CHAPTER EIGHT

ORDER PROCESSING SETUP

The final step of any online store is the actual placement of the customer orders. Unfortunately, this is also an area where a great number of changes are made in order to accommodate the individual logic needed by each store front.

For example, the logic for ordering from some online stores is straightforward. All the store owner wants is a subtotal of all the items and then uses this total to charge the customer. However, other store owners may want more complex logic including the automatic calculation of sales tax, automatic discounts based on volume or quantity purchased, or automatic shipping costs calculated based on whether the customer wants to ship via UPS, U.S. Postal Service, FedEx, or other shipping carriers.

The Web store order processing has been programmed with these various scenarios and more in mind. Through mere changes in the Setup file, you should be able to accommodate any type of order processing logic you need to express. In addition, since ordering frequently involves the processing of credit card information, we will cover server security related to orders in this and the next chapter.

Changes to the Outlet Order Form

The outlet order form is the heart of order processing. In the Setup file (discussed below) there is a variable that specifies which HTML file contains the Web store's order form. This is the form displayed to users when they click the **Checkout**

Stand button while they are shopping. The Web Store script displays the contents of their cart as well as a form on which they can fill in information related to the order they are placing. By default, this file generally called **outlet_order_form.html**.

You may wonder, if this is a static HTML file, how dynamic information such as the user's cart contents can be displayed inside the HTML form. The answer lies in the fact that the **web_store.cgi** script actually filters the outlet order form before displaying it. When **web_store.cgi** finds certain tags, it replaces them with this dynamic information.

The two tags the script looks for are the form tag and the tag that lets the **web_store.cgi** script know where to place the displayed cart contents. The first tag is the FORM tag. If the script finds any line that contains a <FORM string in it (FORM tag), it will replace this with a new <FORM> tag that actually points to the script specified in the Setup file for order processing (**$sc_order_script_url**). Special hidden input-field tags that include information about the user's current cart i.d. are also presented after the replaced <FORM> tag. For example, the following line in the default **outlet_order_form.html** file would be replaced when it is filtered by **web_store.cgi**:

```
<FORM ACTION = "web_store.cgi" METHOD = "post">
```

An example of the code that would replace it appears below.

```
<FORM ACTION="https://www.yourdomain.com/cgi-
bin/Web_store/web_store.cgi" METHOD="post">
<INPUT TYPE = "hidden" NAME = "page"
             VALUE = "[PREVIOUS PAGE]">
<INPUT TYPE = "hidden" NAME = "cart_id"
             VALUE = "[CART ID]">
```

Also, the tag <H2>CART CONTENTS HERE</H2> automatically gets replaced with a display of the user's cart contents. Figure 8.1 displays an example of what the resulting Web store order form would look like.

Figure 8.1 Sample order form display.

The rest of the order form contains the actual input elements for users to enter when they wish to make an order. Obviously, users need to enter certain basic information such as first name, last name, address to send the order to, how they intend to pay for the order, and more. In the Setup file discussion below, you will find that there is an array that specifies the names of the <INPUT> tags you can use on the form. These input tags must have two-digit numbers preceding them so that the Web store script can automatically determine the correct order to display the information back to the user once it has been entered in. An example of the HTML that exists in **outlet_order_form.html** appears below:

```
<HTML>
<HEAD>
<TITLE>Standard Order Form</TITLE>
</HEAD>
<BODY BGCOLOR="#ffffff">

<!-- The following will be replaced with
     another form tag plus hidden input fields
-->
<FORM ACTION = "web_store.cgi" METHOD = "post">

<!-- The following will be replace with
     the contents of the users cart
-->
<H2>CART CONTENTS HERE</H2>

<CENTER>
<TABLE WIDTH="90%" BORDER="3" CELLPADDING="2">

<TR>
<TD COLSPAN="2"><FONT SIZE=+1>Personal Information:</FONT></TD>
</TR>
<TR><TD>Name(First &amp;amp; Last)</TD>
<TD><INPUT TYPE="text" NAME="01-name" SIZE="30" MAXLENGTH="30"></TD>
</TR>

<TR>
<TD COLSPAN="2"><FONT SIZE="+1">Billing Address:</FONT></TD>
</TR>

<TR>
<TD>Street:</TD>
<TD><INPUT TYPE="text" NAME="02-b_street_address" SIZE="30"></TD>
</TR>

<TR>
<TD COLSPAN="2">City:
<INPUT TYPE="text" NAME="03-b_city" SIZE="10">
<INPUT TYPE="text" NAME="03-b_city" SIZE="10">
State:
<INPUT TYPE="text" NAME="04-b_state" SIZE="2" MAXLENGTH="8">
Zip:
<INPUT TYPE="text" NAME="05-b_zip" SIZE="5" MAXLENGTH="5">
Country:
```

```
<INPUT TYPE="text" NAME="06-b_country" SIZE="10" MAXLENGTH="20">
</TD>
</TR>

<TR>
<TD COLSPAN="2"><FONT SIZE="+1">Mailing Address (If
Different):</FONT></TD>
</TR>

<TR>
<TD>Street:</TD>
<TD><INPUT TYPE="text" NAME="07-m_street_adress" SIZE="30"></TD>
</TR>

<TR>
<TD COLSPAN="2">
City:
<INPUT TYPE="text" NAME="08-m_city" SIZE="10"> State:
<INPUT TYPE="text" NAME="09-m_state" SIZE="2" MAXLENGTH="8">
Zip:
<INPUT TYPE="text" NAME="10-m_zip" SIZE="5"
MAXLENGTH="5">Country:
<INPUT TYPE="text" NAME="11-m_country" SIZE="10" MAXLENGTH="20">
</TD>
</TR>

<TR>
<TD>
Phone:
<INPUT TYPE="text" NAME="12-phone" SIZE="10" MAXLENGTH="12">
</TD>
<TD>
Fax:
<INPUT TYPE="text" NAME="13-fax" SIZE="10" MAXLENGTH="12">
</TD>
</TR>

<TR>
<TD>
E-Mail:
</TD>
<TD>
<INPUT TYPE="text" NAME="14-e-mail" MAXLENGTH="30">
</TD>
</TR>
```

```
<TR>
<TD>
URL:
</TD>
<TD>
<INPUT TYPE="text" NAME="15-URL" MAXLENGTH="30"> Request Link:
<INPUT TYPE="checkbox" NAME="16-link" VALUE="on">
</TD>
</TR>

<TR>
<TD COLSPAN="2">
<FONT SIZE="+1">Credit Card Information:</FONT>
</TD>
</TR>

<TR>
<TD COLSPAN="2">
<INPUT TYPE="radio" NAME="17-type_of_card" VALUE="visa">Visa
<INPUT TYPE="radio" NAME="17-type_of_card"
VALUE="mastercard">Mastercard
<INPUT TYPE="radio" NAME="17-type_of_card" VALUE="discover">Discover
</TD>
</TR>

<TR>
<TD>
Name on Card:
</TD>
<TD>
<INPUT TYPE="text" NAME="19-cardname" SIZE="30" MAXLENGTH="30">
</TD>
</TR>

<TR>
<TD>
Number:
</TD>
<TD>
<INPUT TYPE="text" NAME="20-card_number" MAXLENGTH="20">
</TD>
</TR>
```

```
<TR>
<TD>
Exp. Date:
</TD>
<TD>
<INPUT TYPE="text" NAME="21-ex_date" SIZE="10" MAXLENGTH="10">
</TD>
</TR>

</TABLE>
</CENTER>

<CENTER>
<P>
Allow 3-4 weeks for delivery. Shipping prices and delivery times may
vary when shipping to cities outside the continental US.
<P>
<INPUT TYPE=reset>
<INPUT TYPE=submit NAME = "submit_order_form_button"
            VALUE = "Submit Secure Order"><BR>
</CENTER>
</FORM>
</BODY></HTML>
```

Notice that the bottom of the HTML form contains an <INPUT> submit button. You can change the caption on this button to whatever you want. For example, you may wish to simply call it "Submit Order" if your server is not really a secure server. However, you absolutely must keep the **NAME** of the <INPUT> tag equal to **submit_order_form_button**.

To summarize, there are four things to keep in mind when editing the **outlet_order_form.html** file. First, the <FORM> tag must exist so that the web_store.cgi script can replace it with the appropriate form tag along with hidden variables that need to be passed along such as the user's cart id. Second, the **<H2>CART CONTENTS HERE</H2>** must exist so that it can be replaced with the actual user's cart contents. Third, the form variables for the order form that are specified in the Setup file must exist in the order form HTML file so that the user can fill them out. (The Setup file will be discussed in greater detail below.) Fourth, the **submit_order_form_button** submit tag must be in the form so that the form can actually be sent to the **web_store.cgi script**.

Changes to the Setup File

Like most of the other parameters for the Web store, the ordering logic is specified inside the Setup file. The following is a list of variable inside the Setup file that affect order processing. Example values for these fields will be displayed below the total descriptions for these fields.

- **$sc_mail_lib_path** is the location of **mail-lib.pl**, which is used to mail nonencrypted email.
- **$sc_order_lib_path** is the location of **web_store_order_lib.pl**, which contains the routines that process orders.
- **$sc_pgp_lib_path** is the location of **pgp-lib.pl**, which has a routine to automatically encrypt final cart orders for sending in email or logging to a file. You must have installed PGP on your Web server and configured it for use previously. Setting up and using PGP with the Web store will be discussed further in Chapter 9.
- **%sc_order_form_array** is the associative array of form variables that are used on the order form to send in an order. These variables may ask for the user's name, address, or other information. The array maps a form field name with a descriptive name so that a well-formatted email will be produced later.

The form field names must begin with a two-digit number followed by a dash. For example, the descriptive field name **First Name** might correspond with a field name **01-fname**. The prefixed numbers tell the Web store script what order to process the form variables when the orders finally get emailed or logged to a file. Thus, **01-fname** would be processed before **02-lname**. The prefixed number is used so that the Web store can sort the values from the form. If the example order form from the previous section was to be sent, this variable would look like the following code:

```
%sc_order_form_array =('01-name', 'Name',
     '02-b_street_address', 'Billing Address Street',
     '03-b_city', 'Billing Address City',
     '04-b_state', 'Billing Address State',
```

```
'05-b_zip', 'Billing Address Zip',
'06-b_country', 'Billing Address Country',
'07-m_street_adress', 'Mailing Address Street',
'08-m_city', 'Mailing Address City',
'09-m_state', 'Mailing Address State',
'10-m_zip', 'Mailing Address Zip',
'11-m_country', 'Mailing Address Country',
'12-phone', 'Phone Number',
'13-fax', 'Fax Number',
'14-e-mail', 'Email',
'15-URL', 'URL',
'16-link', 'Link',
'17-type_of_card', 'Type of Card',
'18-cardname', 'Name Appearing on Card',
'19-card_number', 'Card Number',
'20-ex_date', 'Card Expiration',
'22-shipping', 'Shipping Method');
```


NOTE

Remember to always define the numbers as *two* digits. This should be done because the array elements are sorted on their string value rather than their numeric value. Thus, a string like **12** actually comes before **3** because the ASCII value of the first character in the string **12** is less than **3**. To avoid this problem, all the numbers in this array should have two digits. In a string comparison, **12** is greater than **03**.

- **@sc_order_form_required_fields** is an array containing the form field names (as defined in **%sc_order_form_array**) that are required fields. The order will not be processed without these field names being entered on the form. If the user fails to enter any of these fields into the order form, the script will inform them of which fields they failed to enter.
- **$sc_order_with_hidden_fields** is **yes** or **no**. If you want to submit orders to another server or to a MAILTO: URL, then you can use this option to make sure that hidden fields are actually generated with the contents of the cart in them. Chapter 9 goes into more detail regarding situations where you may wish to submit the cart information with hidden fields.

The following variables determine the shipping logic. While a brief overview of them is given below, subsequent sections in this chapter will go into greater detail as to examples of their use:

- **$sc_calculate_discount_at_display_form** is a numerical variable that tells the script how to calculate discounts at the display of the order form. If this value is **0**, it tells the script not to process discounts at the display of the order form at all. If the value is **1**, **2**, or **3** then the script will process discounts relative to the numbers that are set for other order-processing variables.
- **$sc_calculate_discount_at_process_form** is just like the above variable but instead of telling the script how to process discounts at the stage where the order form is displayed, it tells the script how to process discounts at the stage where the order form has been submitted and is currently being processed.
- **$sc_calculate_shipping_at_display_form** is a numerical variable that tells the script how to calculate shipping costs at the display of the order form. If this value is **0**, it tells the script not to process shipping at the display of the order form at all. If the value is **1**, **2**, or **3,** then the script will calculate shipping relative to the numbers that are set for other order-processing variables.
- **$sc_calculate_shipping_at_process_form** is just like the preceding variable but instead of telling the script how to process shipping at the stage where the order form is displayed, it tells the script how to process shipping at the stage where the order form has been submitted and is currently being processed.
- **$sc_calculate_sales_tax_at_display_form** is a numerical variable that tells the script how to calculate sales tax at the display of the order form. If this value is 0, it tells the script not to process sales tax at the display of the order form at all. If the value is 1, 2, or 3 then the script will process sales tax relative to the numbers that are set for other order processing variables.
- **$sc_calculate_sales_tax_at_process_form** is just like the above variable but instead of telling the script how to process sales tax at the stage where the order form is displayed, it tells the script how to process sales tax at the stage where the order form has been submitted and is currently being processed.

- **@sc_order_form_shipping_related_fields** is an array containing the names of the form variables on the order form that will be used in calculating shipping. If you are calculating shipping without regard to form values, leave this array empty. The field names here correspond to the form field names in the **%sc_order_form_array**.
- **@sc_order_form_discount_related_fields** is an array containing the names of the form variables on the order form that will be used in calculating a discount for the user. If you are calculating a discount without regard to form values, leave this array empty. The field names here correspond to the form field names in the **%sc_order_form_array**.
- **@sc_shipping_logic** is an array containing the logic for applying the shipping cost to the order. Each criteria is a separate list element. The fields within the criteria are pipe-delimited (|).

The values of the criteria are equal whole values (such as UPS or 5 or 11) or they can be ranges separated by hyphens (for example, 1–5, 1–, –5). If a second number is left off the hyphen, then the range is open-ended up to the value defined by the hyphen. For example, "5–" means anything greater than or equal to 5.

The first fields correspond to the fields in the **@sc_order_form_shipping_related_fields** array. If this array is empty, then no fields in **@sc_shipping_logic** will correspond to the shipping.

The next field is the subtotal amount to compare against if you are determining shipping cost based on the total sum of money needed to purchase what is in the cart.

The following field after that is the quantity of items to compare against to determine shipping based on quantity.

The next field after quantity is the measured total of items based on the measured field index determined in the cart setup above.

The final field is the amount of money the shipping will be if the criteria is matched in the above fields. If the value is followed by a % symbol, then the value of the shipping will be a percentage of the current subtotal.

- **@sc_discount_logic** is an array containing the logic for applying a discount to the order. The discount is calculated as a dollar amount. Do not make the amounts negative. The Web store automatically subtracts the values in this array from the subtotal when the discount is calculated.
- **$sc_sales_tax** is the value of sales tax. For example, Maryland has a 5 percent sales tax, so this value would be **.05** or (5/100).
- **$sc_sales_tax_form_variable** is the name of a form variable that will be used on the order form to determine if the sales tax is applicable. For example, **05-b_state** could be a form variable that would determine whether the customer needs to have sales tax applied. This variable corresponds to the **%sc_order_form_array** form field names.
- **@sc_sales_tax_form_value** are the possible, case insensitive values that the form variable above (**$sc_sales_tax_form_variable**) should be equal to in order to apply sales tax. For the Maryland sales tax example, this would be an array containing **md** and **maryland**.
- **$sc_order_email** is the email address to send orders to. Do not forget to "escape" the @ symbols with a backslash character:

  ```
  "you@yourdomain.com"
  ```

  must be written as:

  ```
  "you\@yourdomain.com"
  ```

- **$sc_send_order_to_email** should be set equal to **yes** if you want orders sent to the above email address.
- **$sc_send_order_to_log** should be set equal to **yes** if you want the orders to be recorded in a local log file.
- **$sc_order_log_file** is the path and filename of the logfile where you want orders recorded if the above variable is **yes**.
- **$sc_order_check_db** should be set equal to **yes** if you want to use the database routines to double-check that the user has not attempted to fool around with the database by entering in values for items based on form manipulation. The Web store script double-checks to see if the price in the cart for the item being ordered is the same as the recorded price in the database file.

- **$sc_use_pgp** should be set equal to**yes** if you want to use the PGP library to communicate with PGP for encrypting orders. You must have previously installed PGP on your system and set up your public/private key pairs. This process will be discussed in more detail in Chapter 9.
- **$sc_pgp_temp_file_path** is the path where you want the PGP program to generate temporary files. This should be a directory that is writable to the Web server.

The following is a list of example values for the variables that have been discussed above:

```
$sc_mail_lib_path = "./Library/mail-lib.pl";
$sc_order_lib_path = "./Library/web_store_order_lib.pl";
$sc_pgp_lib_path = "./Library/pgp-lib.pl";
$sc_order_lib_path = "./Library/web_store_order_lib.pl";
$sc_order_script_url = "web_store.cgi";

%sc_order_form_array =('01-name', 'Name',
     '02-b_street_address', 'Billing Address Street',
     '03-b_city', 'Billing Address City',
     '04-b_state', 'Billing Address State',
     '05-b_zip', 'Billing Address Zip',
     '06-b_country', 'Billing Address Country',
     '07-m_street_adress', 'Mailing Address Street',
     '08-m_city', 'Mailing Address City',
     '09-m_state', 'Mailing Address State',
     '10-m_zip', 'Mailing Address Zip',
     '11-m_country', 'Mailing Address Country',
     '12-phone', 'Phone Number',
     '13-fax', 'Fax Number',
     '14-e-mail', 'Email',
     '15-URL', 'URL',
     '16-link', 'Link',
     '17-type_of_card', 'Type of Card',
     '18-cardname', 'Name Appearing on Card',
     '19-card_number', 'Card Number',
     '20-ex_date', 'Card Expiration',
     '22-shipping', 'Shipping Method');

@sc_order_form_required_fields =
    ("01-name",
     "02-b_street_address",
```

```
     "03-b_city",
     "04-b_state",
     "05-b_zip",
     "12-phone",
     "14-e-mail");

$sc_order_with_hidden_fields = "yes";

$sc_calculate_discount_at_display_form = 1;
$sc_calculate_discount_at_process_form = 1;

$sc_calculate_shipping_at_display_form = 0;
$sc_calculate_shipping_at_process_form = 1;

$sc_calculate_sales_tax_at_display_form = 1;
$sc_calculate_sales_tax_at_process_form = 1;

@sc_order_form_shipping_related_fields =
    ("22-shipping");

@sc_order_form_discount_related_fields =
    ();

@sc_shipping_logic =
     ("ups||1-10||5",
      "ups||11-||10",
      "fedex||1-10||20",
      "fedex||11-||30");

@sc_discount_logic  = ("|1-||1");

$sc_sales_tax = ".05"; # 5%
$sc_sales_tax_form_variable = "04-b_state";
@sc_sales_tax_form_values = ("md", "Maryland");

$sc_order_email = "you\@yourdomain.com";
$sc_send_order_to_email = "yes";
$sc_send_order_to_log = "no";
$sc_order_log_file = "./Admin_files/order.log";
$sc_order_check_db = "yes";

$sc_use_pgp = "no";
$sc_pgp_temp_file_path = "./Admin_files";
```

Defining Shipping, Discount, and Sales Tax Calculation Order

The Web store shipping costs, discounts, and sales tax can be calculated either when the order form is displayed or after the order form has been submitted by the customer. All three types of calculations have the flexibility of being configured to rely on what the customer types into the order form.

For example, some Web stores may be set up such that the shipping is always done through UPS with rigid logic. Other Web stores may allow the user to choose the type of shipping on the order form itself. Allowing the user to choose the type of shipping will generally affect the cost. Thus, each of these calculations also have the ability to be configured such that they only calculate after the order form has been submitted. In addition, each calculation has the flexibility of being able to be calculated in conjunction with, before, or after any other calculation is applied to the subtotal. All these options are discussed below.

The key to specifying the calculations that are performed lies in the following variables:

```
$sc_calculate_discount_at_display_form
$sc_calculate_discount_at_process_form
$sc_calculate_shipping_at_display_form
$sc_calculate_shipping_at_process_form
$sc_calculate_sales_tax_at_display_form
$sc_calculate_sales_tax_at_process_form
```

These variables are numeric variables valued from 0 to 3. If any of them is equal to 0, then the Web store does not calculate that value. Each type of value (shipping, discount, and sales tax) has two variables: **at_display_form** and **at_process_form**. The **at_display_form** variable corresponds to whether the calculation occurs at the display of the order form to the user. The **at_process_form** variables corresponds to whether the calculation is performed after the order form has been submitted (the order form is being processed).

The order of the calculation is determined by the value of the variables (1 to 3). The following is an illustration of the basic algorithm.

1. *Stage One.* All three variables are checked to see if they are equal to 1. If any variable is equal to 1, then the current subtotal is used to calculate their appropriate values. At the end of all the calculations, the subtotal is adjusted to reflect the calculations done at stage 1. The fact that all the calculations are done before the subtotal value is affected is very important. If you wish the calculations to be applied to the subtotal before the next calculation is performed, you merely need to make sure that the values of the variables are not equal. An example will be given below.
2. *Stage Two.* All three variables are checked to see if they are equal to 2. If any variable is equal to 2, then the current subtotal after Stage One is used to calculate the appropriate values. At the end of all the Stage Two calculations, the subtotal is adjusted to reflect these calculations.
3. *Stage Three.* Finally, all three variables are checked to see if they are equal to 3, if any variable is equal to 3, then the current subtotal after Stage Two is used to calculate the appropriate values. At the end of all the Stage Three calculations, the subtotal is adjusted to reflect these calculations.

The following are examples of the application of this algorithm and how you would set up the above variables to accomplish your goal. Various different scenarios are presented.

All Calculations Done at the Same Time

If all calculations are to be performed at the same time, you need only set up the variables so that all the numbers are equal to 1:

```
$sc_calculate_discount_at_display_form = 1;
$sc_calculate_discount_at_process_form = 1;
$sc_calculate_shipping_at_display_form = 1;
$sc_calculate_shipping_at_process_form = 1;
$sc_calculate_sales_tax_at_display_form = 1;
$sc_calculate_sales_tax_at_process_form = 1;
```

If you follow the algorithm above, only Step One will actually calculate any values since all the variables are equal to 1. Furthermore, all the values (discount, shipping, and sales tax) will all be calculated based on the Stage One subtotal. Only after all these values have been calculated independent of each other will the subtotal be changed to reflect the new values.

Thus, if the subtotal of all the items in an order was $10.00, and the shipping came out to $1.00, a discount came to $2.00, and the sales tax of 5 percent came to 50 cents, then the new subtotal is $9.50 after all the calculations are applied to the subtotal after Step One of the algorithm.

Calculations Done at Different Times

If the shipping is to be performed before sales tax which is to be performed before the discount is applied, then the relative variables should be set to **1**, **2**, and **3**, respectively. An example of these values appears below.

```
$sc_calculate_discount_at_display_form = 3;
$sc_calculate_discount_at_process_form = 3;
$sc_calculate_shipping_at_display_form = 1;
$sc_calculate_shipping_at_process_form = 1;
$sc_calculate_sales_tax_at_display_form = 2;
$sc_calculate_sales_tax_at_process_form = 2;
```

If you follow the three steps in the calculation algorithm, then shipping will be calculated at Stage One. Then, the subtotal will have shipping added to it. Next, at Stage Two, the sales tax will be calculated based on the subtotal plus the shipping from the end of Stage One. Then, the subtotal will have the sales tax added to it. At Stage Three, the discount will be calculated based off of the subtotal from Stage Two (with the added sales tax and shipping). Finally, the subtotal from Stage Two will have the discount subtracted from it. This will be the final subtotal.

Thus, if the subtotal for all the items was $10.00 and the shipping was $1.00, the new subtotal after Stage One is $11.00. Then, if a 5 percent sales tax is applied to the new $11.00 subtotal at Stage Two, a sales tax of 55 cents will be calculated and added to the subtotal. This results in a subtotal of $11.55. Finally, if the discount came to $2.00 at Stage Three, the new subtotal will be $9.55 after the discount is applied.

Notice how a slight change in the logic adjusts the final result subtotal from when we calculated all the calculations at once. Since, this time around, we are calculating the sales tax after the shipping was applied, the sales tax is being calculated on the basis of the larger value of the $11.00 subtotal instead of the $10.00 subtotal in the previous scenario.

Some Calculations Done After Submitting the Order Form

Sometimes you will want to produce logic where certain calculations are performed only after the order form has been submitted. For example, many online stores calculate shipping differently depending on whether the user chooses UPS, FedEx, or some other shipping method at the level of the order form. Since it is obvious that shipping depends on an order form variable, in this case we cannot calculate shipping at this time. In addition, the sales tax may depend on what the user has typed into the order form as their state. A person who is out of state should not have state sales tax applied.

The first example where everything was calculated at once would be affected by making the shipping and sales tax variables at the display of the order form equal to 0. The discount will still be calculated just like before. An example of these variables appears below:

```
$sc_calculate_discount_at_display_form = 1;
$sc_calculate_sales_tax_at_display_form = 0;
$sc_calculate_sales_tax_at_process_form = 1;
```

The case of the second example where the sales tax is calculated after the shipping and then the discount is applied after the shipping and sales tax is more interesting. In this case, although discount does not directly rely on a form variable, we do not want to calculate it at the display of the order form since the discount is clearly dependent on the resulting subtotals after shipping and sales tax have been applied. However, if sales tax and shipping cannot be applied at the display of the order form, then we can not apply the discount here either. The following is a list of what the variables would be given this new twist on the scenario:

```
$sc_calculate_discount_at_display_form = 0;
$sc_calculate_discount_at_process_form = 3;
$sc_calculate_shipping_at_display_form = 0;
$sc_calculate_shipping_at_process_form = 1;
$sc_calculate_sales_tax_at_display_form = 0;
$sc_calculate_sales_tax_at_process_form = 2;
```

Defining Shipping and Discount Logic

Shipping and discount logic are defined in exactly same way except that the value calculated from the discount logic is subtracted, rather than added, to the current subtotal. Thus, our examples of how we define shipping and discount logic will focus on defining the shipping logic variables.

Logic Where Shipping Is Independent of Any Form Variable

If shipping is independent of any form variable value from the order form, then **@sc_order_form_shipping_related_fields** is set equal to an empty list. In the example below, **@sc_shipping_logic** is set so that the shipping is always equal to $5.00:

```
@sc_order_form_shipping_related_fields = ();
@sc_shipping_logic = ("|1-||5");
```

Recall that **@sc_shipping_logic** is an array of elements that correspond to logic performed to get the final shipping value. Each logical element is a pipe-delimited list that determines whether the logic is satisfied and the amount of shipping to be applied if that logic is satisfied. The first elements of the pipe-delimited list correspond to the **@sc_order_form_shipping_related_fields** array. If there are no elements in that array, then the first element of the list is the subtotal to compare against, the second element is the quantity, and the third field is the measured quantity to compare against. The last field is the actual shipping cost to apply if the previous fields are matched.

In the above example, the first field is empty so there is no match done on the subtotal. The second field consists of **1-**, which is an open-ended range. This means that quantities of 1 or greater have this shipping logic applied to them. Finally, since the third field is empty so there is no measured quantity to compare against. Since the quantity of **1-** applies to all orders with any items, the shipping will always be $5.00 (the value of the last field).

Logic Where Shipping Is a Percentage of the Subtotal

In this example, the shipping is calculated the same way as in the example above. Except that instead of a flat amount for shipping ($5.00), we calculate the shipping as a percentage of the subtotal. This is done by simply adding a % symbol to the shipping amount in the field. An example of applying a 10 percent shipping charge is given below:

```
@sc_order_form_shipping_related_fields = ();
@sc_shipping_logic = ("|1-||10%");
```

Notice that the value to apply is listed as the number **10** followed by a **%** sign. The % sign tells the Web store to apply the shipping as a percentage rather than a whole shipping amount.

Logic Where Shipping Is Dependent on Shipping Type

When the shipping logic is dependent on the shipping type that the user selected in the order form, we need to set the **@sc_order_form_shipping_related_fields** to show that this form field needs to be compared against. In the example below, the array is set to the form variable field name **22-shipping**. In this case, the first pipe-delimited field in each element of the **@sc_shipping_logic** array now gets compared against this form variable:

```
@sc_order_form_shipping_related_fields =  ("22-shipping");
@sc_shipping_logic =  (      "ups|1-10|||5",
"ups|11-19|||10",
"ups|20-|||12",
"fedex|1-10|||7",
"fedex|11-19|||14",
"fedex|20-|||21"      );
```

In the preceding example, if the value filled into **22-shipping** form field is **ups**, then the first three elements of **@sc_shipping_logic** are compared. If the value filled into **22-shipping** is **fedex**, then the last three values of

@sc_shipping_logic will be examined further. Since the first field after the shipping type field has values to compare against this time instead of the second field, this means that we are comparing subtotal amounts instead of quantities. The measured amount is also left blank.

Based on the above variable settings, there are several examples to examine. First, if the **22-shipping** field is **ups** and the subtotal is $15.00, then we match the element that says **ups|11-19|||10**. This means that the shipping would be calculated as $10.00. If the **22-shipping** field is fedex and the subtotal is $50.00, then the shipping would become $21.00 since this matches **fedex|20-||||21**.

Logic Where Shipping Is Dependent on Shipping Type and Zip Code

This example is just like the previous one except that the shipping will be dependent on the zip-code field that the user enters as well as the shipping type field. Notice below that the **@sc_order_form_shipping_related_fields** has the zip code added to it. In addition, the zip code range to match against is now added to the pipe-delimited logical elements in the **@sc_shipping_logic array**. The shipping logic is no longer dependent on the subtotal amount in this example. Notice that this is indicated by leaving the three fields after the shipping type and zip code empty—since the fields after the **@sc_order_form_shipping_related_fields** correspond to subtotal, quantity, and measured-quantity comparisons.

```
@sc_order_form_shipping_related_fields =
("22-shipping", "05-b_zip");
@sc_shipping_logic =  (      "ups|-10000||||5",
"ups|10001-20000||||10",
"ups|20001-||||12",
"fedex|-10000||||7",
"fedex|10001-20000||||14",
"fedex|20001-||||21"  );
```

If the zip code is 09000 and the shipping type is **ups**, then the shipping will be $5.00 since this matches the element that has a shipping type of **ups** and open ended range for the zip code **-10000**. If the zip code is 20855, and the shipping

type is **fedex**, then the shipping will be $21.00 since this matches element with a shipping type of **fedex** and the open-ended zip code range **20001-**.

Logic Where Shipping Is Dependent on Measured Quantity

This example has no dependency on form variables. Instead, the shipping costs will be dependent on the weight of what was ordered. For this example, assume that **$sc_cart_index_of_measured_value** is set equal to a field in the cart that corresponds to the weight of the items being ordered. Also, assume that the weight is measured in kilograms. It really does not matter what the weight field unit is, as long as it is consistent throughout the catalog. When this field is set, the Web store automatically keeps track of this field and keeps a running subtotal of it. Since this example does not depend on any order form variables, **@sc_order_form_shipping_related_fields** consists of an empty list (nothing). The **@sc_shipping_logic** array consists of shipping prices for the various weights. The third field is the measured quantity (weight). The first two fields (Subtotal and Total quantity) are left blank because they will not be affecting the shipping cost in this example. The code for the Measured quantity example appears below:

```
@sc_order_form_shipping_related_fields = ();
@sc_shipping_logic =  (      "||-10|1",
"||11-20|2",
"||21-30|3",
"||31-50|4",
"||51-75|5",
"||76-|20");
```

Thus, if the weight is **5 kg**, the shipping will be $1.00 because the open-ended comparison (**-10**) corresponds to **5 kg**. If the weight is **40 kg**, then the shipping will be $4.00 since **40 kg** falls in the comparison range **31-50**.

Shipping and Discount Logic Summary

@sc_order_form_shipping_related_fields and the **@sc_shipping_logic** Setup variables are key to determining how shipping is calculated. Flat-rate shipping can be indicated by simply having one element in **@sc_shipping_logic**, which is not dependent on any form variable. Complex shipping tables including zip codes can also be represented easily by adding form variables to the

@sc_order_form_shipping_related_fields array and then representing their different values and options in the **@sc_shipping_logic** array elements.

The same flexibility of logic that can be applied to calculating shipping can also be applied to discounts. The **@sc_order_form_discount_related_fields** and **@sc_discount_logic** arrays correspond exactly to the arrays related to shipping logic. All the rules that apply to configuring shipping logic apply to configuring discount logic as well.

Setting Up Mail-lib.pl

Setting up **mail-lib.pl** for emailing nonencrypted orders is very simple. The only variables that need to be set are **$sc_mail_lib_path**,which should be set to the path and filename of the mail library you are using, **$sc_send_order_to_email** should be **yes**, and **$sc_order_email** should be set to your email address. Note that **$sc_order_email** should have a backslash (\) appear before the @ symbol.

The **mail-lib.pl** distributed with the system by default is UNIX-specific. It uses the UNIX **sendmail** command located at **/usr/lib/sendmail** by default. If you find your UNIX-based Web store is not sending email, you may need to change the path of **sendmail** in the **mail-lib.pl** file. At line 42, the variable to change is **$mail_program** in this file.

Windows NT and Windows 95 Server Considerations

If you are using a Windows NT–based Web server, you should be using **smtp-mail-lib.pl**. To use this version of the library instead of the sendmail version, copy the **smtpmail-lib.pl** file over the **mail-lib.pl** that is distributed with the UNIX version of the script. Then, edit the **mail-lib.pl** file so that **$mail_os** is set equal to **NT**. If you are using Windows 95, you will need to do an additional adjustment to the source code so that the line that reads:

```
$SMTP_PORT = (getservbyname('smtp','tcp'))[2];
```

is changed to

```
$SMTP_PORT = 25;
```

As of this writing, the Windows 95 version of Perl has trouble with the **getservbyname** socket call.

In addition, if you are using the SMTP version of the **mail-lib.pl** file, then you will need to make sure that the email address that you are sending orders to is the direct email host that is processing the SMTP mail.

Very large sites typically give you an email address that fits the large domain name such as **you@yourdomain.com**. However, in a large domain, there may be too many email addresses for any one server to handle all the Internet traffic. Therefore, the **yourdomain.com** part of the email address is usually just an alias for another machine that does the actual mail handling. This machine name is usually more specific than **yourdomain.com**, such as **mailserver.yourdomain.com**. Figure 8.2 shows an example of how it is possible that the server that actually processes the mail may not always be the hostname that you are sending mail to.

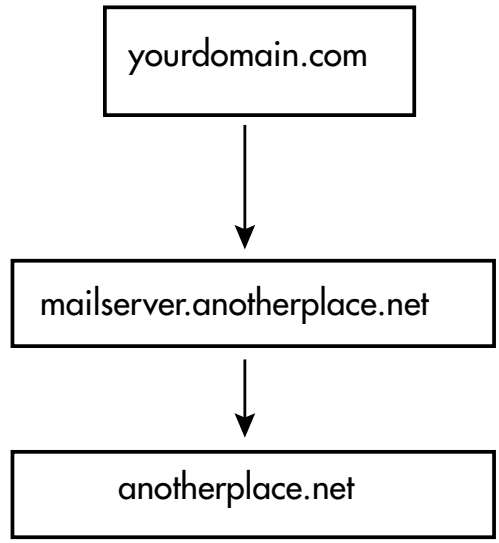

Figure 8.2 Emailing to someone@anotherplace.net does not always go directly to anotherplace.net.

It is recommended that you first try to use your normal email address. If it does not work, then contact your Internet service provider and ask if they are using another machine to actually process email.