```
a:1:{i:0;a:4:{s:8:"dog_name";s:7:"Spot";s:10:"dog_weight";s:2:"19";s:9:
"dog_color";s:5:"Green";s:9:"dog_breed";s:3:"Lab";}}
```

The data in a serialized string can be returned to an array format (or another format) using the `unserialize` method. The second line creates a string (`$chge_log_file`), which uses the `date` method and the log file name located in the `dog_applications` XML file to create a backup file name (and location). The string created is then passed to this log using the `error_log` method. The contents of the log file will look similar to the following.

```
07142015042510 | Insert | a:1:{i:0;a:4:{s:8:"dog_name";s:7:"tester1";s:10:
"dog_weight";s:2:"19";s:9:"dog_color";s:5:"Green";s:9:"dog_breed";s:3:"Lab";}}
07142015042510 | Update | a:1:{i:1;a:4:{s:8:"dog_name";s:7:"tester2";s:10:
"dog_weight";s:2:"19";s:9:"dog_color";s:5:"Green";s:9:"dog_breed";s:3:"Lab";}}
07142015042510 | Delete | 1
```

This format provides all the information needed to help with the recovery process. If the current version of the dog data file is corrupted, the change log file can be used to apply changes to a good version of the file to develop a new current version.

The only other changes needed to the `data` class are a few additional code lines in the destructor.

```
$new_valid_data_file = preg_replace('/[0-9]+/', '', $this->dog_data_xml);
// remove the previous date and time if it exists
$oldxmldata = date('mdYhis') . $new_valid_data_file;
if (!rename($this->dog_data_xml, $oldxmldata))
        {
          throw new Exception("Backup file $oldxmldata could not be created.");
        }
file_put_contents($new_valid_data_file,$xmlstring);
```

Before the destructor uses the `file_put_contents` method to apply changes to the XML file, a *backup* should be created in case the changes cause corruption to the current data. The recovery process will allow the support personnel to select which data file contains good data and which change file(s) will be applied to the data to produce the correct current version of the data.

Because this process may use a backup file of the data, which includes a file name with a date and time, the `preg_replace` method is used to remove any numerical information from the data file name. The regular expression (`/[0-9]+/`) in the first parameter directs the method to search for all occurrences of numbers in `$this->dog_data_xml`. If any occurrence is found, it is replaced with the value in the second parameter (`''`). In this case, nothing. The new file name is then placed in `$new_valid_data_file`. This will not cause any change to a "normal" non-backup file name because it does not contain any numerical information. A new backup file name is created using the file name in `$new_valid_data_file` with the date and time information. The new backup file name is stored in `$oldxmldata`.

Now the last valid data can be moved to the new backup file using the `rename` method. The data in `$this->dog_data_xml` (the location of the good data without changes) is copied to the new backup file location (`$oldxmldata`). If the file cannot be renamed, an exception is thrown.

Finally the valid changed data (located in `$xmlstring`) can be placed into the new location of the valid data (which is the same file name without any date information) contained in the `$new_valid_data_file` property.

For example, if `07142015042510dog_data.xml` contains the last valid data available, `07152015001510dog_data.xml` might be the new location of this data before any changes are applied. `dog_data.xml` would be the location of the valid data after changes have been applied. The last coding change to the dog data class is the inclusion of a `set` method.

```
function setChangeLogFile($value)
{
$this->dog_data_xml = $value;
}
```