Implementing Backup

Introduction

The new FreePBX backup now allows each module to handle their own data. To see modules that implement the new backup you can run the command: fwconsole bu --Implemented

> **Technical Debt**
>
> Please note in 15 the backup and restore methods are removed. Please remove these from the modules BMO class. They are unimplemented.

The Files

Implementation uses 2 classes. One class for each purpose.

- rawname/Backup.php
- rawname/Restore.php

Skeleton Files

You can copy these from the backup directory then change the module name as required.

```
[root@freepbx blacklist]# pwd
/usr/src/freepbx/blacklist
[root@freepbx blacklist]# cp ../backup/examples/Backup.php.template Backup.
php
[root@freepbx blacklist]# cp ../backup/examples/Restore.php.template
Restore.php
[root@freepbx blacklist]# sed -i 's/__MODULENAME__/Blacklist/g' Backup.php
[root@freepbx blacklist]# sed -i 's/__MODULENAME__/Blacklist/g' Restore.php
```

The Code

Backup.php

Backup.php

```
<?php
namespace FreePBX\modules\Recordings;
use FreePBX\modules\Backup as Base;
class Backup Extends Base\BackupBase{
  public function runBackup($id,$transaction){
    //$this->FreePBX
  }
}
```

Restore.php

Restore.php

```
<?php
namespace FreePBX\modules\Recordings;
use FreePBX\modules\Backup as Base;
class Restore Extends Base\RestoreBase{
  public function runRestore(){
    //$this->FreePBX
    //$this->tmpdir
  }
}
```

Namespaces

- The namespace should be FreePBX\modules\Rawname
- This module uses FreePBX\modules\Backup as Base

```
<?php
namespace FreePBX\modules\Recordings;
use FreePBX\modules\Backup as Base;
class Restore Extends Base\RestoreBase{
  ..code goes here..
}
```

```
<?php
namespace FreePBX\modules\Recordings;
use FreePBX\modules\Backup as Base;
class Backup Extends Base\BackupBase{
  ..code goes here..
}
```

Logging

Backup.php and Restore.php both implement a backend logger which you can utilize as such:

```
$this->log($msg,$level);
```

If $level is not set then the default level will be used which is INFO, you can also use the following levels:

- DEBUG
- NOTICE
- WARNING
- ERROR
- CRITICAL
- ALERT
- EMERGENCY
- INFO

Backup.php

Backup Module Variables

By default the class inherits:

- **$thisFreePBX:** The FreePBX BMO Object
- **$thisbackupObj:** The Backup BMO Object

Backup Module Methods

Backup configuration methods

- **addConfig:** Used to add a single configuration to the configs array

```
/**
 * Add single config by key to the configs array
 *
 * @param string $key The key in the array to add the data
 * @param mixed $settings Data to insert
 * @return void
 */
$this->addConfig($key, $settings)
```

- **addConfigs:** Used to add configuration as an array

```
/**
 * Add Configuration item this should be an array. The contents will
depend on your module.
 *
 * @param array $settings Multidimensional Array of settings to add
 * @return void
 */
$this->addConfigs(array $configs);
```

- **addDependency:** Add a single dependency for this module

```
/**
 * Add a single dependency for this module
 *
 * These are dependencies for restore. During the module reset
dependencies should be resolved. This can be called multiple times.
 *
 * @param string $dependency
 * @return void
 */
$this->addDependency($string);
```

- **addDirectories:** Add multiple Directories

```
/**
 * Add multiple Directories
 *
 * Add Directories you use IF you are backing up files item this
should be an array. The contents will depend on your module.
 *
 * @param array $list
 * @return void
 */
$this->addDirectories($array);
```

- **addFile:** Add Single file to Files List

```
/**
 * Add Single file to Files List
 *
 * @param string $filename The file name: File.ext
 * @param string $path /path/to/file (no trailing slash)
 * @param string $base If you are using a path variable such as
VARLIBDIR
 * @param string $type A file type identifier. This is module
dependent and can be any string.
 * @return void
 */
$this->addFile($filename,$path,$base,$type);
```

- **addSplFile:** Similar to addFile except you can use splfile info to generate what you need

```
$file = new \SplFileInfo('example.php');
/**
 * Utilizes SplFileInfo to add a file
 *
 * @param \SplFileInfo $file
 * @return void
 */
$this->addSplFile($file);
```

Backup Overridable methods

- **runBackup:** This is where your main logic goes. This method is called when a backup is requested

```
/**
 * Run Backup Method used by other modules
 *
 * @param [type] $id
 * @param [type] $transaction
 *
 * @return void
 */
public function runBackup($id,$transaction);
```

Backup Helper Methods

- **dumpAll:** Dumps all relevant settings into a multidimensional array

```
/**
 * Dump all relevant settings into an array
 *
 * Advanced Settings
 * Feature Codes
 * Module Tables
 * Key Value Store
 *
 * @return array
 */
$this->dumpAll()
```

- **dumpAdvancedSettings:** Dumps Module's Advanced Settings into an array

```
/**
 * Dump all known advanced settings
 *
 * @return array
 */
$this->dumpAdvancedSettings()
```

- **dumpFeatureCodes:** Dumps module's feature codes into an array

```
/**
 * Dump all known feature codes
 *
 * @return array
 */
$this->dumpFeatureCodes()
```

- **dumpTables:** Dumps module's database tables into an array

```
/**
 * Dump all known databases from said module
 *
 * @return array
 */
$this->dumpTables()
```

- **dumpKVStore:** Dumps module's Key Value store entries into an array

```
/**
 * Dump KVStore to a multidimensional array
 *
 * @return array
 */
$this->dumpKVStore()
```

Restore.php

Restore Module Variables

By default you will have access to:

- **$thisFreePBX:** The FreePBX BMO Object
- **$thisbackupObj:** The Backup BMO Object
- **$thistmpdir:** Where files are extracted to

Restore Module Methods

Restore configuration methods

- **getConfigs:** Get the configurations as a multidimensional array for said module

```
/**
 * Get Configurations
 *
 * @param array $options
 * @return array
 */
$this->getConfigs();
```

- **getFiles:** Gets the file list for said module

```
/**
 * Get Files
 *
 * @param array $options
 * @return array
 */
$this->getFiles();
```

- **getDirectories:** Gets the list of directories

```
/**
 * Get Directories Alias
 *
 * @param array $options
 * @return array
 */
$this->getDirectories();
```

- **getDependencies:** Gets the list of dependencies for said module

```
/**
 * Get Module Dependencies
 *
 * @param array $options
 * @return array
 */
$this->getDependencies();
```

- **getVersion:** Get's the version of the module being restored

```
/**
 * Get Module Version
 *
 * @return string
 */
$this->getVersion();
```

Restore Overridable methods

- **runRestore:** This is where your main logic goes. This method is called when a restore is requested

```
/**
 * Run Restore Method used by other modules
 *
 * @return void
 */
public function runRestore();
```

- **reset:** Typically during restore modules are "reset" by uninstalling and reinstalling the module by executing the reset() method. You can override this in your own class to prevent the default action or perform other actions

```
/**
 * The reset method is run right before the restore method is executed
 *
 * You can override this and add custom code here
 *
 * Example. Framework skips this method by overwriting it and doing
nothing
 *
 * @return void
 */
public function reset() {
```

Restore Helper Methods

- **importAll:** This function expects a multidimensional array in the format returned from the backup method dumpAll()

```
/**
 * Import all from a multidimensional array
 *
 * @param array $data
 * @return void
 */
$this->importAll([
  'settings' => [],
  'features' => [],
  'tables' => [],
  'kvstore' => []
]);
```

- **importAdvancedSettings:** This function expects an array in the format returned from the backup method dumpAdvancedSettings()

```
/**
 * Import advanced settings from a multidimensional array
 *
 * @param array $settings
 * @return void
 */
$this->importAdvancedSettings([
  'KEYWORD' => 'value'
]);
```

- **importFeatureCodes:** This function expects an array in the format returned from the backup method dumpFeatureCodes()

```
/**
 * Import Feature codes from a multidimensional array
 *
 * @param array $codes
 * @return void
 */
$this->importFeatureCodes([
  'keyword' => [
    'customcode' => '',
    'enabled' => true
  ]
]);
```

- **importAstDB:** Expects a multidimensional array in the format of key => children => child => value

```
/**
 * Import Asterisk Database from a multidimensional array
 *
 * @param array $families
 * @return void
 */
$this->importAstDB([
  'family' => [
    'child' => 'value'
  ]
]);
```

- **importTables:** This function expects an array in the format returned from the backup method dumpTables()

```
/**
 * Import tables from a multidimensional array
 *
 * @param array $data
 * @return void
 */
$this->importTables([
  'table_from_database' => [
    [
      'col1' => 'val1'
    ]
  ]
]);
```

- **importKVStore:** This function expects an array in the format returned from the backup method dumpKVStore()

```
/**
 * Import KVStore from a multidimensional array
 *
 * @param array $data
 * @return void
 */
$this->importKVStore([
  'id' => [
    'key' => 'val'
  ]
]);
```

- **addDataToTableFromArray:** This is used by the importTables() method above

```
/**
 * Dynamically add data from an array to a table
 *
 * This uses doctrine to see the column names and types to match them
up
 * and quote them correctly, if any columns are missing then a
warning is displayed
 *
 * @param string $table The table name
 * @param array $data The data to import
 * @param boolean $delete If set to true then delete everything from
the table before inserting
 * @return void
 */
$this->addDataToTableFromArray($table, $data, $delete);
```

Legacy Restore

Restore legacy methods are added to the same Restore.php class file as the 15+ restore. They do not conflict.

Legacy Restore Configuration Methods

- **getVersion:** Get's the version of the module being restored

```
/**
 * Get Module Version
 *
 * @return string
 */
$this->getVersion();
```

Legacy Restore Overridable methods

- **processLegacy:** Process Legacy Method used by other modules

```
/**
 * Process Legacy Method used by other modules
 *
 * @param PDO $pdo The pdo connection for the temporary database
 * @param array $data An array with 'manifest', 'astdb', 'settings',
'features' as arrays
 * @param array $tables is a list of tables we determined belong to
the module
 * @param array $unknownTables is an array of tables we don't have an
owner for
 * @return void
 */
public function processLegacy($pdo, $data, $tables, $unknownTables)
```

- **reset:** Typically during restore modules are "reset" by uninstalling and reinstalling the module by executing the reset() method. You can override this in your own class to prevent the default action or perform other actions

```
/**
 * The reset method is run right before the restore method is executed
 *
 * You can override this and add custom code here
 *
 * Example. Framework skips this method by overwriting it and doing
nothing
 *
 * @return void
 */
public function reset() {
```

Legacy Restore Helper Methods

- **restoreLegacyAll:** Simply send this method the $pdo handler for the temporary database and it will run through all functions below and try to determine what needs to be restored

```
/**
 * Restore Legacy from All storage locations
 *
 * @param \PDO $pdo The pdo connection for the temporary database
 * @return void
 */
$this->restoreLegacyAll($pdo)
```

- **restoreLegacyDatabaseKvstore:** Simply send this method the $pdo handler for the temporary database and it will run through only the database and kvstore methods below and try to determine what needs to be restored

```
/**
 * Restores databases and kvstore based on present XML tables and
backup KVStore
 *
 * @param \PDO $pdo The pdo connection for the temporary database
 * @return void
 */
$this->restoreLegacyDatabaseKvstore($pdo)
```

- **restoreLegacyDatabase:** Simply send this method the $pdo handler for the temporary database and it will run through only the database methods below and try to determine what needs to be restored

```
/**
 * Restores database based on present XML tables and backup database
 *
 * @param \PDO $pdo The pdo connection for the temporary database
 * @return void
 */
$this->restoreLegacyDatabase($pdo)
```

- **restoreLegacyFeatureCodes:** Simply send this method the $pdo handler for the temporary database and it will run restore only the legacy feature codes

```
/**
 * Restore Legacy Feature Codes
 *
 * @param \PDO $pdo The pdo connection for the temporary database
 * @return void
 */
$this->restoreLegacyFeatureCodes($pdo)
```

- **restoreLegacySettings:** Simply send this method the $pdo handler for the temporary database and it will run restore only the legacy advanced settings

```
/**
 * Restore Legacy Advanced Settings
 *
 * @param \PDO $pdo The pdo connection for the temporary database
 * @return void
 */
$this->restoreLegacySettings($pdo)
```

- **restoreLegacyKvstore:** Simply send this method the $pdo handler for the temporary database and it will run restore only the legacy key /value store

```
/**
 * Restores kvstore based on backup KVStore
 *
 * @param \PDO $pdo The pdo connection for the temporary database
 * @return void
 */
$this->restoreLegacyKvstore($pdo)
```