# Managing Simple Mailing List

# Lesson 9a

In this lesson, you'll learn the methods for creating a managed distribution list, which can be used to send out newsletters or anything else that you want to send, to a list of email addresses in a database.

The mailing mechanism you'll use in this lesson is not meant to be a replacement for mailing list software, which is specifically designed for bulk messages. The type of system you'll build in this lesson should only be used for small lists, fewer than a few hundred email addresses.

In this lesson, you will learn how to

- Create a subscribe/unsubscribe form and script
- Create a front end for sending your message
- Create the script that sends your message

## Developing the Subscription Mechanism

You learned in earlier lessons that planning is the most important aspect of creating any product. In this case, think of the elements you will need for your subscription mechanism:

- A table to hold email addresses
- A way for users to add or remove their email addresses
- A form and script for sending the message

The following sections will describe each item individually.

## Creating the `subscribers` Table

You really need only one field in the `subscribers` table: to hold the email address of the user. However, you should have an ID field just for consistency among your tables, and also because referencing an ID is a lot simpler than referencing a long email address in `where` clauses. So, in this case, your MySQL query would look something like

```
mysql> create table subscribers (
    -> id int not null primary key auto_increment,
    -> email varchar (150) unique not null
    -> );
Query OK, 0 rows affected (0.00 sec)
```

Note the use of `unique` in the field definition for `email`. This means that although `id` is the primary key, duplicates should not be allowed in the `email` field either. The `email` field is a unique key, and `id` is the primary key.

```
mysql> desc subscribers;
+-------+--------------+------+-----+---------+----------------+
| Field | Type         | Null | Key | Default | Extra          |
+-------+--------------+------+-----+---------+----------------+
| id    | int(11)      |      | PRI | NULL    | auto_increment |
| email | varchar(150) |      | UNI |         |                |
+-------+--------------+------+-----+---------+----------------+
2 rows in set (0.00 sec)
```

Now that you have a table, you can create the form and script that place values in there.

## Creating the Subscription Form

The subscription form will actually be an all-in-one form and script called `manage.php`, which will handle both subscribe and unsubscribe requests. Listing 1 shows the code for `manage.php`, which uses a few user-defined functions to eliminate repetitious code. The code looks long, but a line-by-line description follows, and the majority of the code is an HTML form, so no worries!

### *Listing 1. Subscribe and Unsubscribe with `manage.php`*

```
 1: <?php
 2: //set up a couple of functions
 3: function doDB() {
 4:     global $conn;
 5:    //connect to server and select database; you may need it
 6:    $conn = mysql_connect("localhost", "joeuser", "somepass")
 7:         or die(mysql_error());
 8:    mysql_select_db("testDB",$conn) or die(mysql_error());
 9: }
10:
11: function emailChecker($email) {
12:     global $conn, $check_result;
13:    //check that email is not already in list
14:    $check = "select id from subscribers where email = '$email'";
15:    $check_result = mysql_query($check,$conn) or
die(mysql_error());
16: }
17:
18: //determine if they need to see the form or not
19: if ($_POST[op] != "ds") {
20:    //they do, so create form block
21:    $display_block = "
22:    <form method=POST action=\"$_SERVER[PHP_SELF]\">
23:
24:    <p><strong>Your E-Mail Address:</strong><br>
25:    <input type=text name=\"email\" size=40 maxlength=150>
26:
27:    <p><strong>Action:</strong><br>
28:    <input type=radio name=\"action\" value=\"sub\" checked>
subscribe
```

```
29:     <input type=radio name=\"action\" value=\"unsub\"> unsubscribe
30:
31:     <input type=\"hidden\" name=\"op\" value=\"ds\">
32:
33:     <p><input type=submit name=\"submit\" value=\"Submit
Form\"></p>
34:     </form>";
35:
36: } else if (($_POST[op] == "ds") && ($_POST[action] == "sub")) {
37:     //trying to subscribe; validate email address
38:     if ($_POST[email] == "") {
39:         header("Location: manage.php");
40:         exit;
41:     }
42:     //connect to database
43:     doDB();
44:     //check that email is in list
45:     emailChecker($_POST[email]);
46:
47:     //get number of results and do action
48:     if (mysql_num_rows($check_result) < 1) {
49:         //add record
50:         $sql = "insert into subscribers values('',
'$_POST[email]')";
51:         $result = mysql_query($sql,$conn) or die(mysql_error());
52:         $display_block = "<P>Thanks for signing up!</P>";
53:     } else {
54:         //print failure message
55:         $display_block = "<P>You're already subscribed!</P>";
56:     }
57: } else if (($_POST[op] == "ds") && ($_POST[action] == "unsub")) {
58:     //trying to unsubscribe; validate email address
59:     if ($_POST[email] == "") {
60:         header("Location: manage.php");
61:         exit;
62:     }
63:     //connect to database
64:     doDB();
65:     //check that email is in list
66:     emailChecker($_POST[email]);
67:
68:     //get number of results and do action
69:     if (mysql_num_rows($check_result) < 1) {
70:         //print failure message
71:         $display_block = "<P>Couldn't find your address!</P>
72:         <P>No action was taken.</P>";
73:     } else {
74:         //unsubscribe the address
75:         $id = mysql_result($check_result, 0, "id");
76:         $sql = "delete from subscribers where id = '$id'";
77:         $result = mysql_query($sql,$conn) or die(mysql_error());
78:         $display_block = "<P>You're unsubscribed!</p>";
79:     }
80: }
81: ?>
82: <HTML>
83: <HEAD>
84: <TITLE>Subscribe/Unsubscribe</TITLE>
85: </HEAD>
86: <BODY>
87: <h1>Subscribe/Unsubscribe</h1>
```

```
88: <?php echo "$display_block"; ?>
89: </BODY>
90: </HTML>
```

Listing 1 may be long, but it's not complicated. In fact, it could be longer, were it not for the user-defined functions at the top of the script. One of the reasons for creating your own functions is to be able to reuse it within your scripts. Lines 39 set up the first function, `doDB()`, which is simply the database connection you've been making in your lessons for a while now. Lines 11 - 16 define a function called `emailChecker()`, which takes an input and returns an output like most functions do. We'll look at this one in the context of the script, as we get to it.

Line 19 starts the main logic of the script. Because this script performs several actions, we need to determine which action it is currently attempting. If the value of `$_POST[op]` is not `"ds"` (that stands for "do something"), we know the user has not submitted the form; therefore, we must show it to the user.

Lines 21 - 34 create the subscribe/unsubscribe form, using `$_SERVER[PHP_SELF]` as the action (line 22), creating a text field called `email` for the user's email address, and setting up a set of radio buttons (lines 28 - 29) to find the desired task. At this point, the script breaks out of the `if...else` construct, skips down to line 82, and proceeds to print the HTML. The form is displayed as shown in Figure 1.

***Figure 1. The subscribe/unsubscribe form.***


Back inside the `if...else` construct, if the value of `$_POST[op]` is indeed `"ds"`, we need to do something. There are two possibilities: subscribing and unsubscribing. We determine which action to take by looking at the value of `$_POST[action]`, from the radio button group.

In line 36, if the value of `$_POST[op]` is `"ds"` and the value of `$_POST[action]` is `"sub"`, we know the user is trying to subscribe. To subscribe, he will need an email address, so we check for one in lines 38 - 41. If no address is present, the user is redirected back to the form.

However, if an address is present, we call the `doDB()` function in line 43 to connect to the database, to issue a query (or two). In line 45, we call the second of our user-defined functions: `emailChecker()`. This function takes an input (`$_POST[email]`, in this case) and processes it. If we look back to lines 12 - 15, we see that the function is checking for an `id` value in the `subscribers` table, when the email address stored in the record that matches the value passed to the function. The function then returns the result set, called `$check_result`, for use within the larger script.

NOTE:

Note the definition of global variables at the beginning of both user-defined functions in Listing 1. These variables need to be shared with the entire script, and so are declared global.

Jump down to line 48 to see how the `$check_result` variable is used: The number of records referred to by the `$check_result` variable is counted to determine whether the email address already exists in the table. If the number of rows is less than 1, the address is not in the list, and it can be added. The record is added and the response is stored in lines 50 - 52, and the failure message (if the address is already in the table) is stored in line 55. At that point, the script breaks out of the `if...else` construct, skips down to line 82, and proceeds to print the HTML. You'll test this functionality later.

The last combination of inputs occurs if the value of the `$_POST[op]` variable is `"ds"` and the value of the `$_POST[action]` variable is `"unsub"`. In this case, the user is trying to unsubscribe. To unsubscribe, he will need an email address, so we check for one in lines 59 - 61. If no address is present, the user is sent back to the form.

If an address is present, we call the `doDB()` function in line 64 to connect to the database. Then, in line 66, we call `emailChecker()`, which again will return the result set, `$check_result`. The number of records in the result set is counted in line 69, to determine whether the email address already exists in the table. If the number of rows is less than 1, the address is not in the list, and it cannot be unsubscribed.

In this case, the response message is stored in lines 71 - 72. The user is unsubscribed (the record deleted) and the response is stored in lines 75 - 77, and the failure message (if the address is already in the table) is stored in line 78. At that point, the script breaks out of the `if...else` construct, skips down to line 82, and proceeds to print the HTML.

Figures 2 through 5 show the various results of the script, depending on the actions selected and the status of email addresses in the database.

*Figure 2. Successful subscription.*


*Figure 3. Subscription failure.*

*Figure 4. Successful unsubscribe action.*

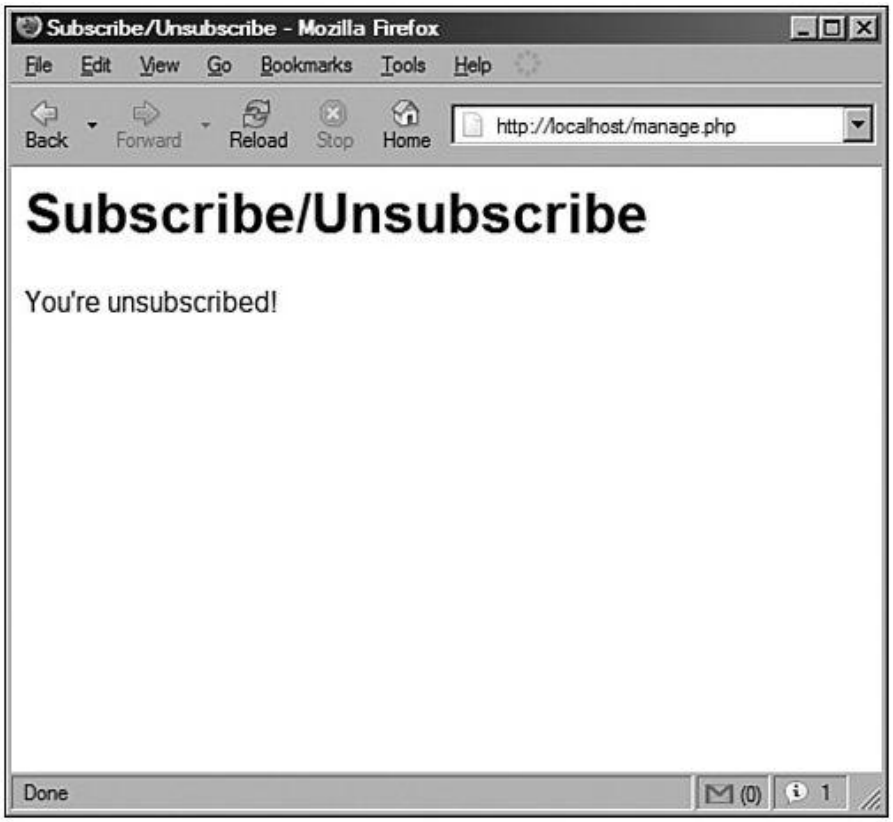



*Figure 5. Unsuccessful unsubscribe action.*

Next, you'll create the form and script that sends along mail to each of your subscribers.

## Developing the Mailing Mechanism

With the subscription mechanism in place, you can create a basic form interface for a script that will take the contents of your form and send it to every address in your `subscribers` table. This is another one of those all-in-one scripts, called `sendmymail.php`, and it is shown in Listing 2.

### *Listing 2. Send Mail to Your List of Subscribers*

```
 1: <?php
 2: if ($_POST[op] != "send") {
 3:    //haven't seen the form, so show it
 4:    echo "
 5:    <HTML>
 6:    <HEAD>
 7:    <TITLE>Send a Newsletter</TITLE>
 8:    </HEAD>
 9:    <BODY>
10:    <h1>Send a Newsletter</h1>
11:    <form method=\"post\" action=\"$_SERVER[PHP_SELF]\">
12:    <P><strong>Subject:</strong><br>
13:    <input type=\"text\" name=\"subject\" size=30></p>
14:    <P><strong>Mail Body:</strong><br>
15:    <textarea name=\"message\" cols=50 rows=10
wrap=virtual></textarea>
16:    <input type=\"hidden\" name=\"op\" value=\"send\">
17:    <p><input type=\"submit\" name=\"submit\" value=\"Send
It\"></p>
18:    </FORM>
19:    </BODY>
20:    </HTML>";
21:
22: } else if ($_POST[op] == "send") {
23:     //want to send form, so check for required fields
24:     if (($_POST[subject] =="") || ($_POST[message] == "")) {
25:        header("Location: sendmymail.php");
26:        exit;
27:    }
28:
29:    //connect to database
30:    $conn = mysql_connect("localhost", "joeuser", "somepass")
31:         or die(mysql_error());
32:    mysql_select_db("testDB",$conn) or die(mysql_error());
33:
34:    //get emails from subscribers list
35:    $sql = "select email from subscribers";
36:    $result = mysql_query($sql,$conn) or die(mysql_error());
37:
38:    //create a From: mailheader
39:    $headers = "From: Your Mailing List <you@yourdomain.com>";
40:
41:    //loop through results and send mail
42:    while ($row = mysql_fetch_array($result)) {
43:        set_time_limit(0);
```

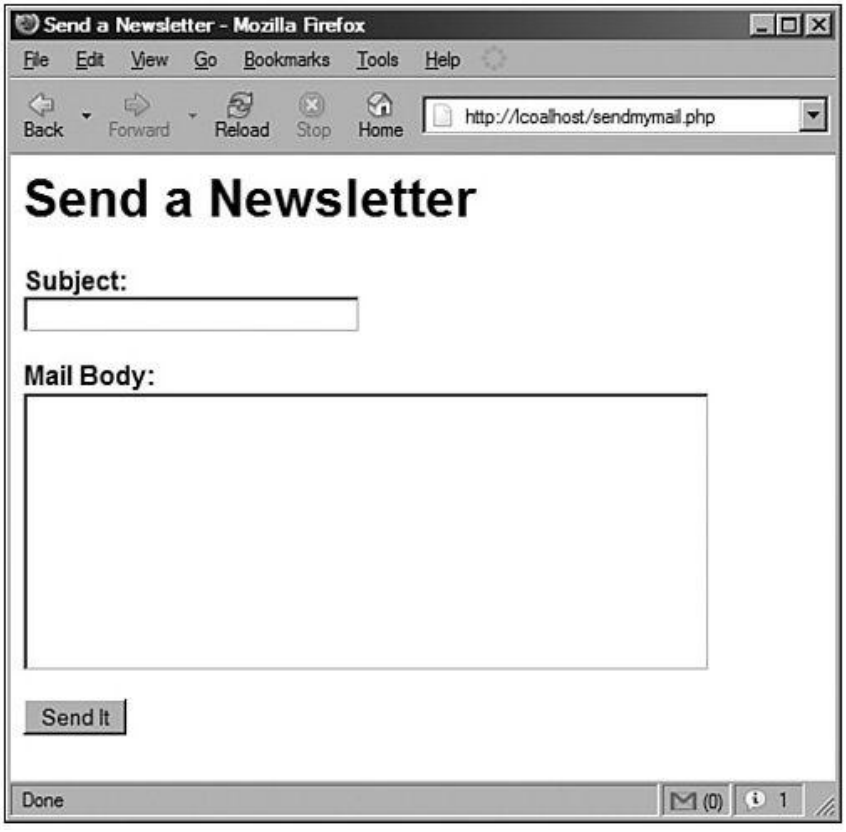
```
44:         $email = $row['email'];
45:         mail("$email", stripslashes($_POST[subject]),
46:              stripslashes($_POST[message]), $headers);
47:         echo "newsletter sent to: $email<br>";
48:     }
49: }
50: ?>
```

The main logic of the script starts right there at line 2, where we determine whether the user has seen the form yet. If the value of the `$_POST[op]` variable is not `"send"`, we know the user has not submitted the form; therefore, we must show it.

Lines 4 - 20 create the form for sending the newsletter to your subscriber list, which uses `$_SERVER[PHP_SELF]` as the action (line 11), creates a text field called `subject` for the subject of the mail, and creates a `textarea` called `message` for the body of the mail to be sent.

At this point, the script breaks out of the `if...else` construct and the HTML is printed. The form is displayed as in Figure 6.

***Figure 6. Form for sending the bulk mail.***




If the value of the `$_POST[op]` variable is indeed `"send"`, we have to send the form to the recipients. Before we send the message, we must check for the two required items

from the form: `$_POST[subject]` and `$_POST[message]`. If either of these items is not present, the user is redirected to the form again.

If the required items are present, the script moves on to lines 30 - 32, which connect to the database. A query is issued in line 36, which grabs all the email addresses from the `subscribers` table. There is no order to these results, although you could throw an `order` by clause in there if you want to send them out in alphabetical order for whatever reason.

Line 39 creates a `From:` mail header, which is used inside the upcoming `while` loop, when the mail is sent. This header ensures that the mail looks like it is from a person and not a machine, as you've specifically provided a value in this string. The `while` loop, which begins on line 42, extracts the email addresses from the result set, one at a time. On line 43, we use the `set_time_limit()` function to set the time limit to `0`, or "no limit." Doing so allows the script to run for as long as it needs to.

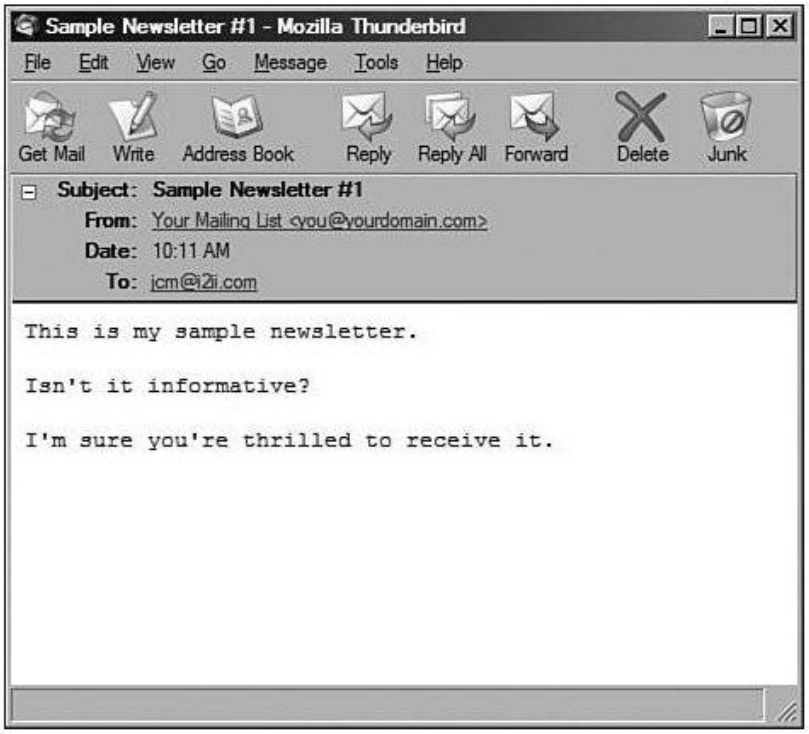
NOTE:

Because all the script in Listing 2 does is execute the `mail()` function numerous times, it does not take into account the queuing factors in actual mailing list software, which are designed to ease the burden on your outgoing mail server. Using `set_time_limit()` does not ease its burden; it just allows the script to continue to run when it might have timed out before.

In line 45, the mail is sent using the `mail()` function, inserting the values from the form where appropriate. Line 46 prints a message to the screen for you, to show who should have received the mail. In Figures 7 and 8, you can see the outcome of the script.

*Figure 7. Mail has been sent!*


*Figure 8. The mail arrived safely.*



## Q&A

**Q** How can I ease the burden on my mail server?

**A** Besides looking into package mailing list software, you can bypass the `mail()` function and talk directly to your SMTP server via a socket connection. Such an example is shown in the PHP manual for the `fsockopen()` function (http://www.php.net/manual/fsockopen), as well as in other developer resource sites.

**Q** Where do bounced messages go?

**A** Bounces go to whatever address you specify in your `From:` or `Reply-to:` mail headers.

## Workshop

The workshop is designed to help you anticipate possible questions, review what you've learned, and begin learning how to put your knowledge into practice.

## Quiz

1. What function sends mail?
2. What function call causes the script to execute for as long as it needs to run?

## Answers

1. This is not a trick question. It's the `mail()` function!
2. `set_time_limit(0)`