 19
- Using runEventMonitor.sh Tool, on page 20
- Using rdu.properties, on page 23
- Using adminui.properties, on page 24
- Using verifydb.sh Tool, on page 25
- Using passwordEncryption.sh, on page 26
- Using changeSSLProperties.sh, on page 27
- Using ws-cli.sh, on page 30
- Scripts to Manage and Troubleshoot RDU Redundancy, on page 31
- Using deviceReader Tool, on page 34
- Using Live DB Compaction Tool, on page 36
- DPE Event Publisher, on page 40


# Prime Cable Provisioning Tools

Prime Cable Provisioning provides automated tools that you use to perform certain functions more efficiently. The following table lists the various tools that this Prime Cable Provisioning release supports.

*Table 1: Prime Cable Provisioning Tools*

| Tool | Description | Refer... |
|---|---|---|
| Configuration File Utility | Used to test, validate, and view Prime Cable Provisioning template and configuration files. | Using Configuration File Utility for Template |
| Prime Cable Provisioning Process Watchdog | Interacts with the Prime Cable Provisioning watchdog daemon to observe the status of the Prime Cable Provisioningsystem components, and stop or start servers. | Using Prime Cable Provisioning Process Watchdog from CLI |
| RDU Log Level Tool | Sets the log level of the RDU, and enables or disables debugging log output. | Using the RDU Log Level Tool |
| PacketCable Certificates Tool | Installs, and manages, the KDC certificates that are required by the KDC for its operation. | Using PKCert.sh |
| KeyGen Tool | Generates PacketCable service keys. | Using KeyGen Tool |
| Changing Network Registrar Properties Tool | Used to change key configuration and SSL related properties used by Prime Cable Provisioning extensions that are incorporated into the Prime Network Registrar DHCP server. | Using changeNRProperties.sh |
| SNMP Agent Configuration Tool | Manages the SNMP agent. | Using PKCert.sh |
| Diagnostics Tool | Collects server data related to system performance and troubleshooting. | Troubleshooting Using Diagnostics Tool |
| BundleState.sh Tool | Bundles diagnostics data related to server state for support escalations. | Bundling Server State for Support |
| Disk Space Monitoring Tool | Sets threshold values for one or more file systems. When these thresholds are surpassed, an alert is generated until additional disk space is available. | Using disk_monitor.sh |
| changeSSLProperties.sh Tool | Used to change various SSL properties like enable or disable SSL connection, change secure key, secure port number, secret key and key password. | Using changeSSLProperties.sh, on page 27 |

| Tool | Description | Refer... |
|---|---|---|
| ws-cli.sh Tool | Used to change key PWS configuration properties like adding and deleting RDU accounts, changing the log severity level. | Using ws-cli.sh |
| RDU Export Tool | The Export tool which is located at $BPR_HOME/rdu/internal/db/bin , can be used to export the group of devices from a RDU to an intermediate database. | RDU Export Import Tool , on page 3 |
| RDU Import Tool | The Import tool is located in $BPR_HOME/rdu/internal/db/bin. This tool can be used to import all the devices from the intermediate database (generated by the export tool) to target RDU database. | RDU Export Import Tool , on page 3 |
| Delete Tool | The Delete tool which is located in $BPR_HOME/rdu/internal/db/bin , can be used to delete the devices from the source RDU that are exported to the intermediate database. | RDU Export Import Tool , on page 3 |
| deviceReader Tool | It reads the device objects along with the associated resources, CoS, DHCP criteria and extracts the device details from a RDU database. | Using deviceReader Tool, on page 34 |
| Live DB Compaction Tool | The Live DB Compaction tool is used to compress the RDU database without stopping the RDU. | Using Live DB Compaction Tool, on page 36 |

# RDU Export Import Tool

The tool allows user to export and import device data from one RDU to another. The exported device data includes DHCP discovered information, which allows the service provider to seamlessly migrate devices between the RDUs. The Export Tool provides a filter based support which allows service provides to move devices based on Provisioning Group(PG) or a "giaddr".

The RDU Export Import Tool is platform independent. For instance, the tool allows user to export data from a RDU running on Solaris platform and import the data to a RDU running on a Linux platform.

**Note** While migrating the device data from one RDU to another RDU by using the Export Import Tool, the template files used in another template file will not be exported or imported. So, it is recommended to migrate all the resources from the source database to target database before migrating the device data.

### Export Tool:

The export tool which is located at *$BPR_HOME/rdu/internal/db/bin*, can be used to export the group of devices from a RDU to an intermediate database. The devices to export can be filtered by providing:

| | |
|---|---|
| 1. Provisioning Group | The devices which belong to the provisioning group will be exported.. |
| 2. Provisioning Group and a giaddr | The devices under a provisioning group which has the specified giaddr will be exported. |

The help option (exportTools.sh -help) of the Export Tool will provide the different options available for the tool.

### Parameters:

| | |
|---|---|
| **-dbdir** | The directory from which the devices are to be exported is provided with the **-dbdir** parameter. The default location will be the RDU's $BPR_DATA directory. |
| **-dblogdir** | This optional parameter mentions the dblog directory of the source database. |
| **-targetdbdir** | The directory where the intermediate database should be created will be provided with the **-targetdbdir** |
| **-targetdblogdir** | This optional parameter mentions the dblog directory of the intermediate database. |
| **-pg** | This is a required parameter to export the devices in that provisioning group. |
| **-giaddr** | This is an optional parameter. Devices with the specified giaddr will be exported. |
| **-expaddrdir** | This optional parameter creates a directory in a specified location which consists of MAC file and DUID file. MAC file contains the MAC addresses of the exported devices. The DUID file contains the DUID addresses of the exported devices which doesn't have the MAC addresses. |
| **-logfile** | This is a required parameter specifies the location of the log file for the exportTool |
| **-help** | Help option to display the options the tool supports |

### SAMPLE USAGE:

There are 2 ways to filter devices that are to be exported from the source RDU.

1. **Filtering using Provisioning Group (PG):**

   The following command can be used to export devices and its data by filtering based on provisioning group

```
./exportTool.sh -dbdir <source_dir_path> -targetdbdir <intermediate_db_path> -pg <PG_id>
-expaddrdir <location_where_MAC_and_DUID_file_to_be_generated> -logfile
<export_tool_logfile>
```

2. **Filtering using giaddr:**

   The following command can be used to export devices and its data by filtering based on giaddr in a provisioning group.

```
./exportTool.sh -dbdir <source_dir_path> -targetdbdir <intermediate_db_path> -pg <PG_id>
-giaddr <giaddr> -expaddrdir
<location_where_MAC_and_DUID_file_to_be_generated>-logfile<export_tool_logfile
```

### Import Tool:

The import tool is located in $BPR_HOME/rdu/internal/db/bin. This tool can be used to import all the devices from the intermediate database (generated by the export tool) to target RDU database. There are options to resolve name conflicts in resources (File, CoS, DHCPCriteria or OwnerID) between source and target databases.

The help option (importTool.sh -help) of the Import Tool will display list of menu options available for the tool.

***Table 2: Basic Parameters:***

| | |
|---|---|
| **-dbdir** | The **-dbdir** is a required parameter for providing the path of the intermediate database generated by the Export Tool. |
| **-dblogdir** | This optional parameter mentions the dblog directory of the intermediate database. |
| **-targetdbdir** | This is the required parameter which mentions the target database to which the devices should be imported. This should be a backup snapshot of the target RDU. |
| **-targetdblogdir** | This optional parameter mentions the dblog directory of the target database. |
| **-logfile** | The **-logfile** is a required parameter to log the Import Tool logs. |
| **-help** | Help option to display the options the tool support |

**Conflict resolving Parameters:**

If a resource (File,CoS, DHCPCriteria or OwnerID) with same name is already available in target database then conflicts might arise. The following parameters can be used to handle the name conflict scenarios during an import and to take appropriate actions for conflicting objects..

| | |
|---|---|
| **-reportconflicts** | This parameter is to generate a report of name conflicts between source and target database objects.<br>This option generates a configuration file which can be used as an input for**-prefixfile**option. |
| **-prefix** | The value passed along with this parameter will be used to create a new resource on target database with the prefixed name for the conflicting objects. |

| | |
|---|---|
| **-prefixfile** | This optional parameter controls the way in which **-prefix** option should selectively prefix only specific conflicting objects or prefix for all the conflicting objects.<br>This parameter should be followed by the location of the configuration file generated by the **-reportconflicts** option.<br>If the **-prefixfile** option is provided then the prefix functionality which is mentioned above will only be applicable to the conflicting objects available in this configuration file.<br>If this **-prefixfile** option is not provided along with the prefix option then the prefix functionality will be applicable to all the conflicting objects. |
| **-forcecreate** | This is an optional parameter to be used along with the –prefix option.<br>If there is a name conflict with the prefixed name, **-forcecreate** option creates unique name by adding sequence number to the prefixed name. |
| **-fileconflicts** | This parameter is used to check the file name conflicts which will be imported.<br>This parameter should be followed by the location of the report conflicts file generated by the **-reportconflicts** option.<br>If there is a name conflict with the prefixed owner ID relationship, then it will exit the import the database. |

### SAMPLE USAGE:

**./importTool.sh -dbdir <intermediate_db_dir_path> -targetdbdir <backup_ dir_path> -logfile <log_path>**

The above command will import the devices and resources from the `<intermediate_db_dir_path>` to the `< backup_dir_path >`.

Since prefix option is not provided here, the name conflicting resources will not be imported and the resources from the target database will be mapped for the imported devices. This default behavior can be changed by using conflict resolving parameters.

**./importTool.sh -dbdir <intermediate_db_dir_path> -targetdbdir <backup_ dir_path> -logfile <log_path> –reportconflicts**

The above command will generate a configuration which can be used as an input for *-prefixfile* and *-fileconflicts* options.

### Delete Tool:

The delete tool which is located in $BPR_HOME/rdu/internal/db/bin , can be used to delete the devices from the source RDU that are exported to the intermediate database. The exported devices in the source RDU can be deleted by using the following inputs:

1. **Intermediate database**
2. **MAC File and DUID File**

The help option (deleteTool.sh -help) of the Delete Tool will provide the different options available for the tool.

### Parameters:

| | |
|---|---|
| **-inputdbdir** | A required parameter if **-inputmacfile** or **-inputduidfile** is not specified. This is the input database directory path. This will be the intermediate database directory created by the export tool. If specified, tool reads devices from this database for deletion. |
| **-inputdblogdir** | An optional parameter with input database log directory path, if **-inputdbdir** is used. If not specified, directory specified with **-inputdbdir** parameter is used by default. |
| **-inputmacfile** | A required parameter if **-inputdbdir** or **-inputduidfile** is not specified.<br>If specified, tool reads MAC from this file for deletion. |
| **-inputduidfile** | A required parameter if **-inputdbdir** or **-inputmacfile** is not specified.<br>If specified, tool reads DUID from this file for deletion. |
| **-dbdir** | An optional parameter which is the database directory in which the devices will be deleted. This should be the database from which the devices were exported using the export tool. If not specified, RDU database location is used by default. |
| **-dblogdir** | An optional parameter with database log directory path. If not specified, directory specified with **-dbdir** or RDU database location is used by default. |
| **-cachesize** | An optional parameter that specifies cache size in MB for the db. If not specified, the default cache size is 100MB |

| | |
|---|---|
| **-includeCPEs** | An optional parameter which specifies to delete behind devices.<br>Applicable when **-inputmacfile** parameter is used.<br>The behind devices will not be deleted by default, unless the *behind device MAC* is included in the *inputMACfile* or option **-includeCPEs** is given as an input. |
| **-help** | Help option to display the options the tool support |

**SAMPLE USAGE:**

1. **To delete using intermediate database:**

   The following command shall be used to delete using intermediate database:

   ```
   ./deleteTool.sh -dbdir <source_dir_path> -inputdbdir <intermediate_db_dir_path>
   ```

2. **To delete using MAC and DUID files:**

   The following command shall be used to delete devices from source RDU using MAC file:

   ```
   ./deleteTool.sh -dbdir <source_dir_path> -inputmacfile <mac_file>
   ```

**Note** By default the behind devices will be deleted automatically if the MAC file generated during export is provided as an input to the deleteTool.

The following command shall be used to delete devices from source RDU using DUID file:

```
./deleteTool.sh -dbdir <source_dir_path> -inputduidfile <duid_file>
```

# Using PKCert.sh

The PKCert tool creates the KDC certificate and its corresponding private key. It also allows you to verify certificate chains and copy and rename a certificate chain to the names required by the KDC.

**Note** This tool is available only when the KDC component is installed.

## Running PKCert Tool

Run the PKCert tool by executing the PKCert.sh command, which resides by default in the *BPR_HOME/kdc* directory.

**Syntax Description**

**PKCert.sh** *function option*

- *function*—Identifies the function to be performed. You can choose:

- **-c**—Creates a KDC certificate. See Creating a KDC Certificate.
- **-v**—Verifies and normalizes the PacketCable certificate set. See Validating KDC Certificates.
- **-z**—Sets the log level for debug output that is stored in the *pkcert.log* file. See Setting Log Level for Debug Output.

**Note** If you have trouble using these options, specify **-?** to display available help information.

- *option*—Implements optional functions, depending on the function you selected.

# Creating a KDC Certificate

To create the KDC certificate:

**Step 1** Change directory to */opt/CSCObac/kdc*.

**Step 2** Run the PKCert.sh tool using this syntax:

**PKCert.sh -s** *dir* **-d** *dir* **-c** *cert* **-e -r** *realm* **-a** *name* **-k** *keyFile* [**-n** *serial#*] [**-o**]

- **-s** *dir*—Specifies the source directory
- **-d** *dir*—Specifies the destination directory
- **-c** *cert*—Uses the service provider certificate (DER encoded)
- **-e**—Identifies the certificate as a Euro-PacketCable certificate
- **-r** *realm*—Specifies the Kerberos realm for the KDC certificate
- **-a** *name*—Specifies the DNS name of the KDC
- **-k** *keyFile*—Uses the service provider private key (DER encoded)
- **-n** *serial#*—Sets the certificate serial number
- **-o**—Overwrites existing files

When a new certificate is created and installed, the new certificate identifies the realm in the subject alternate name field. The new certificate is unique to its current environment in that it contains the:

- KDC realm.
- DNS name associated with this KDC that the Multimedia Terminal Adapter (MTA) will use.

Examples

```
# ./PKCert.sh  -c "-s . -d /opt/CSCObac/kdc/<Operating System>/packetcable/certificates
-k CLCerts/Test_LSCA_privkey.der -c CLCerts/Test_LSCA.cer -r PCTEST.CISCO.COM -n 100
-a kdc.pctest.cisco.com -o"
Pkcert Version 1.0
Logging to pkcert.log
Source Directory: .
```

```
Destination Directory: /opt/CSCObac/kdc/<Operating System>/packetcable/certificates
Private Key File: CLCerts/Test_LSCA_privkey.der
Certificate File: CLCerts/Test_LSCA.cer
Realm: PCTEST.CISCO.COM
Serial Number: 100
DNS Name of KDC: kdc.pctest.cisco.com
WARNING - Certificate File will be overwritten
SP Cert subject name: C=US,O=CableLabs\, Inc.,OU=ABC Cable Company,CN=Shared-01 CableLabs Local
 System CA
File written: /opt/CSCObac/kdc/<Operating System>/packetcable/certificates/KDC_private_key.pkcs8
File written: /opt/CSCObac/kdc/<Operating
System>/packetcable/certificates/KDC_private_key_proprietary.
File written: /opt/CSCObac/kdc/<Operating System>/packetcable/certificates/KDC_PublicKey.der
File written: /opt/CSCObac/kdc/<Operating System>/packetcable/certificates/KDC.cer
KDC Certificate Successfully Created at /opt/CSCObac/kdc/<Operating
System>/packetcable/certificates/KDC.cer
```

This command creates the following files:

- */opt/CSCObac/kdc/<Operating System>/packetcable/certificates/KDC.cer*
- */opt/CSCObac/kdc/<Operating System>/packetcable/certificates/KDC_private_key.pkcs8*.

The KDC certificate will have a realm set to PCTEST.CISCO.COM, a serial number set to 100, and the fully qualified domain name (FQDN) of the KDC server set to kdc.pctest.cisco.com.

# Validating KDC Certificates

This command examines all files in the source directory specified and attempts to identify them as X.509 certificates. If legitimate X.509 certificates are found, the files are properly renamed and copied to the destination directory. An error is generated when more than one legitimate chain of certificates for a particular purpose (service provider or device) is identified. If this occurs, you must remove the extra certificate from the source directory and run the command again.

**Note** When you enter the **PKCert.sh -v -?** command, usage instructions for validating KDC certificates by using the PKCert tool appear.

To validate the KDC certificate:

**Step 1** Change directory to */opt/CSCObac/kdc*.

**Step 2** Run the PKCert.sh tool using this syntax:

**PKCert.sh -v -s** *dir* **-d** *dir* **-r** *dir* **-e**

- **-s** *dir*—Specifies the source directory
- **-d** *dir*—Specifies the destination directory
- **-o**—Overwrites any existing files
- **-r** *dir*—Specifies the reference certificate directory

- **-e**—Identifies the certificate as a Euro-PacketCable certificate

  Verification is performed against reference certificates built into this package. If you specify the **-d** option, the certificates are installed in the target directory with name normalization.

  Examples

```
# ./PKCert.sh  -v "-s /opt/CSCObac/kdc/TestCerts -d /opt/CSCObac/kdc/<Operating
System>/packetcable/certificates -o"
Pkcert Version 1.0
Logging to pkcert.log
Output files will overwrite existing files in destination directory

Cert Chain(0)    Chain Type: Service Provider
[Local File]                          [Certificate Label]                    [PacketCable Name]
CableLabs_Service_Provider_Root.cer   CableLabs_Service_Provider_Root.cer
Service_Provider.cer                  Service_Provider.cer
Local_System.cer                      Local_System.cer
KDC.cer                               KDC.cer

Cert Chain(1)    Chain Type: Device
[Local File]                          [Certificate Label]                    [PacketCable Name]
MTA_Root.cer                          MTA_Root.cer
File written: /opt/CSCObac/kdc/<Operating
System>/packetcable/certificates/CableLabs_Service_Provider_Root.cer
File written: /opt/CSCObac/kdc/<Operating System>/packetcable/certificates/Service_Provider.cer
File written: /opt/CSCObac/kdc/<Operating System>/packetcable/certificates/Local_System.cer
File written: /opt/CSCObac/kdc/<Operating System>/packetcable/certificates/KDC.cer

Service Provider Certificate Chain Written to Destination Directory /opt/CSCObac/kdc/<Operating
 System>/packetcable/certificates


File written: /opt/CSCObac/kdc/<Operating System>/packetcable/certificates/MTA_Root.cer

Device Certificate Chain Written to Destination Directory /opt/CSCObac/kdc/<Operating
System>/packetcable/certificates
```

# Setting Log Level for Debug Output

This command enables you to set the log level for debug output that is logged in *pkcert.log,* which resides in *BPR_HOME/kdc*. You can use the data in the log file to troubleshoot any problems that may have occurred while performing the requested tasks.

To set the log level for debug output:

**Step 1** Change directory to */opt/CSCObac/kdc*.

**Step 2** Run the PKCert.sh tool using this syntax:

**PKCert.sh -s** *dir* **-d** *dir* **-k** *keyFile* **-c** *cert* **-r** *realm* **-a** *name* **-n** *serial#* **-o** {**-z error** | **info** | **debug**}

- **-s** *dir*—Specifies the source directory
- **-d** *dir*—Specifies the destination directory
- **-k** *keyFile*—Uses the service provider private key (DER encoded)

- **-c** *cert*—Uses the service provider certificate (DER encoded)
- **-r** *realm*—Specifies the Kerberos realm for the KDC certificate
- **-a** *name*—Specifies the DNS name of the KDC
- **-n** *serial#*—Sets the certificate serial number
- **-o**—Overwrites existing files
- **-z**—Sets the log level for debug output that is stored in the *pkcert.log* file. The values you can choose are:
  - **error**—Specifies the logging of error messages.
  - **info**—Specifies the logging of informational messages.
  - **debug**—Specifies the logging of debug messages. This is the default setting.

### Example

#### Example 1

In this example, the log level is set for collecting error messages.

```
# ./PKCert.sh -c "-s /var/certsInput -d /var/certsOutput -k /var/certsInput/Local_System.der
 -c /var/certsInput/Local_System.cer -r PCTEST.CISCO.COM -n 100 -a kdc.pctest.cisco.com -o
 -z error"
Pkcert Version 1.0
Logging to pkcert.log
Source Directory: /var/certsInput
Destination Directory: /var/certsOutput
Private Key File: /var/certsInput/Local_System.der
Certificate File: /var/certsInput/Local_System.cer
Realm: PCTEST.CISCO.COM
Serial Number: 100
DNS Name of KDC: kdc.pctest.cisco.com
Setting debug to error
WARNING - Certificate File will be overwritten
SP Cert subject name: C=US,O=CableLabs\, Inc.,OU=ABC Cable Company,CN=Shared-01 CableLabs
Local System CA
File written: /var/certsOutput/KDC_private_key.pkcs8
File written: /var/certsOutput/KDC_private_key_proprietary.
File written: /var/certsOutput/KDC_PublicKey.der
File written: /var/certsOutput/KDC.cer
KDC Certificate Successfully Created at /var/certsOutput/KDC.cer

Copy KDC.cer to the KDC certificate directory (i.e.
/opt/CSCObac/kdc/linux/packetcable/certificates)
Copy KDC_private_key.pkcs8 to the KDC platform directory (i.e. /opt/CSCObac/kdc/linux)
Copy KDC_private_key_proprietary. to the KDC platform directory (i.e. /opt/CSCObac/kdc/linux)
```

#### Example 2

In this example, the log level is set for collecting information messages.

```
# ./PKCert.sh -c "-s /var/certsInput
> -d /var/certsOutput
```

```
> -k /var/certsInput/Local_System.der
> -c /var/certsInput/Local_System.cer
> -r PCTEST.CISCO.COM
> -n 100
> -a kdc.pctest.cisco.com
> -o -z info"
 INFO [main] 2007-05-02 06:32:26,280 (PKCert.java:97) - Pkcert Version 1.0
Pkcert Version 1.0
Logging to pkcert.log
Source Directory: /var/certsInput
Destination Directory: /var/certsOutput
Private Key File: /var/certsInput/Local_System.der
Certificate File: /var/certsInput/Local_System.cer
Realm: PCTEST.CISCO.COM
Serial Number: 100
DNS Name of KDC: kdc.pctest.cisco.com
Setting debug to info
 INFO [main] 2007-05-02 06:32:26,289 (PKCCreate.java:69) - PKCCreate startup
WARNING - Certificate File will be overwritten
 INFO [main] 2007-05-02 06:32:26,291 (PKCCreate.java:341) - WARNING - Certificate File will
 be overwritten
SP Cert subject name: C=US,O=CableLabs\, Inc.,OU=ABC Cable Company,CN=Shared-01 CableLabs
Local System CA
File written: /var/certsOutput/KDC_private_key.pkcs8
File written: /var/certsOutput/KDC_private_key_proprietary.
File written: /var/certsOutput/KDC_PublicKey.der
File written: /var/certsOutput/KDC.cer
KDC Certificate Successfully Created at /var/certsOutput/KDC.cer

Copy KDC.cer to the KDC certificate directory (i.e.
/opt/CSCObac/kdc/linux/packetcable/certificates)
Copy KDC_private_key.pkcs8 to the KDC platform directory (i.e. /opt/CSCObac/kdc/linux)
Copy KDC_private_key_proprietary. to the KDC platform directory (i.e. /opt/CSCObac/kdc/linux)
```

### Example 3

In this example, the log level is set for debugging.

**Note** The sample output has been trimmed for demonstration purposes.

```
# ./PKCert.sh -c "-s /var/certsInput -d /var/certsOutput -k /var/certsInput/Local_System.der
 -c /var/certsInput/Local_System.cer -r PCTEST.CISCO.COM -n 100 -a kdc.pctest.cisco.com -o
 -z debug"
 INFO [main] 2007-05-02 06:32:06,029 (PKCert.java:97) - Pkcert Version 1.0
Pkcert Version 1.0
Logging to pkcert.log
Source Directory: /var/certsInput
Destination Directory: /var/certsOutput
Private Key File: /var/certsInput/Local_System.der
Certificate File: /var/certsInput/Local_System.cer
Realm: IPFONIX.COM
Serial Number: 100
DNS Name of KDC: bacdev3-dpe-4.cisco.com
Setting debug to debug
 INFO [main] 2007-05-02 06:32:06,038 (PKCCreate.java:69) - PKCCreate startup
WARNING - Certificate File will be overwritten
 INFO [main] 2007-05-02 06:32:06,039 (PKCCreate.java:341) - WARNING - Certificate File will
 be overwritten
DEBUG [main] 2007-05-02 06:32:06,054 (PKCert.java:553) - Characters Read: 1218
```

```
DEBUG [main] 2007-05-02 06:32:06,056 (PKCert.java:583) - Binary File:
/var/certsInput/Local_System.der Read. Length: 1218
DEBUG [main] 2007-05-02 06:32:06,062 (PKCert.java:553) - Characters Read: 943
DEBUG [main] 2007-05-02 06:32:06,063 (PKCert.java:583) - Binary File:
/var/certsInput/Local_System.cer Read. Length: 943
DEBUG [main] 2007-05-02 06:32:06,064 (PKCert.java:455) - Jar File Path:
/opt/CSCObac/lib/pkcerts.jar
DEBUG [main] 2007-05-02 06:32:06,065 (PKCert.java:456) - Opened jar file:
/opt/CSCObac/lib/pkcerts.jar
DEBUG [main] 2007-05-02 06:32:06,067 (PKCert.java:460) - Jar entry unfiltered:
Tag_Packetcable_Tag/
DEBUG [main] 2007-05-02 06:32:06,068 (PKCert.java:460) - Jar entry unfiltered:
Tag_Packetcable_Tag/CableLabs_Service_Provider_Root.cer
...
DEBUG [main] 2007-05-02 06:32:06,115 (PKCert.java:472) - File: Tag_Packetcable_Tag/Manu.cer
DEBUG [main] 2007-05-02 06:32:06,116 (PKCert.java:472) - File:
Tag_Packetcable_Tag/Service_Provider.cer
DEBUG [main] 2007-05-02 06:32:06,121 (PKCCreate.java:91) - Found 7 files in jar.
DEBUG [main] 2007-05-02 06:32:06,827 (KDCCert.java:98) - SP Cert subject name:
C=US,O=CableLabs\, Inc.,OU=ABC Cable Company,CN=Shared-01 CableLabs Local System CA
SP Cert subject name: C=US,O=CableLabs\, Inc.,OU=ABC Cable Company,CN=Shared-01 CableLabs
Local System CA
DEBUG [main] 2007-05-02 06:32:07,687 (KDCCert.java:293) - Setting issuer to:
C=US,O=CableLabs\, Inc.,OU=ABC Cable Company,CN=Shared-01 CableLabs Local System CA
DEBUG [main] 2007-05-02 06:32:07,699 (KDCCert.java:231) - DERVisibleToGeneral
org.bouncycastle.asn1.DERGeneralString@bd0b4ea6

DEBUG [main] 2007-05-02 06:32:07,700 (KDCCert.java:231) - DERVisibleToGeneral
org.bouncycastle.asn1.DERGeneralString@5035bc0

DEBUG [main] 2007-05-02 06:32:07,701 (KDCCert.java:231) - DERVisibleToGeneral
org.bouncycastle.asn1.DERGeneralString@5035bc0

DEBUG [main] 2007-05-02 06:32:07,703 (KDCCert.java:210) - DERCombineTagged [0] IMPLICIT
    DER ConstructedSequence
        ObjectIdentifier(1.3.6.1.5.2.2)
        Tagged [0]
            DER ConstructedSequence
                Tagged [0]
                    org.bouncycastle.asn1.DERGeneralString@5035bc0
                Tagged [1]
                    DER ConstructedSequence
                        Tagged [0]
                            Integer(2)
                        Tagged [1]
                            DER ConstructedSequence
                                org.bouncycastle.asn1.DERGeneralString@bd0b4ea6
                                org.bouncycastle.asn1.DERGeneralString@5035bc0

File written: /var/certsOutput/KDC_private_key.pkcs8
File written: /var/certsOutput/KDC_private_key_proprietary.
File written: /var/certsOutput/KDC_PublicKey.der
File written: /var/certsOutput/KDC.cer
KDC Certificate Successfully Created at /var/certsOutput/KDC.cer

Copy KDC.cer to the KDC certificate directory (i.e.
/opt/CSCObac/kdc/linux/packetcable/certificates)
Copy KDC_private_key.pkcs8 to the KDC platform directory (i.e. /opt/CSCObac/kdc/linux)
Copy KDC_private_key_proprietary. to the KDC platform directory (i.e. /opt/CSCObac/kdc/linux)
```

# Using KeyGen Tool

The KeyGen tool is used to generate PacketCable service keys. The service keys are symmetric triple data encryption standard (triple DES or 3DES) keys (shared secret) required for KDC communication. The KDC server requires service keys for each of the provisioning FQDNs of the DPE. Any changes made to the DPE provisioning FQDN from the DPE command-line interface (CLI) requires a corresponding change to the KDC service key filename. This change is necessary because the KDC service key uses the DPE provisioning FQDN as part of its filename.

The KeyGen tool, which resides in the *BPR_HOME/kdc* directory, uses command-line arguments for the DPE provisioning FQDN, realm name, and a password, and generates the service key files.

**Note** When running this tool, remember to enter the same password that you used to generate the service key on the DPE (by using the **service packetCable** *1..1* **registration kdc-service-key** command from the DPE CLI). For information on setting this password, see the Cisco Prime Cable Provisioning 6.1.2 DPE CLI Reference Guide.

The KDC server reads the service keys on startup. Any modification to the service keys requires that you restart the KDC server.

**Syntax Description**

**keygen** *options fqdn realm password*

- *options* are:
  - **-?**—Displays this usage message and exits the command.
  - **-v** or **-version**—Displays the version of this tool and exits the command.
  - **-q** or **-quiet**—Implements a quiet mode whereby no output is created.
  - **-c** or **-cms**—Creates a service key for the CMS system.
- *fqdn*—Identifies the FQDN of the DPE and is a required entry.
- *realm*—Identifies the Kerberos realm and is a required entry.
- *password*—Specifies the password to be used. This is also a required field. The password must be from 6 to 20 characters.

Three service key files are written in the KDC keys directory using this filename syntax:

mtafqdnmap,*fqdn@REALM*

mtaprovsrvr,*fqdn@REALM*

krbtgt,*REALM@REALM*

- *fqdn*—Identifies the FQDN of the DPE.
- *REALM*—Identifies the Kerberos realm.

The service key file always contains a version field of 0x0000.

### Examples

```
# keygen dpe.cisco.com CISCO.COM changeme
```

When this command is implemented, these KDC service keys are written to the *BPR_HOME/kdc/<Operating System>/keys* directory:

```
mtafqdnmap,dpe.cisco.com@CISCO.COM
mtaprovsrvr,dpe.cisco.com@CISCO.COM
krbtgt,CISCO.COM@CISCO.COM
```

Restart the KDC, so that the new keys are recognized. Use this Prime Cable Provisioning process watchdog command to restart the KDC:

```
# /etc/init.d/bprAgent restart kdc
```

This example illustrates the generation of a CMS service key:

```
# keygen -c cms-fqdn.com CMS-REALM-NAME changeme
```

When this command is implemented, this CMS service key is written to the *BPR_HOME/kdc/<Operating System>/keys* directory.

```
cms,cms-fqdn.com@CMS-REALM-NAME
```

### Verifying the KDC Service Keys

Once you generate the service keys on the KDC and the DPE, verify if the service keys match on both components.

The KeyGen tool requires you to enter the same password that you used to generate the service key on the DPE using the **service packetCable** *1..1* **registration kdc-service-key** command. Once you set this password on the DPE, you can view the service key from the *dpe.properties* file, which resides in the *BPR_HOME/dpe/conf* directory. Look for the value against the */pktcbl/regsvr/KDCServiceKey=* property.

For example:

```
# more dpe.properties
/pktcbl/regsvr/KDCServiceKey=2e:d5:ef:e9:5a:4e:d7:06:67:dc:65:ac:bb:89:e3:2c:bb:
71:5f:22:bf:94:cf:2c
```

**Note** The output of this example has been trimmed for demonstration purposes.

To view the service key generated on the KDC, run the following command from the *BPR_HOME/kdc/<Operating System>/keys* directory:

**od -Ax -tx1 mtaprovsrvr***,fqdn@REALM*

- *fqdn*—Identifies the FQDN of the DPE.
- *REALM*—Identifies the Kerberos realm.

The output that this command generates should match the value of the */pktcbl/regsvr/KDCServiceKey=* property in the *dpe.properties* file.

For example:

```
# od -Ax -tx1 mtaprovsrvr,dpe.cisco.com@CISCO.COM
0000000 00 00 2e d5 ef e9 5a 4e d7 06 67 dc 65 ac bb 89
0000010 e3 2c bb 71 5f 22 bf 94 cf 2c
000001a
```

In the examples shown here, note that the service key generated at the KDC matches the service key on the DPE.

# Using changeNRProperties.sh

The Prime Cable Provisioning installation program establishes values for configuration properties used by Prime Cable Provisioning extensions that are incorporated into the Network Registrar DHCP server. You use the **changeNRProperties.sh** command, which is found in the *BPR_HOME/cnr_ep/bin* directory, to change key configuration properties.

Invoking the script without any parameters displays a help message listing the properties that can be set.

To run this command:

**Step 1** Change directory to *BPR_HOME/cnr_ep/bin*.

**Step 2** Run the **changeNRProperties.sh** command using this syntax:

**changeNRProperties.sh** *options*

Where *options* are:

- **-help**—Displays this help message. The **-help** option must be used exclusively. Do not use this with any other option.
- **-d**—Displays the current properties. The **-d** option must be used exclusively. Do not use this with any other option.
- **-ep enabled** | **disabled**—Enables or disables the PacketCable property. Enter **-ep enabled** to enable the property, and **-ep disabled** to disable it.
- **-epv6 enabled | disabled**—Enables or disables the PacketCable v6 property. Enter **-epv6 enabled** to enable the property, and **-epv6 disabled** to disable it.
- **-ee enabled | disabled** - sets the eRouter enabled property

  e.g. -ee enabled or -ee disabled
- **-eev6 enabled | disabled** - sets the eRouter v6 enabled property

  e.g. -eev6 enabled or -eev6 disabled
- **-ec enabled | disabled**—Enables or disables the CableHome property. Enter **-ec enabled** to enable the property, and **-ec disabled** to disable it.
- **-s** *secret*—Identifies the Prime Cable Provisioning shared secret. For example, if the shared secret is the word *secret*, enter **-s secret**.
- **-pdss <primary dss_id>**—Sets the primary DHCPv6 Server Selector for the options CL_V4OPTION_CCCV6(123) and CL_OPTION_CCCV6(2171), where <primary dss_id> is an opaque identifier and can have a maximum value of 32 bytes.

For example: -pdss FF:FF:FF:FF

- **-sdss <secodary dss_id>**—Sets the secondary DHCPv6 Server Selector for the options CL_V4OPTION_CCCV6(123) and CL_OPTION_CCCV6(2171), where <secodary dss_id> is an opaque identifier and can have a maximum value of 32 bytes.

  For example: -sdss 00:00:00:00

- **-f** *fqdn*—Identifies the RDU FQDN. For example, if you use rdu.example.com as the fully qualified domain name, enter **-f rdu.example.com**.
- **-p** *port*—Identifies the RDU port you want to use. For example, if you want to use port number 49187, enter **-p 49187**.
- **-r** *realm*—Identifies the PacketCable realm. For example, if your PacketCable realm is EXAMPLE.COM, enter **-r EXAMPLE.COM**.

  **Note** You must enter the realm in uppercase letters.

- **-g** *prov_group*—Identifies the provisioning group. For example, if you are using provisioning group called group1, enter **-g group1**.
- **-t 00** | **01**—Identifies whether or not the PacketCable TGT is set to off or on. For example, to set the TGT to off, enter **-t 00;** to set this to on, enter **-t 01**.
- **-a** *ip*—Identifies the PacketCable primary DHCP server address. For example, if the IP address of your primary DHCP server is 10.10.10.2, enter **-a 10.10.10.2**.
- **-b** *ip*—Identifies the PacketCable secondary DHCP server address. For example, if the IP address of your secondary DHCP server is 10.10.10.4, enter **-b 10.10.10.4**. You can also enter **-b null** to set a null value, if appropriate.
- **-y** *ip*—Identifies the PacketCable primary DNS server address. For example, if the IP address of the PacketCable primary DNS server is 10.10.10.6, enter **-y 10.10.10.6**.
- **-z** *ip*—Identifies the PacketCable secondary DNS server address. For example, if the IP address of your secondary DNS server is 10.10.10.8, enter **-z 10.10.10.8**. You can also enter **-z null** to set a null value, if appropriate.
- **-edns <ip>** - sets the eRouter DNS server address. It can be a single IP Address or a list of IP addresses (in CSV format). For example: -edns 192.168.4.3,192.168.5.1
- **-o** *prov_ip man_ip*—Sets the management address to use for communication with the DPE identified by the given provisioning address. For example, if the IP address of your provisioning group is 10.10.10.7, enter **-o 10.10.10.7 10.14.0.4**. You can also enter a null value, if appropriate; for example, **-o 10.10.10.7 null**.
- **-ssl**—Enables or disables CNR-EP secure mode of communication with the RDU.
- **-ckl**—Sets the rootCA.pem certificate location. By default, the certificate is stored in the BPR_HOME/lib/security directory.
- **-ckp**—Changes the keystore password.
- **-sk** *secretkey*—Updates the secret key which is configured during installation and is used with shared secret for communication.

**Step 3** Restart the DHCP server.

### Examples

This is an example of changing the Network Registrar extensions by using the NR Extensions Properties tool:

```
# /opt/CSCObac/cnr_ep_bin/changeNRProperties.sh -g primary1
RDU Port: 49187
RDU FQDN: bactst-lnx-4
RDU Secure Communication: false
Provisioning Group: primary1
Shared Secret: fgL7egT9zcYHs
Keystore Location: /opt/CSCObac/lib/security/.keystore
PacketCable V4 Enable: enabled
PacketCable V6 Enable: enabled
DSS_ID Primary: aa:aa:aa:aa
DSS_ID Secondary: dd:dd:dd:dd:dd
CableHome V4 Enable: NOT SET
CableLabs client TGT: 01
CableLabs client Realm: CISCO.COM
CableLabs client Primary DHCP Server: 10.81.90.90
CableLabs client Secondary DHCP Server: NOT SET
CableLabs client Primary DNS Server: 10.81.90.90
CableLabs client Secondary DNS Server: NOT SET
```

**Note** You must restart your Prime Network Registrar DHCP server for the changes to take effect.

This is an example of viewing the current properties:

```
#  opt/CSCObac/cnr_ep/bin/changeNRProperties.sh -d
Current NR Properties:
RDU Port: 49187
RDU FQDN: bactst-lnx-4
RDU Secure Communication: false
Provisioning Group: default
Shared Secret: fgL7egT9zcYHs
Keystore Location: /opt/CSCObac/lib/security/.keystore
PacketCable V4 Enable: enabled
PacketCable V6 Enable: enabled
DSS_ID Primary: aa:aa:aa:aa
DSS_ID Secondary: dd:dd:dd:dd:dd
CableHome V4 Enable: NOT SET
CableLabs client TGT: 01
CableLabs client Realm: CISCO.COM
CableLabs client Primary DHCP Server: 10.81.90.90
CableLabs client Secondary DHCP Server: NOT SET
CableLabs client Primary DNS Server: 10.81.90.90
CableLabs client Secondary DNS Server: NOT SET
```

# Using disk_monitor.sh

Monitoring available disk space is an important system administration task. You can use a number of custom written scripts or commercially available tools to do so.

The **disk_monitor.sh** command, which resides in the *BPR_HOME/rdu/samples/tools* directory, sets threshold values for one or more file systems. When these thresholds are surpassed, an alert is generated through the Solaris syslog facility, at 60-second intervals, until additional disk space is available.

**Note** We recommend that, at a minimum, you use the **disk_monitor.sh** script to monitor the *BPR_DATA* and *BPR_DBLOG* directories.

**Syntax Description**

**disk_monitor.sh** *filesystem-directory x [filesystem-directory* x*]*

- *filesystem-directory*—Identifies any directory in a file system to monitor.
- x—Identifies the percentage threshold applied to the specified file system.
- *filesystem-directory**—Identifies multiple file systems.
- *x**—Specifies percentage thresholds to be applied to multiple file systems.

**Example 1**

This example specifies that a notification be sent out when the */var/CSCObac* file system reaches 80 percent of its capacity.

```
# ./disk_monitor.sh /var/CSCObac 80
```

When the database logs disk space reaches 80-percent capacity, an alert similar to the following one is sent to the syslog file:

```
Dec 7 8:16:06 perf-u80-1 BPR: [ID 702911 local6.warning] File system /var/bpr usage is 81%
 (threshold is 80%)
```

**Example 2**

This example describes how you can run the disk_monitor.sh tool as a background process. Specifying an ampersand (&) at the end of the command immediately returns output while running the process in the background.

```
# ./disk_monitor.sh /var/CSCObac 80 &
1020
```

# Using runEventMonitor.sh Tool

You can run the **runEventMonitor.sh** tool to view the events that are being fired in Prime Cable Provisioning. You can run this tool from the *BPR_HOME/rdu/internal/bin* directory.

The following table describes the types of events that you can view from the event monitor:

| Event | Sub-Event | Description |
|---|---|---|
| Batch | Completion | Displays when a batch submitted by a client application ends. Contains the batch status. |
| Class of service | New | Indicates when a class of service is added to the system. |
| Class of service | Deleted | Indicates when a class of service is deleted from the system. |

| Event | Sub-Event | Description |
|---|---|---|
| Configuration | Generated | Indicates when a configuration is generated. |
| Configuration | Uncommitted Generated | Indicates when a configuration that is temporarily stored at the DPE is generated. |
| Configuration | Rollback Uncommitted | Indicates that the uncommitted configuration should be discarded from the DPE. |
| CRS | Enabled | Indicates when configuration regeneration service is enabled. |
| CRS | Disabled | Indicates when configuration regeneration service is disabled. |
| CRS | Paused | Indicates when configuration regeneration service is paused. |
| CRS | Resumed | Indicates when configuration regeneration service is resumed. |
| CRS | Update | Indicates when a CRS request is updated. |
| CRS | Delete | Indicates when a CRS request is deleted. |
| CRS | Complete | Indicates when a CRS request has completed execution. |
| Device | Changed Class Of Service | Indicates when a device changes its Class of Service. |
| Device | Changed Domain Name | Indicates when a device changes its Domain Name. |
| Device | Changed Host Name | Indicates when a device changes its Host Name. |
| Device | Changed Device Properties | Indicates when a device changes its Device Properties. |
| Device | Changed IP Address | Indicates when a device's IP address changes. |
| Device | Deleted | Indicates when a device is deleted. |
| Device | Deleted Voice Service | Indicates when a voice service is deleted from a device. |
| Device | New Provisioned Device | Indicates when a device is added through the provisioning API. |
| Device | New Unprovisioned Device | Indicates when a device is added when booting on the network. |
| Device | New Voice Service | Indicates when a voice service is added to a device |
| Device | Roaming | Indicates when a device roams provisioning groups. |
| DHCP Criteria | New | Indicates when a DHCP criteria is added to the system. |
| DHCP Criteria | Deleted | Indicates when a DHCP criteria is deleted from the system. |

| Event | Sub-Event | Description |
|---|---|---|
| External File | Added | Indicates when a file is added to the system. |
| External File | Deleted | Indicates when a file is deleted from the system. |
| External File | Replaced | Indicates when a file is replaced in the system. |
| Messaging | Connection Up | Indicates when a connection on the local instance of the messaging system starts |
| Messaging | Connection Down | Indicates when a connection on the local instance of the messaging system stops. |
| Messaging | Queue Full | Indicates when the queue on the local instance of the messaging system is full and starts dropping messages. |
| Provisioning Group | Changed | Indicates when the provisioning group is changed. |
| Server Properties | Common Properties | Indicates when common properties that effect the RDU or DPE change. |
| System Configuration | Server Defaults Changed | Indicates when properties are changed on an user, RDU, or DPE. |
| System Configuration | System Configuration Changed | Indicates when the system configuration is changed. |
| System Configuration | System Defaults Changed | Indicates when defaults are changed. |

### Syntax Description

To run the event monitor, enter:

```
# /opt/CSCObac/rdu/internal/bin/runEventMonitor.sh [options]
```

Options are used to specify the RDU connection parameters and amount of output. You have the following options:

- **-noverbose**—Forces the event monitor to display only the types of events being fired, not their contents.
- **-host** *host*—Specifies the host where the RDU is located. Default is the localhost.
- **-username** *username*—Specifies username for RDU host.
- **-password** *password*— Specifies password of the RDU host.
- **-port** *port*—Specifies the port on which the RDU is listening. Default is 49187.
- **-secure**—Sets secure mode of communication with RDU.
- **-stopOnDisconnect**—Stops event monitoring process on disconnecting from the RDU.
- **-help**—Displays help for the tool.

### Sample Event Monitor Output

```
If need help, please restart command with '?' parameter.
Verbose mode: true
RDU host: localhost
RDU port: 49187
Connecting to RDU...ok
Listening for events...
ExternalFileEvent added filename=gold.cm
```

```
  rev=1014671115124(Mon Feb 25 16:05:15 EST 2002)
  source=BPR Provisioning API:BPR Regional Distribution Unit:AddExternalFile command
DeviceEvent newProvDevice ID=1,6,01:02:03:04:05:06
  rev=1014671179380(Mon Feb 25 16:06:19 EST 2002)
  source=BPR Provisioning API:BPR Regional Distribution Unit:AddIPDevice command IP=null
  FQDN=null group=null
```

# Using rdu.properties

**Caution** Do not modify the *rdu.properties* without consulting the Cisco support. Changes to this file might have an adverse impact on the RDU.

The *rdu.properties* file contains a variety of controls that specify the behavior of the RDU. You can open this file using any text editor, and change its content to perform the functions that you want.

You can configure the RDU by using the options available in the *rdu.properties* file. These options are controlled by Prime Cable Provisioning settings or defined in the *rdu.properties* file in the *BPR_HOME/rdu/conf/* directory. The default configuration parameters are:

- /server/port—Specifies the listening port of the RDU in nonsecured mode. The default port number is 49187.
- /server/secure/port—Specifies the listening port of the RDU in secure mode using SSL. The default port number is 49188.
- /server/rdu/secure/enabled—Specifies that the communication between RDU and other Prime Cable Provisioning components is secure.
- /server/rdu/unsecure/enabled—Specifies that the communication between RDU and other Prime Cable Provisioning components is unsecure.
- /secure/keystore/password—Specifies the keystore password for the keystore file. This password must be between 6 and 30 characters.
- /secure/keystore/file—Specifies the location of the keystore file.
- /secure/rdu/certificateKeyPassword—Specifies the password used to encrypt the certificate keys added in the keystore.
- /rdu/sharedSecret—Specifies the password used to encrypt the communication between Prime Cable Provisioning components and the RDU.
- /auth/user/session/limit/enabled=true - Specifies that the User session is Enabled. User session limit is disabled by default and same has to be enabled.

**Note** When you manually change properties in the rdu.properties file, remember to restart the RDU. RDU restart required for property changes to take effect. Use the `/etc/init.d/bprAgent restart rdu` command.

Sample rdu.properties File

```
cat /opt/CSCObac/rdu/conf/rdu.properties
```

```
/server/port=49187
/server/secure/port=49188
/server/rdu/secure/enabled=true
/server/rdu/unsecure/enabled=true
/secure/keystore/password=f2c2060fdbca0e60ae1864adb73155b9
/secure/keystore/file=/opt/CSCObac/lib/security/.keystore
/secure/rdu/certificateKeyPassword=b46411a3f24f08cd090bddd6e55d8de3
/rdu/sharedSecret=fgL7egT9zcYHs
```

# Using adminui.properties

Before you use the Admin UI, examine the *adminui.properties* file. This file contains a variety of controls that specify the behavior of the interface.

You can open this file using any text editor, and change its content to perform the functions that you want. After you save the changes, restart the Admin UI so that the changes take effect.

To start the Admin UI, enter:

```
# /etc/init.d/bprAgent start adminui
```

To stop the Admin UI, enter:

```
# /etc/init.d/bprAgent stop adminui
```

To restart the Admin UI, enter:

```
# /etc/init.d/bprAgent restart adminui
```

You can configure the Admin UI by using the options available in the *adminui.properties* file. These options are controlled by Prime Cable Provisioning settings or defined in the *adminui.properties* file in the *BPR_HOME/rdu/conf* directory. The configuration parameters are:

- /adminui/port—Specifies the listening port of the RDU. The default port number is 49187.
- /adminui/fqdn—Specifies the fully qualified domain name of the host on which the RDU is running. The default value is the FQDN of the host; for example, bac_test.EXAMPLE.COM.
- /adminui/maxReturned—Specifies the maximum number of search results. You can set this value to a maximum of 5000. The default value is 1000.
- /adminui/maxDetailsReturned —Specifies the maximum number of search results when search for detailed information is requested. You can set this value to a maximum of 1000 which is also the default value.

**Note** If the memory of the deployed server is having a smaller heap size, then the maxReturned and maxDetailsReturned will become half of its values. For example, if the value of maxReturned is set to 5000, it will retrieve only 2500.

- /adminui/pageSize—Specifies the number of search results displayed per page. You can set this number at 25, 50, or 75. The default value is 25.
- /adminui/refresh—Specifies if the refresh function is enabled or disabled. This option is, by default, disabled.

- /adminui/extensions—Specifies if the use of extensions in Prime Cable Provisioning is enabled or disabled. You use extensions to augment Prime Cable Provisioning behavior or add support for new device technologies. The use of extensions is, by default, enabled.
- /adminui/maxFileSize—Specifies the maximum size of a file uploaded to Prime Cable Provisioning. The default file size is 20 MB.
- /adminui/refreshRate—Specifies the duration (in seconds) after which a screen is refreshed. The default value is 90 seconds. Before setting a value for this option, ensure that the /adminui/refresh option is enabled.
- /adminui/file/extensions—Specifies the extensions of the files that the Admin UI supports. The supported extensions are by default *.bin*, *.cm*, and *.jar*.
- /adminui/timeout—Specifies the length of time after which an idle session times out. The default period is set as 10 minutes. In case of any value lesser than 10 minutes, the idle session time out still happens after 10 minutes.
- /adminui/noOfLines—Specifies the last number of lines from *rdu.log* or *dpe.log* that appear on the Admin UI. The default number of lines that appear is 250.
- /adminui/redirectToHttps—Specifies whether the Admin UI should be in HTTPS mode or not. The default is true.
- /adminui/enableDomainAdministration—Specifies whether Security Domain(RBAC) can be assigned to various entities. If set to true, the Instance Level Authorization check box is shown in the RDU Defaults page. The default value is false.

**Sample adminui.properties File**

```
/adminui/port=49187
/adminui/fqdn=doc.example.com
/adminui/maxReturned=5000
/adminui/pageSize=25
/adminui/refresh=disabled
/adminui/extensions=enabled
/adminui/maxFileSize=20000000
/adminui/refreshRate=90
/adminui/file/extensions=.bin,.cm,.jar
/adminui/timeout=10
/adminui/noOfLines=250
/adminui/redirectToHttps=false
```

**Note** By default, Prime Cable Provisioning redirects all HTTP communications over HTTPS. If you want to bypass the HTTPS redirection, set the property adminui/redirectToHttps to false in the admin.properties file.

# Using verifydb.sh Tool

This tool verifies the integrity of the database. It is a resource-intensive operation and should be performed on the RDU database when RDU server is down or on the backup snapshot. Verification of large database can take an extended length of time, to decrease the amount of time use a RAM disk or set the heap size to a higher value, for example, -Xms1024M -Xmx2048M.

The **verifyDb.sh** tool resides in the *$BPR_HOME/rdu/internal/db/bin/* directory. Invoking the script without any parameters verifies the active RDU database. In this case, the RDU server must be down for **verifyDb.sh** tool to operate.

To run this command:

**Step 1** Change directory to *BPR_HOME/rdu/internal/db/bin/*.

**Step 2** Run the verifyDb.sh command using this syntax:

**verifyDb.sh** *options*

where *options* are:

- **-dbdir**—Specifies the location of the database backup that is to be verified.
- **-dblogdir**—Specifies the location of the database logs that are to be verified.
- **-logdir**—Specifies the location of the logs that are to be verified.
- **-help**—Displays this help message. The **-help** option must be used exclusively.
- **-cachesize**—Specifies the size of the memory cache in MB.
- **-physical**—Verifies consistency of low level DB structures.
- **-logical**—Verifies logical consistency of data.

  The following are the suboptions of –logical option. These options can be used alone or in combination to narrow down the scope of the **-logical** consistency checks.

  - **-attrindexes**—Verifies attribute indexes.
  - **-objects**—Verifies objects and relationships.
  - **-relindexes**—Verifies relationship indexes.
  - **-relayagent**—Verify relay agent relationship.
  - **-properties**—Verifies object properties map.
  - **-cosFileProperty**—Verifies COS -File relationship issues.

**Example:**

**# $BPR_HOME/rdu/internal/db/bin/verifydb.sh -dbdir /disk1/backup**

where /disk1/backup is the path of the backup snapshot of the RDU database.

**Note** In case of any error while verifying the database, contact Cisco support.

# Using passwordEncryption.sh

The password encryption tool, passwordEncryption.sh allows you to enable password encryption using SHA1. This tool is available under BPR_HOME/rdu/bin. By default, SHA1 encryption is enabled for fresh installation

of Prime Cable Provisioning but disabled if you are upgrading from an earlier version. If you wish to enable encryption post upgrade, execute the command:

```
./passwordEncryption.sh -enable
```

Once you enable encryption, Prime Cable Provisioning will not be able to support the 4.0 and 4.0.x API clients.

To check if the SHA1 encryption is enabled or not, execute the command:

```
./passwordEncryption.sh -status
```

# Using changeSSLProperties.sh

You can use the **changeSSLProperties.sh** tool, which is found in the *BPR_HOME/bin* directory, to change key SSL configuration properties.

The following table lists the various options that you can use to change the SSL configuration.

***Table 3: changeSSLProperties.sh Options***

| Option | Description | Option Parameters |
|---|---|---|
| ./changeSSLProperties.sh **-ssl** | Use -ssl to enable or disable SSL or secure connection on RDU, API client, Admin UI or PWS. In case of Admin UI and PWS, this enables or disables the HTTPS mode of communication. | [rdu\|api\|adminui\|pws] [enable/disable]<br>For example:<br>`./changeSSLProperties.sh -ssl rdu enable` |
| ./changeSSLProperties.sh -nssl | Use -nssl to enable or disable non-secure connection with RDU, API client, Admin UI or PWS. In case of Admin UI and PWS, this enables or disables the HTTP mode of communication. | [rdu\|api\| adminui\|pws] [enable/disable]<br>For example:<br>`./changeSSLProperties.sh -nssl rdu disable` |
| ./changeSSLProperties.sh -secret | Use -secret to change the secret key for RDU, DPE and PWS, | [secret]<br>For example:<br>`./changeSSLProperties.sh -secret changeme` |

| Option | Description | Option Parameters |
|---|---|---|
| ./changeSSLProperties.sh -csp | Use -csp to change the default non-secure port number that RDU, Admin UI, API client or PWS listen on. By default, RDU listens on 49188.<br>In case of an API client, the command lists all the secure RDU hosts and you can change the port number of any of those RDU hosts using the tool. | [rdu\|api\|adminui\|pws]<br>For example:<br>`./changeSSLProperties.sh -csp rdu` |
| ./changeSSLProperties.sh -cnsp | Use -cnsp to change the default non-secure port number that RDU, Admin UI, API client or PWS listen on. By default, RDU listens on 49187.<br>In case of an API client, the command lists all the secure RDU hosts and you can change the port number of any of those RDU hosts using the tool. | [rdu\|api\|adminui\|pws]<br>For example:<br>`./changeSSLProperties.sh -cnsp rdu` |
| ./changeSSLProperties.sh -list | Use -list to list the secure or non-secure hosts. Use argument s to list the secure hosts and ns for non-secure hosts. | [s\|ns]<br>For example:<br>`./changeSSLProperties.sh -list n` |
| ./changeSSLProperties.sh -ckl | Use -ckl to changes the default keystore location. Respective property files get updated with this new keystore location.<br>By default, the keystore is stored in BPR_HOME/lib/security folder. | [new location]<br>For example:<br>`./changeSSLProperties.sh -ckl /opt/CSCObac/lib/security/.keystore` |
| ./changeSSLProperties.sh -ckp | Use -ckp to change the keystore password. You will be prompted to enter the old and new passwords. For security reasons all passwords will be prompted. | [new location]<br>For example:<br>`./changeSSLProperties.sh -ckp` |

| Option | Description | Option Parameters |
|---|---|---|
| ./changeSSLProperties.sh -utp | Use -utp to update the truststore password in case you have changed the default truststore (cacerts) password. This option updates only the related property files and does not change cacerts password. Since cacerts can contain other trusted entries/certificate chains, there is no option to change the trust store passwords. However you can change the truststore (cacerts) password using java keytool command, if you wish so. | For example:<br>`./changeSSLProperties.sh -utp` |
| ./changeSSLProperties.sh -cpkp | Use -cpkp to change the password used to store the RDU, Admin UI and PWS keys. You will be prompted for old and new passwords. For security reasons all passwords will be prompted. | [rdu\|adminui\|pws]<br>For example:<br>`./changeSSLProperties.sh -cpkp` |
| ./changeSSLProperties.sh -gk | Use -gk to generates a key pair, a public key and an associated private key.<br>The new created RDU key pair is stored in the .keystore file under BPR_HOME/lib/security. The following values would be set by default (keylength 1024, validity 2 years, keyalg RSA, alias rducert, storetype JCEKS).<br>You will be prompted for both keystore and key passwords. | For example:<br>`./changeSSLProperties.sh -gk` |
| ./changeSSLProperties.sh -exp | Use -exp to self-sign and export the certificate.<br>This option locates you keystore file, self-signs the RDU certificate and exports rootCA.crt and rootCA.pem files to the BPR_HOME/lib/security folder. | For example:<br>`./changeSSLProperties.sh -exp` |

| Option | Description | Option Parameters |
|---|---|---|
| ./changeSSLProperties.sh -imp | Use -imp to import a certificate to the cacerts trust store so that a chain of trust can be established between the certificate and RDU. If a chain of trust cannot be established, an error message appears.<br>In case of CNR-EP you should copy the rootCA.pem file to the machine where CNR-EP is installed. The files must be copied under the BPR_HOME/bin/security folder. | [location form where to import] [alias]<br>For example:<br>`./changeSSLProperties.sh -imp` |
| ./changeSSLProperties.sh -help | Use -help to view the help tips. | For example:<br>`./changeSSLProperties.sh -help` |

# Using ws-cli.sh

You can use the **ws-cli.sh** tool, which is found in the *BPR_HOME/pws/bin* for SOAP and *BPR_HOME/restpws/bin* for RESTful directory, to carry out some of the PWS configuration functions.

The following table lists the various options that are part of the ws-cli tool.

***Table 4: WS CLI Tools***

| Option | Description | Option Parameters |
|---|---|---|
| ./ws-cli.sh -ardu,--addrdu <host> <port> <username> <password> | Use this option to add a new RDU account. You could either use -ardu or --addrdu.<br>Repeat the same command to add multiple RDUs. | For example:<br>`#./ws-cli.sh -ardu test1-host 49187 admin changeme` |
| ./ws-cli.sh -rrdu,--removerdu <host> | Use this option to delete an existing RDU account.<br>You could either use -rrdu, or --removerdu. | For example:<br>`./ws-cli.sh -rrdu bac-test-lnx` |
| ./ws-cli.sh -srduc <host> <port> <username> <password> | Use this option to update RDU username and password by providing host and port number.<br>You could either use -srduc, or --setrducredentials. | For example:<br>`#./ws-cli.sh -srductest1 -host 49187 admin changeme` |
| #./ws-cli.sh -lrdu <host> | Use this option to list the RDU commands.<br>You could either use -lrdu, or --listrdu. | For example:<br>`#./ws-cli.sh -lrdu test1-host` |

| Option | Description | Option Parameters |
|---|---|---|
| ./ws-cli.sh -ll,--listlog <logger> | Use this option to list all the loggers and their current severity levels.<br>You could either use -ll, or --listlog. | For example:<br>`./ws-cli.sh -ll` |
| ./ws-cli.sh -lp,--listproperty <property> | Use this option to list all properties and their values.<br>You could either use -lp, or --listproperty. | For example:<br>`./ws-cli.sh -lp` |
| ./ws-cli.sh -rm,--removeproperty <property> | Use this option to remove the specified property -rm,--removeroperty. Triggers running app to reload the properties. | For example:<br>`./ws-cli.sh -removeproperty /cache/timeout` |
| ./ws-cli.sh -rp,--reloadproperty <property> | Use this option to remove the specified property **-rp,--reloadproperty**. Triggers running app to reload the properties. | For example:<br>`./ws-cli.sh -removeproperty /cache/timeout` |
| ./ws-cli.sh -rpd,--reloadpropertydef | Clears and reloads all the RDU property def caches. | For example:<br>`./ws-cli.sh -rpd` |
| ./ws-cli.sh -sap,--saveproperty | Saves the modifications. | For example:<br>`./ws-cli.sh -sap` |
| ./ws-cli.sh -sl,--setlog <logger=value> | Updates the logger level to either error, warn, info, or debug. | For example:<br>`./ws-cli.sh -sl general=DEBUG` |
| ./ws-cli.sh -sp,--setproperty <property=value> | Adds a new property or updates an existing property. | For example:<br>`./ws-cli.sh -sp /cache/timeout=100` |

# Scripts to Manage and Troubleshoot RDU Redundancy

Following are scripts that you can run to configure properties of HA resources as well as troubleshoot RDU redundancy. These scripts are available only when RDU is installed in redundancy mode. All these scripts are located under BPR_HOME/agent/HA/bin.

The following table lists the scripts that you can use to configure, monitor, and troubleshoot RDU redundancy.

*Table 5: RDU Redundancy Scripts - Configuration*

| Option | Description | Option Parameters |
|---|---|---|
| BPR_HOME/agent/HA/bin/ manage_ha_resources.sh <stop\|start\|restart> <res_IPaddr2_1\|res_Filesystem_1\|res_Filesystem_2\|res_Filesystem_3\|res_bprAgent_1> | This script is used to stop, start, and restart failed CRM resources. Where,<br>• res_IPaddr2_1: Virtual IP resource<br>• res_Filesystem_1: This file system refers to BPR_HOME<br>• res_Filesystem_2: This file system refers to BPR_DATA<br>• res_Filesystem_3: This file system refers to BPR_DBLOG<br>• res_bprAgent_1: This is the resource of bprAgent which the watch dog process of RDU | For example:<br>`BPR_HOME/agent/HA/bin/ manage_ha_resources.sh stop res_IPaddr2_1` |
| **Local HA:** BPR_HOME/agent/HA/bin/ configure_vip.sh <ipaddress> <ipaddress value> <CIDR> <CIDR value><br>**GEO HA:** BPR_HOME/agent/HA/bin/configure_vip.sh <ipaddress> <ipaddress value> | This script configures the VIP address of redundancy setup. It takes two inputs, first the new VIP address and the second is CIDR value which should be in the range >0 and <=30. | For example:<br>`BPR_HOME/agent/HA/bin/ configure_vip.sh ipaddress 10.105.202.46` |
| BPR_HOME/agent/HA/bin/ configure_failover_threshold.sh <timeout\|threshold> <value in mins> | This script configures failover timeout value and the migration threshold value of redundancy setup. You must specify the time in minutes. | For example:<br>`BPR_HOME/agent/HA/bin/ configure_failover_threshold .sh timeout 60`<br>`BPR_HOME/agent/HA/bin/ configure_failover_threshold .sh threshold 6` |
| BPR_HOME/agent/HA/bin/ set_prefered_ha_location.sh <primary\|secondary\|none> | This script is used to configure the preferred location either as Primary node or secondary node for automatic failback. To disable automatic failback, set this to none. | For example:<br>`BPR_HOME/agent/HA/ bin/set_prefered_ha_location .sh primary`<br>`BPR_HOME/agent/HA/ bin/set_prefered_ha_location .sh none` |
| BPR_HOME/agent/HA/bin/ping_resource_ha.sh <create\|update\|delete> | The script configures the cluster to create, update and delete ping resources to move resources when external connectivity is lost. It prompts for list of IP addresses separated by space to monitor ping resources. | For example:<br>`BPR_HOME/agent/HA/ bin/ping_resource_ha.sh create`<br>`BPR_HOME/agent/HA/ bin/ping_resource_ha.sh update`<br>`BPR_HOME/agent/HA/ bin/ping_resource_ha.sh delete` |

*Table 6: RDU Redundancy Scripts - Monitoring*

| Option | Description | Option Parameters |
|---|---|---|
| BPR_HOME/agent/HA/bin/ monitor_ha_cluster.sh [timing-details] | Displays the summary of the CRM resource status. timing-details is an optional parameter, if it is provided, it displays complete summary about the CRM resources with timing details. | For example:<br>`BPR_HOME/agent/HA/bin/ monitor_ha_cluster.sh timing-details` |
| BPR_HOME/agent/HA/bin/ monitor_fs_sync_status.sh | This script prints the current status of file system syncup between primary and secondary node. | For example:<br>`BPR_HOME/agent/HA/bin/ monitor_fs_sync_status.sh` |

*Table 7: RDU Redundancy Scripts - Troubleshooting*

| Option | Description | Option Parameters |
|---|---|---|
| BPR_HOME/agent/HA/bin/ cleanup_ha_resources.sh <[illegible]> | This script cleans up the CRM resources. Specify the resource name that you want to clean up as the parameter. To clean up all resources in one go, use the option all. Where the CRM resources are,<br>• res_IPaddr2_1: Virtual IP resource<br>• res_Filesystem_1: This file system refers to BPR_HOME<br>• res_Filesystem_2: This file system refers to BPR_DATA<br>• res_Filesystem_3: This file system refers to BPR_DBLOG<br>• res_bprAgent_1: This is the resource of bprAgent which the watch dog process of RDU | For example:<br>`BPR_HOME/agent/HA/bin/ manage_ha_resources.sh res_Filesystem_1` |
| BPR_HOME/agent/HA/bin/ standby_ha_switch.sh <primary\|secondary> <on\|off> | This script troubleshoots or tests the setup manually. You can make primary or secondary node as standby using this script. Use the option on to make a node standby and off to get it online. | For example:<br>`BPR_HOME/agent/HA/bin/ standby_ha_switch.sh secondary off` |
| BPR_HOME/agent/HA/bin/ resolve_sb_victim.sh <bprHome\|bprData\|bprLog\|all> | Run this script if you want to discard the updates being made to a server during a split-brain scenario. You must run the script from the server where you want to discard the updates. You need to mention which data should be discarded and give the option as either bprHome, bprData, or bprLog. To discard all the updates, use the option all. | For example:<br>`BPR_HOME/agent/HA/bin/ resolve_sb_victim.sh bprData` |

| Option | Description | Option Parameters |
|---|---|---|
| BPR_HOME/agent/HA/bin/ resolve_sb_survivor.sh <bprHome\|bprData\|bprLog\|all> | This script should be run from the other server from where user wants to get update from, when there is a split-brain scenario. User has to mention option as either bprHome, bprData, or bprLog. | For example:<br>`BPR_HOME/agent/HA/bin/ resolve_sb_survivor.sh bprLog` |

# Using deviceReader Tool

The **deviceReader** tool (**deviceReader.sh**) is used to extract the device details from a RDU database. It reads the device objects along with the associated resources like, CoS, DHCP criteria and presents the device information in a default file. This tool can be used against the RDU database when the RDU server is down or against a backup snapshot of the database by specifying the location with *-dbdir* and *-dblogdir* options.

This **deviceReader** tool provides options to save the device details in a file and it provides customization options to process the device details.

The **deviceReader.sh** tool is present in the *$BPR_HOME/rdu/internal/db/bin/* directory.

**Syntax Description**

**deviceReader.sh** *[-file outputfile] [-dbdir dir] [-dblogdir dir]*

- *-file*— Specifies the output file name and the path to save the device information. By default, the output file (deviceinfo.txt) is generated on the current working directory.
- *-dbdir*— Specifies the database directory path. By default, the RDU database location is used.
- *-dblogdir*— Specifies the database log directory path. By default, the directory specified with dbdir option or RDU database location is used.
- *-help*— Displays help for the tool.

**Example**

```
# ./deviceReader.sh -dbdir /opt/backup/rdu-backup-20170410-150614/ -file
/opt/result/devicedump.txt
```

Where,

— */opt/backup/rdu-backup-20170410-150614/* is the database directory

— */opt/result/devicedump.txt* is the output file and the path to save the file

### Output Device File Format

In the output device file, the device properties are stored seperated by '|':

```
MAC/DUID | OwnerID| Hostname | Domain| ProvGroupName | CoS | DHCPCriteria | docsis_version|
 device_serial_number| custom_properties
```

For example:

```
#  cat /opt/result/devicedump.txt
```

```
000000000211|testowner1|testhost1|testdomain1|chennai|null|null|1.0|000000000211|
/snmp/writeCommunityString|write123|test2|22|test1|11|/snmp/readCommunityString|read123
000000000212|testowner2|testhost2|testdomain2|chennai|null|null|1.0|000000000212|
/snmp/writeCommunityString|write1234|test2|2234|test1|1123|/snmp/readCommunityString|read1234
000000000290|null|null|null|chennai|null|null|null|null
000000000291|null|null|null|chennai|null|null|null|null
```

### Customizing Device Data Usage

By default, the **deviceReader** tool provides the device data in a file. This tool also allows you to customize the device data output as per your requirement, i.e., the data handling is customized to send the device properties to a remote server / another file in a required format.

#### Data Handler Customization

1. Write a custom DataPrinter implementaion in Java.

   Implement the interface, com.cisco.csrc.db.util.devicereader.DataPrinter.

```
// Source code of the interface
 package com.cisco.csrc.db.util.devicereader;

 import java.util.Map;

 /**
  * The interface to mandate the methods to be implemented by the custom printer
  * implementations
  */
 public interface DataPrinter
 {

     /**
      * This method will be invoked while reading each device object from the
      * database.
      * @param properties the device properties
      */
     public void print(Map<String, Object> properties);

     /**
      * This method will be called the the tool has completed reading all the
      * devices. This can be used to close any resources used by the custom
      * printer implementations
      */
     public void closeConnections();
 }


The print method
  public void print(Map<String, Object> properties)
   This method exposes the device properties in a Map.
   This method will be invoked when each IP device object is read from database by the
tool
   The keys for accessing the device device properties are available in
com.cisco.provisioning.cpe.constants.DeviceDetailsKeys.
   The keys are same as that of a getDetails() API result

 Connection handling
  public void closeConnections();
   This method will be invoked when the tool has completed.
   This can be used to close any resources opened by the custom data printer
implementation

 // A sample data printer implementation
```

```
package com.test;

import java.util.Map;

/**
 * No op printer used for testing
 */
public class MyCusomDataPrinter implements DataPrinter
{

    public void print(Map<String, Object> properties)
    {
        // process the device properties here.
        // Eg Write to console
        //System.out.println(" CoS "+properties);

    }

    public void closeConnections()
    {

        // Optional - handle (if any) connection house keeping here
    }
}
```

2. Implement the interface, com.cisco.csrc.db.util.devicereader.DataPrinter, and attach it to the tool's classpath.
3. The custom data reader can be configured in <BPR_HOME>/rdu/internal/db/bin/devicereader.conf. Configure the name of class file in this file, for example: *device=com.test.MyCusomDataPrinter*

# Using Live DB Compaction Tool

The Live DB Compaction tool (**configureDbCompaction .sh**) is used to compress the RDU database without stopping the RDU.

Prior to Prime Cable Provisioning 6.1, offline DB compaction was supported as explained in the following link:
***https://supportforums.cisco.com/t5/network-infrastructure-documents/db-compression-tech-note-pdf/ta-p/3149689***
.

For the offline compaction procedure to work, RDU server has to be shut down. Since this offline compaction necessitates a long downtime for the RDU server, the live compaction of RDU Berkeley DB which will avoid the downtime of the RDU server is supported in Prime Cable Provisioning 6.1.

The live DB compaction is disabled by default when the RDU server is started. The live DB compaction triggered using configureDbCompaction.sh results in increase in the fill factor of Berkeley DB (default fill factor for live compaction is 80%). *$BPR_HOME/rdu/internal/db/native/runTool.sh* can be used to check the fill factor of the database after the live compaction is run.

**Note** It is recommended to invoke the **runTool.sh** against a backup snapshot of the Database or when the RDU server is down.

Running of live compaction on a regular basis will free up pages in the database which will be reused by Berkeley DB to write new data. It does not reclaim any disk space but significantly lessens any further increase in disk space. Thus it avoids steep increase of database disk size. When live compaction is carried out on a

regular basis (for example, once a week) the total time taken will be less than a minute. Database checkpoint will be triggered soon after compaction to sync any uncommitted changes to the DB which may take a couple of minutes.

In order to reclaim disk space, another tool **runCompactDB.sh** under *$BPR_HOME/rdu/bin/internal/db/bin* is provided which can be used in offline mode. This tool is similar to the online DB compaction tool with the only difference that it supports an additional option *-reclaimspace*. When this tool is run with *-reclaimspace* option, it will recover disk space. The disk space reclaimed by this tool will vary depending upon the way the nonempty pages are allocated. Only pages at the end of a file can be returned to the file system in this tool. The compact algorithm makes a one-pass over the pages of the database, so nonempty pages at the end of the file will prevent free pages (that are placed on the free list) from being returned to the file system. That is the reason we recommend regular live DB compaction to avoid growing of DB disk size and this offline DB compaction can be used occasionally along with online DB compaction tool.

Even though the online DB compaction tool supports scheduling of live DB compaction at regular intervals, we recommend triggering one time execution of online DB Compaction after taking a Database Backup at periodic intervals (achievable using a cron job). Since data is paramount and database manipulation requires utmost care, this approach will allow the user to have a control over the DB and a valid backup to restore in case of any unforeseen failure during DB compaction.

The online or offline DB compaction tool is not a direct replacement to *DBdump/dbload* utility. If the user doesn't run online or offline DB compaction tool periodically then *DBdump/dbload* utility is best tool to reclaim disk space. In general, most of the data access should be serviced by the cache, so the file fragmentation or low fill factor should not have a noticeable performance impact.

The**configureDbCompaction .sh** tool is present in the *$BPR_HOME/rdu/bin/* directory. Once the compaction is run, the status of the compaction and the time taken will be available in *$BPR_DATA/rdu/db/history.log*.

**Note**

1. We recommend you to use the offline compaction once at the very beginning using (*db_dump* and *db_load BDB* utility) as mentioned in the above support forum link. This will reduce the DB disk size, so that the live compaction will make sure to stop the increase in disk space.
2. It is recommended to take a backup of the database each time before running the live compaction.

**Error Handling:**

If compaction is run when database backup or DB log deletion is in progress, it will throw an error and exit.

Since the compaction process will lock portions of the DB tables when it performs commit, you may see write batches to the DB failing at that time. If write batches higher than10/seconds is sent when compaction is in progress, you are likely to see *RDU_BUSY*.

**Syntax Description of Online Compaction Tool**

**configureDbCompaction.sh** *[-show] [-run option] [-interval value] [-fillfactor ff]*

- *-show*— Displays the current values of the compaction parameters.
- *-run*— This parameter specifies whether the compaction process has to run once or scheduled or disabled. Valid values are 1, 2, and 3. The default value is 3.

  1 (Once) - Trigger compaction once.

  2 (Schedule) - Schedule compaction at regular intervals.

  3 (Disable) - Disable compaction.

- -*interval*— Specifies the interval at which the compaction process has to be scheduled. The interval is 'day of week:hour of day' where day of week is any day from monday-sunday and hour of day is any value from 00-23. The default value is ‘sunday:00’.
- - *fillfactor*— This parameter is to specify the page fill factor. Valid values 1-100. The default value is 80%.
- -*help*— Displays help for the tool.

**Examples:**

1. **Run Compaction Once**

```
$BPR_HOME/rdu/bin/configureDbCompaction.sh -run 1
Please enter RDU username: admin
Please enter RDU password:

Live DB Compaction is enabled to run once.
```

**Note:** For status on the DB Compaction, check the **history.log** present in the *$BPR_DATA/rdu/db* directory.

2. **Schedule compaction**

```
$BPR_HOME/rdu/bin/configureDbCompaction.sh -run 2
Please enter RDU username: admin
Please enter RDU password:

Enter the compaction interval (sunday:0):
monday:02
Live DB Compaction is scheduled to run at regular intervals MONDAY at 02 hrs.
```

**Note:** For status on the DB Compaction, check the **history.log** present in the *$BPR_DATA/rdu/db* directory.

3. **Change compaction interval**

```
$BPR_HOME/rdu/bin/configureDbCompaction.sh -interval Tuesday:20
Please enter RDU username: admin
Please enter RDU password:

Live Compaction interval set to TUESDAY at 20 hrs.
```

4. **Disable compaction**

```
$BPR_HOME/rdu/bin/configureDbCompaction.sh -run 3
Please enter RDU username: admin
Please enter RDU password:

Live DB Compaction is disabled.
```

5. **Set page fill factor**

```
$BPR_HOME/rdu/bin/configureDbCompaction.sh -fillfactor 70
Please enter RDU username: admin
Please enter RDU password:

Fill Factor for Live Compaction set to 70
```

6. **Using the -help option**

```
$BPR_HOME/rdu/bin/configureDbCompaction.sh -help
This tool can be used to configure the DB Compaction parameters. The command line syntax
 for this tool is as follows:
configureDbCompaction.sh [-show] [-run option] [-interval value] [-fillfactor ff]
-show           Displays the current values of the compaction parameters
-run            An optional parameter to specify whether the Compaction process is to
be run once or scheduled or disabled. Valid values 1,2 and 3.
1 Once          Enter 1 to trigger compaction once.
2 Schedule      Enter 2 to schedule compaction at regular intervals.
3 Disable       Enter 3 to disable compaction.
-interval       An optional parameter to provide the interval in which the compaction
process is to be scheduled.
                The interval is value is'day of week:hour of day' where day of week is
any day from monday-sunday and hour of day is any value from 00-23.
                For example, to schedule compaction every sunday at 1am, it can be set
to 'sunday:01'.
-fillfactor     An optional parameter to provide the page fill factor. Valid values
1-100.
```

### Syntax Description of Offline Compaction Tool

**runCompactDB.sh** *[-cachesize mb] [-dbdir dir] [-dblogdir dir] [-fillfactor ff] [-reclaimspace]*

- *-cachesize*— An optional parameter that specifies cache size in MB. The default cache size is 10MB.
- *-dbdir*— An optional parameter with database directory path. RDU database location is used by default.
- *-dblogdir*— An optional parameter with database directory path. Directory specified with *-dbdir* option or RDU database location is used by default.
- *- fillfactor*— An optional parameter to provide the page fill factor.
- *-reclaimspace*— An optional parameter to enable disk space reclaimation when compaction is run.

### Example to Reclaim Disk Space:

```
$BPR_HOME//runCompactDB.sh -reclaimspace

-----------------------------------
Starting DB Compaction
-----------------------------------

Reclaim Disk Space option value: true

Running DB Compaction with a fill factor of 100%


Time Taken for DB Compaction: 286584
```

```
Running DB Compaction with a fill factor of 90%
Time Taken for DB Compaction: 164891


Disk Space Reclaimed after compaction in bytes: 24576

-----------------------------------
```

# DPE Event Publisher

DPE event publisher allows the user to view the events that are being fired in the Prime Cable Provisioning DPE. The publisher framework allows to customize the DPE events publishing as per your requirement. To publish the DPE events, it provides options to plug-in your own producer implementation and, the DPE events can be,

- published to any messaging system based on the producer implementation (By default, the DPE offers Kafka based producer implementation).

- published to any remote server.

- logged in to a file in a required format.

For information on custom producer implementation, see the below **Custom DPE Event Implementation** section.

The sample implementation file *SampleEventPublisherImpl.java* is present in *$BPR_HOME/dpe/samples/event/* directory.

**DPE Event Schema**

The publisher sends the events as per the below defined Avro schema:

```
{
 "namespace": "com.cisco.csrc.dpe.events.specific",
 "type": "record",
 "name": "DpeEvent",
 "fields": [
         {"name": "dpe_event_id", "type": "int"},
         {"name": "event_source", "type": "string"},
         {"name": "host_name", "type": "string"},
         {"name": "received_time", "type": "string"},
         {"name": "display_message_tag", "type": "string"},
         {"name": "display_message", "type": "string"},
         {"name": "event_data", "type": {"type": "map", "values": "string"}}
 ]
}
```

The *dpe_event_schema.avsc* schema file is present in the *$BPR_HOME/dpe/samples/event/* directory.

### Custom DPE Event Implementation

For custom DPE event implementation:

1. Write a custom producer event implementation in java to publish event to any custom messaging system by implementing the interface **com.cisco.csrc.dpe.events.ProduceEvent**.

   ```
   // Source code of the interface
   package com.cisco.csrc.dpe.events;
   import org.apache.commons.io.output.ByteArrayOutputStream;
   /**
   ```

```
* The interface to mandate the methods to be implemented by the custom produce event
* implementations
*/
public interface ProduceEvent extends DpeEventConstants
{
/**
* This method will be invoked if DPE event enable to send event.
* @param topic event type represents in DpeEventConstants
* @param event event data represents as avro standard ByteArrayOutputStream
*/
    public void eventPublisher(int topic, ByteArrayOutputStream event);
}
```

The **eventPublisher** method exposes the DPE object data in Avro Standard ByteArrayOutputStream for event publishing. This method will be invoked when DPE operations like File Operation, Configuration/Cache Operation, Log Operation, TFTP Operation, Device level request Operation, ToD request and SNMP reset Operation.

The keys for accessing the events are available in dpe_event_schema.avsc schema file which is present in the *$BPR_HOME/dpe/samples/event/* directory.

The topic is used to differentiate the event type

```
    /** DPERequestEvent */
    final public static int DPE_REQUEST_EVENT = 1;
    /** FileEvent */
    final public static int FILE_EVENT = 2;
    /** ConfigurationEvent */
    final public static int CONFIGURATION_EVENT = 3;
    /** TftpEvent */
    final public static int TFTP_EVENT = 4;
    /** LogEvent */
    final public static int LOG_EVENT = 5;
```

```
// A sample produce event implementation
package com.cisco.test.dpeevent;

import org.apache.commons.io.output.ByteArrayOutputStream;
import com.cisco.csrc.dpe.events.ProduceEvent;

public class MyCustomProducerEventImpl implements ProduceEvent
{

    @Override
    public void eventPublisher(int topic, ByteArrayOutputStream event)
    {
// process the event data here.
// Eg Write to console
//System.out.println(" Event Type " + topic " : "+ event);
// Or Implement your custom code for any messaging system
   }
 }
```

**Note** To compile the custom implementation class, we need to include bpr.jar, bac-common.jar, bacbase.jar and commons-io.jar which are present in the *$BPR_HOME/lib/* directory.

2. Compile the *MyCustomProducerEventImpl.java* and bundle as a jar file (for eg: *dpeevent.jar*). Add the bundled jar and its dependencies jar (i.e., any messaging system dependency jars) to the publisher`s classpath *$BPR_HOME/lib/*.

3. The custom event can be configured in *$BPR_HOME/dpe/internal/bin/dpeeventmonitor.conf*. You can also configure the name of the class file in this file, for example, **/dpe/producer/class= com.cisco.test.dpeevent.MyCustomProducerEventImpl**.
4. Add the path for custom jar and its dependent jar to BPR_CP variable in *$BPR_HOME/bpr_definitions.sh* before exporting BPR_CP

```
BPR_JAVA=$BPR_HOME/jre/bin/java
BPR_CP=$BPR_CP:$BPR_HOME/lib/dpeevent.jar:$BPR_HOME/lib/dependentJar1.jar:$BPR_HOME/lib/dependentJar2.jar:.
```

5. Restart the DPE.
6. Enable the DPE event monitor using CLI, see, **Event System Management Commands** chapter of Cisco Prime Cable Provisioning 6.1.2 DPE CLI Reference Guide.
7. Based on the enabled event types, DPE events are published to custom implementation messaging system.

### DPE Event Monitor CLI Commands

Prime Cable Provisioning generates the event messages from DPE server and publishes it using the custom messaging system. The event messages fired by the DPE server are based on the event types that are enabled in the settings. Using the CLI commands, the event monitor and the event type can be enabled/disabled.

#### Event type and Description

The following table describes the types of events that you can view from the DPE event monitor:

| Event ID | Event Type | Description |
|---|---|---|
| 1 | Request Event | Device request<br>• Sending no cached configuration for device in provisioning group to device.<br>• Sending configuration for device in provisioning group to device.<br>ToD request<br>• Received UDP time of day request from device.<br>• ToD Success/Failure.<br>SNMP reset<br>• Processing SNMP reset for device.<br>• Successfully send SNMP reset for device. |
| 2 | File Event | • Received file from RDU.<br>• Received updated file from RDU.<br>• Removed file from cache. |

| Event ID | Event Type | Description |
|---|---|---|
| 3 | Configuration Event | • Received configuration for device from RDU.<br>• Received updated configuration for device from RDU.<br>• Removed configuration for device from cache.<br>• Completed device attributes dumping process |
| 4 | TFTP Event | • Received a TFTP [read] request from device for file.<br>• Finished handling [read] request from device for file.<br>• TFTP exception. |
| 5 | Log Event | • Send the DPE log as events.<br>• Depend on the DPE log level it send the logs as events. |

1. Login the Telnet using the credentials.
2. Enable the DPE event monitor using CLI command: dpe event monitor.
3. To enable/disable the different event level, see, **Event System Management Commands** chapter of Cisco Prime Cable Provisioning 6.1.2 DPE CLI Reference Guide.

**Note** It is not necessary to restart the DPE service after the event enabling/disabling.

### Sample DPE Events

To use the sample DPE events (Softwares required are, kafka and scala):

1. Start the kafka and zookeeper servers with default configurations as shown below:

```
nohup ./zookeeper-server-start.sh ../config/zookeeper.properties > zoo.out &

nohup ./kafka-server-start.sh ../config/server.properties > kafka.out &
```

2. To verify the status of the servers:

```
ps -ef | grep zookeeper

ps -ef | grep kafka
```

3. Enable the DPE event properties (*monitor, file, log, config, request, tftp*) by using the telnet commands.
4. The sample DPE event will send the events to *localhost:2181* port with the topic *dpeevent*. To consume the published events:

```
./kafka-console-consumer.sh --zookeeper localhost:2181 --topic dpeevent --from-beginning
```

5. If the consumer is in a different server, to consume the published events:

```
./kafka-console-consumer.sh --zookeeper pcp-lnx-xx:2181 --topic dpeevent --from-beginning
```

### Sample Output

Once the integration is done. You can view the published event on your custom messaging system.

### Example sample output

```
{
  "dpe_event_id" : 1,
  "event_source" : "DPE_REQUEST_EVENT",
  "host_name" : "pcp-lnx-90",
  "received_time" : "2018-10-15 04:49:55,955 IST",
  "mnemonic_tag" : "0112",
  "display_message" : "Sending configuration for device [1,6,11:00:00:00:00:10] in
provisioning group [default] to [10.78.109.52:55946]. Time since request received [2 ms].
Rate [0.017/s over 1 min].",
  "event_data" : {
    "device_id" : "1,6,11:00:00:00:00:10",
    "provisioning_group" : "default",
    "event_message" : "Protocol Version=13, Type=CONFIGURATION_REQUEST, Transaction
ID=1533394698, Device ID=1,6,11:00:00:00:00:10, Prov Group=default",
    "inet_address" : "10.78.109.52:55946"
  }
}
```