```
private $dogs_array = array(); defined as an empty array initially
libxml:use_internal_errors(true);
function __construct() {
         $xmlfile = file_get_contents(get_dog_application("datastorage"));
         $xmlstring = simplexml:load_string($xmlfile);
         if ($xmlstring === false) {
                  $errorString = "Failed loading XML: ";
                  foreach(libxml:get_errors() as $error) {
                         $errorString .= $error->message . " ";  }
                  throw new Exception($errorString); }
$json = json_encode($xmlstring);
$this->dogs_array = json_decode($json,TRUE);
}
```

By default, XML parsing errors will cause the system to display the errors to the user and shut down the program. The `libxml:user_internal_errors(true)` method will suppress the errors. When the string is converted to XML format via the `simplexml:load_string` method, the XML is parsed to determine if it is valid. If it is not valid, the method will return `FALSE` instead of the XML information. The `if` statement shown will create an `$errorString` and use the `foreach` statement to loop through each error returned by the `libxml:get_errors` method (which returns an array containing the errors). Once all errors are collected, it will raise an exception passing the `$errorString`. The `dog_interface` program will catch this error and process it, as shown in Chapter 5.

This example does make one bad assumption (which simplifies the example). It assumes that the `$errorString` does not exceed the maximum capacity of 120 characters for the log file. A very badly formatted file could quickly cause `$errorString` to exceed this size. This limit can be adjusted in the PHP configuration file.

With the data automatically being saved whenever the data object is removed from memory, the insert, update, and delete methods only need to adjust the contents of the multidimensional associative array. Let's take a first look at creating a `delete` method since you have already seen an example in Chapter 5.

In the `readerrorlog` program (in Example 5-8) you created a `deleterecord` method. The method was used for regular multidimensional arrays. We could make a few adjustments to this routine to create the `deleteRecord` method for the `dog_data` class.

```
function deleteRecord($recordNumber) {
        foreach ($this->dogs_array as $dogs=>&$dogs_value) {
                for($J=$recordNumber; $J < count($dogs_value) -1; $J++)
                        {
                        foreach ($dogs_value[$J] as $column => $column_value)
                        {
                                $dogs_value[$J][$column] = $dogs_value[$J + 1][$column];
                        }
                }
unset ($dogs_value[count($dogs_value) -1]);
         }
 }
```

In the previous `deleterecord` method, the number of rows in the array and the array itself were passed into the method. The array in `dog_data` class is populated by the XML file containing the dog information. There is no property set with the number of records. This is not a problem. The PHP method **count** will return the size of an array. You can access and update the `dogs_array` (which is a protected private property) using the `$this` pointer. Methods in classes can use the `this` pointer to access and update protected properties; it is not necessary to pass them into a method. The only property you need to pass to the `deleteRecord` method is the record number (`$recordNumber`) to delete.