```
function updateRecords($records_array)
{
        foreach ($records_array as $records=>$records_value)
        {
                foreach ($records_value as $record => $record_value)
                {
                        $this->dogs_array["dog"][$records] = $records_array[$records];
                }
        }
}
}
?>
```

■ **Note** An alternative solution which handles associate arrays with missing indexes, and the possibility that the `dog_data.xml` file may contain one or zero records, is provided on the textbook website.

The only change in this final version of the `dog_data` class is the inclusion of `get_dog_application` method code in the constructor to retrieve the location and name of the XML file holding the dog data.

Example 6-2. The testdata.php file

```
<?php
include("dog_data.php");
$tester = new dog_data();
$records_array = Array (
0 => Array ( "dog_name" => "Sally", "dog_weight" => "19", "dog_color" => "Green",
"dog_breed" => "Lab" ));
$tester->insertRecords($records_array);
print_r ($tester->readRecords("ALL"));
print("<br>");

$records_array = Array (
1 => Array ( "dog_name" => "Spot", "dog_weight" => "19", "dog_color" => "Green",
"dog_breed" => "Lab" ));

$tester->updateRecords($records_array);
print_r ($tester->readRecords("ALL"));
print("<br>");

$tester->deleteRecord(1);
print_r ($tester->readRecords("ALL"));
$tester = NULL; // calls the destructor and saves the xml records in the file
?>
```

Example 6-2 tests some of the possible scenarios of using the `dog_data` class. Notice the last line of code calls the destructor (to save the data). This is accomplished by setting the pointer to the object (`$tester`) to `NULL`, which releases the object. This will inform the garbage collector of the operating system that the object should be removed from memory. This will cause the destructor to execute, which will update the XML file and remove the object from the memory of the server.