```
                {
                        $this->dogs_array["dog"][$records] = $records_array[$records];
                }
        }
$change_string = date('mdYhis') . " | Update | " . serialize($records_array) . "\n";
$chge_log_file = date('mdYhis') . $this->change_log_file;
        error_log($change_string,3,$chge_log_file); // might exceed 120 chars
}
function setChangeLogFile($value)
{
        $this->dog_data_xml = $value;
}
function processRecords($change_Type, $records_array)
{
switch($change_Type)
{
        case "Delete":
                $this->deleteRecord($records_array);
                break;
        case "Insert":
                $this->insertRecords($records_array);
                break;
        case "Update":
                $this->updateRecords($records_array);
                break;
        default:
                throw new Exception("Invalid XML file change type: $change_Type");
} } }
```

■ **Note** An alternative solution is provided on the textbook website which will allow associate arrays with missing indexes and one or no records within the `dog_data.xml` file.

Now that you have the ability to provide backup and recovery, let's make some adjustments to the `readerrorlog` file (in Example 5-8). The new application will need to allow the support personnel to select (and modify) any valid change log file, select the most valid data file available, and apply the changes from the change log file to produce a new valid data XML file.

```
if(isset($_POST['data_File']))
{
          update_XML_File_Process();
}
else if(isset($_GET['rn']))
{
          delete_Process();
}
else if(isset($_POST['change_file']))
{
          display_Process();
}
```