```
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;
        case "Display":
                $this->readRecords($records_array);
                Break;
        default:
                throw new Exception("Invalid XML file change type: $change_Type");
}
}
```

All requests for changes will now be passed through this method. The method accepts a change type (`Insert`, `Delete`, `Update`, or `Display`) and the array (for `Insert` or `Update`) or the record number (for `Delete` or `Update`). The values are passed into `$record_array`. `$record_array` is dynamically created as either an array or a string. This allows the `processRecords` method to provide *polymorphism* (the ability for the same method call to accept different parameters), which is one of the requirements of an object-oriented language (along with encapsulation and inheritance). The `switch` statement looks at `$change_Type` to determine which method to call. It then calls the related method. If an invalid type is passed, an exception is thrown.

> *Security and performance—In a 'live' environment, it would be more secure to pass "codes" into this type of method instead of using a value that indicates the action that will take place. For example, 101 could be used to indicate an update. The* `switch` *statement could easily be adjusted to examine the codes to determine which method to call.*

Each of the methods that you have examined previously (except for the constructor and destructor) are now set to `'private'`. This makes the process much more secure; changes can only occur by using the `processRecords` method. Three code lines have also been added to the end of each of the three methods to provide backup and recovery capabilities.

```
...
$change_string = date('mdYhis') . " | Delete | " . $recordNumber . "\n";
$chge_log_file = date('mdYhis') . $this->change_log_file;
error_log($change_string,3,$chge_log_file); // might exceed 120 chars
...
```

The first line formats a string for a change log file. The format used is similar to the format you look at in Chapter 5. In the previous example, the record number is passed as required for the `delete` method.

```
$change_string = date('mdYhis') . " | Update | " . serialize($records_array) . "\n";
```

For `update` and `insert`, the arrays are passed. However, an array cannot be placed into a string. The `serialize` method transforms an array into a string format similar to the following.