If everything was successful with the update, instead of using echo or print to display a message, the program will set $_SESSION['message'] with the message, which will then be displayed by lab.php.

```
header("Location: lab.php");
```

The application is then redirected back to lab.php. lab.php will verify that is was called from dog_interface and then display "Dog $dog_name Insert/Update was successful
" at the top of the page. ($dog_name is replaced with the actual dog name.)

If the request is delete, a similar process occurs.

```
else if($_POST['delete'])
        {
                $properties_array = $dog_index;
```

The dog_data delete method only needs the position in the array to determine what to remove. Thus, $properties_array is set to the value in $dog_index. Even though $properties_array is now a string and not an array, the processRecords method in dog_data uses polymorphism to accept an array or a string. This allows the code to be very similar to the update and insert code.

```
$lab = $container->create_object($properties_array);
$_SESSION['message'] = "Dog $dog_name Deletion was successful<br />";
header("Location: lab.php");
```

As seen with update and delete, $container (which is an instance of dog_container already created) calls the create_object method to create an instance of the dog class and pass $properties_array (which is really a string). If the delete is successful, $_SESSION['message'] is set with the delete message. Then the lab.php program is called. lab.php will verify that dog_container called it and then display "Dog $dog_name Deletion was successful
" at the top of the page. ($dog_name is replaced with the actual dog name.)

These are the only code changes needed in the dog_interface program in order to handle the request to insert, update, or delete. dog_interface must also accept the dogs list box and complete dogs array from the data tier to format and send to lab.php.

dog_interface must request the dogs array information by calling the display method in dog_data.php.

```
$container = NULL;
```

The container pointer, $container, can be reused after the request to return the breeds list has been processed. By setting it to NULL, it will free up the current container (an instance of dog_container with properties set for retrieving the breeds information).

```
$container = new dog_container("dog");
```

A new instance of the dog_container passes "dog" instead of "selectbox". This lets the container know than an instance of the dog class will be created, not an instance of the breeds class.

```
$properties = "dog";
$lab = $container->create_object($properties);
```