This chapter unpacks the business-logic portion of Currawong Accounting. Most of its software modules are very simple—little more than bridges between hypertext transport protocol (HTTP) requests (of the GET and POST varieties) and simple object access protocol (SOAP) queries to the accessor layer. A few of its elements are quite exciting, though. These have to do with taking raw business data and converting it to graphical form.

8.1 Inserting, Updating, and Deleting

In insert, update, and delete operations, in which activity on the presentation layer, usually in the form of a human user manipulating an hypertext markup language (HTML) form in some way, the business logic layer acts as an intermediary between the presentation layer and the accessor layer. The programs of the business logic layer receive HTTP GET or POST requests from the presentation layer, and in this implementation do little more than repackage the values received as the elements of a parameters array that’s transmitted to the relevant Web service on the accessor layer.

Because the insert, update, and delete modules on the business logic layer are so similar, only a single typical example of each is shown in this chapter.

8.1.1 Inserting a Row

Several interface pages on the presentation layer are concerned with adding rows to the Currawong Accounting database. For example, the enterBankAccount.php program (covered in depth in Chapter 9) presents the user with an HTML form, which is meant to be
filled with the details of a bank account. This form ultimately gets submitted according to this directive:

```html
<Form name='addBankAccount' METHOD='POST' ACTION='http://'.$blHost.'/acct/bl/blEnterBankAccount.php'>
```

That means that the contents of the form are submitted by HTTP POST to a program called `blEnterBankAccount.php` on the business logic layer. Here's what that file looks like:

```php
blEnterAccount.php

require_once('nusoap-0.6/nusoap.php');
require_once('configuration.php');

// Convert $_POST array to $parameters for clarity.
foreach ($_POST as $key => $value)
{
    $parameters[$key] = $value;
}

// BL-level processing of submission could go here.
// Establish NuSOAP soapclient object and make call to Web Service

$soapclient = new soapclient("http://" . $accessorHost . "/acct/accessor/addAccount.php");

$result = $soapclient->call('addAccount',$parameters);

// Refresh the page to show updated data.

$url = "http://" . $presentationHost . "/acct/presentation/enterAccount.php";
$locationHeader = "Location: " . $url;

header($locationHeader);
```

The core of this program is a loop that examines $_POST, which contains the submitted form's contents:

```php
// Convert $_POST array to $parameters for clarity.

foreach ($_POST as $key => $value)
{
    ...
```
$parameters[$key] = $value;

}

The entire contents of the $_POST array are transferred to a new array called $parameters. It's true that a direct assignment could have accomplished the same thing, and even that $_POST itself could be submitted to the accessor layer (as shown shortly), but this approach makes the assignment process more obvious, and there's a clear place to put further processing logic if some is needed in the future.

With $parameters defined, a call is made to the accessor layer:

$soapclient = new soapclient("http://" . $accessorHost . "/acct/accessor/addAccount.php");

$result = $soapclient->call('addAccount',$parameters);

Finally, the page is made to refresh, via a manipulation of the local page's HTTP header:

// Refresh the page to show updated data.

$URL = "http://" . $presentationHost . "/acct/presentation/enterAccount.php";

$locationHeader = "Location: " . $URL;
header($locationHeader);

Other modules on the business-logic layer that are functionally similar to this one are:

- blEnterAccount.php,
- blEnterAcctType.php,
- blEnterCurrency.php,
- blEnterInstitution.php,
- blEnterPayee.php,
- blEnterTransaction.php, and
- blEnterTransType.php.

8.1.2 Updating a Row

The logic involved in updating a row is very similar to that of inserting a row, at least at the business-logic layer (significant differences exist at the presentation layer, and especially at the accessor layer). The incoming $_POST array becomes the input for an update program on the accessor layer. The program blUpdateAccount.php, used for modifying the characteristics of accounting categories, is typical of the “blUpdate” modules.
blUpdateAccount.php

require_once('nusoap-0.6/nusoap.php');
require_once('configuration.php');

$parameters = array();

// Convert $_POST array to $parameters for clarity.
foreach ($_POST as $key => $value)
    {
        $parameters[$key] = $value;
    }

// BL-level processing of submission could go here.
// Establish NuSOAP soapclient object and make call to Web Service

$soapclient = new soapclient("http://" . $accessorHost . "/acct/accessor/updateAccount.php");

$result = $soapclient->call('updateAccount', $parameters);

// Refresh the page to show updated data.

$URL = "http://" . $blHost . "/acct/bl/reloadAndClose.html";

$locationHeader = "Location: " . $URL;

header($locationHeader);

The most significant difference here is in the “page refresh” functionality at the end, shown here again:

// Refresh the page to show updated data.

$URL = "http://" . $blHost . "/acct/bl/reloadAndClose.html";

$locationHeader = "Location: " . $URL;

header($locationHeader);

The page that's loaded is reloadAndClose.html. That's because the pieces of the presentation layer related to update activity run in a window that's separate from the main window. Let's have a look at reloadAndClose.html.
The functionality here is implemented entirely with JavaScript. When the page loads, the onLoad event handler (noted in the opening BODY tag) fires. In doing so, it calls the reloadAndClose() function. It uses the document object model (DOM) to refer to the location of the window that opened the current window (opener.location) and invoke its reload() method. It then invokes window.close(), thus closing its own window.

Other modules on the business-logic layer that resemble blUpdateAccount.php are:

- blUpdateAcctType.php,
- blUpdateBankAccount.php,
- blUpdateCurrency.php,
- blUpdateInstitution.php,
- blUpdatePayee.php,
8.1.3 Deleting a Row

At the business-logic layer, the deletion of a row is very similar to the insertion or modification of a row. Once again, the $_POST array is repackaged as an array of parameters for a module on the accessor layer.

There's only one business-logic-layer program concerned with deletions. That is `blDeleteTransaction.php`.

```php
require_once('nusoap-0.6/nusoap.php');
require_once('configuration.php');

$parameters=array();
// Convert $_POST array to $parameters for clarity.
foreach ($_POST as $key => $value)
{
    $parameters[$key] = $value;
}
// BL-level processing of submission could go here.

// Establish NuSOAP soapclient object and make call to Web Service
$soapclient = new soapclient("http://" . $accessorHost . "/acct/accessor/deleteTransaction.php");

$result = $soapclient->call('deleteTransaction', $parameters);
// Refresh the page to show updated data.

$url = "http://" . $blHost . "/acct/bl/reloadAndClose.html";
$locationHeader = "Location: " . $url;
header($locationHeader);
```
This program is functionally identical to those concerned with update capabilities, right down to the use of reloadAndClose.html (discussed in the previous section) for refreshing the parent frame.

8.2  Reporting

The most important functions of the business-logic layer in Currawong Accounting are the reporting functions. These enable the user to view summary information about the managed bank accounts, and therefore have a better idea of what is happening with his or her money.

The reporting capabilities discussed here comprise a tabular summary page that displays the contents of any group of specified accounts, as well as a series of graphs. There are three graphs:

1. A bar graph showing the value of a specified account in its native currency, with weekly sampling.
2. A similar bar graph with a supplementary line depicting the 60-day moving average of the account's balance.
3. A bar graph showing, simultaneously, the values of several accounts, with all values shown in U.S. dollars for consistency. This graph also employs weekly sampling.

The graphing modules make use of a bar- and line-graph generating class written by Herman Veluwenkamp of New Zealand and released under the lesser GNU public license (LGPL), which means we're free to make use of it as a library here. That's why the graphing modules import graph.php—Veluwenkamp's class—and devote code to manipulating an instance of it.

8.2.1  Generating an Accounts Summary

Currawong Accounting needs a summary page that shows the individual values of all existing accounts. The page, generated by blAccountsReport.php, takes no input from the presentation layer. It always displays information about all accounts.

blAccountsReport.php

<html>
<head>
<title>View Transactions</title>
</head>
<body>
<H1>Summary of Accounts</H1>

require_once('nusoap-0.6/nusoap.php');
require_once('configuration.php');

// Get list of bank accounts
$parameters = array();
$soapclient = new soapclient('http://' . $accessorHost . '/acct/accessor/getBankAccounts.php');
$result = $soapclient->call('getBankAccounts',$parameters);

// Get today's date.
$today = date("Y-m-d");

$totalUSDBalance = 0;

// Generate a table containing accounts and their balances.

echo '<TABLE BORDER="1" CELLPADDING="5">';
echo '<TR>';
    echo '<TD>'.'<B>Description</B>'.';</TD>';
    echo '<TD>'.'<B>Institution</B>'.';</TD>';
    echo '<TD>'.'<B>Number</B>'.';</TD>';
    echo '<TD>'.'<B>Currency</B>'.';</TD>';
    echo '<TD>'.'<B>Native Balance</B>'.';</TD>';
    echo '<TD>'.'<B>USD Balance</B>'.';</TD>';
echo '</TR>';
}
echo '</TD>';
echo '</TR>';

foreach ($result as $key => $subarray)
{
    echo '<TR>';
    echo '<TD>';
    echo $subarray['description'];
    echo '</TD>';
    echo '<TD>';
    echo $subarray['institution'];
    echo '</TD>';
    echo '<TD>';
    echo $subarray['number'];
    echo '</TD>';
    echo '<TD>';
    echo $subarray['currency'];
    echo '</TD>';
    // For each account, get balance as of today.
    echo '<TD>';
    $parameters = array('id'=>$subarray['id'], 'date'=>$today);
    $soapclient = new soapclient('http://' . $accessorHost .
    '/acct/accessor/getSpecifiedBankAccountBalance.php');
    $result2 = $soapclient->call('getSpecifiedBankAccountBalance',
    $parameters);
    echo round(($result2[0]['balance'] + 0),2);
    echo '</TD>';
    // For each account, get relevant currency xRate -- the latest in the
    // application's currency table.
    echo '<TD>';
    $parameters = array();
    $parameters['id'] = $subarray['currencyId'];
    $soapclient = new soapclient('http://' . $accessorHost .
    '/acct/accessor/getSpecifiedCurrency.php');
    $result = $soapclient->call('getSpecifiedCurrency',
    $parameters);
    $xRate = $result[0]['xRate'];
The program begins its operation by retrieving all details of all bank accounts:

// Get list of bank accounts

$parameters = array();
$soapclient = new soapclient('http://'. $accessorHost .
'/acct/accessor/getBankAccounts.php');
$result = $soapclient->call('getBankAccounts', $parameters);

The results of this call to the accessor layer are then used to populate a table. Some of the values require further queries and some manipulation, though. To get the balance of a given account, for example, the program must make another query to the accessor layer:

$parameters = array('id' => $subarray['id'],
'date' => $today);
$soapclient = new soapclient('http://'. $accessorHost .
'/acct/accessor/getSpecifiedBankAccountBalance.php');
$result2 = $soapclient->call('getSpecifiedBankAccountBalance', $parameters);
$usdRate = $result2[0]['balance'];
$usdBalance = $usdRate * $result2[0][balance];
echo round($usdBalance,2);
echo '<TD>';
echo '</TR>';

// Add to overall USD (converted balance) for later display.

$totalUSDBalance += $usdBalance;
}

// Add to overall USD (converted balance) for later display.

$totalUSDBalance += $usdBalance;
}

The program begins its operation by retrieving all details of all bank accounts:

// Get list of bank accounts

$parameters = array();
$soapclient = new soapclient('http://'. $accessorHost .
'/acct/accessor/getBankAccounts.php');
$result = $soapclient->call('getBankAccounts', $parameters);

The results of this call to the accessor layer are then used to populate a table. Some of the values require further queries and some manipulation, though. To get the balance of a given account, for example, the program must make another query to the accessor layer:

$parameters = array('id' => $subarray['id'],
'date' => $today);
$soapclient = new soapclient('http://'. $accessorHost .
'/acct/accessor/getSpecifiedBankAccountBalance.php');
$result2 = $soapclient->call('getSpecifiedBankAccountBalance', $parameters);
$usdRate = $result2[0]['balance'];
$usdBalance = $usdRate * $result2[0][balance];
echo round($usdBalance,2);
That yields the account's balance in its native currency. Notice that 0 is added to the balance figure so that something is displayed in the event that the balance comes back null, and that round() is used to make sure that two decimal places are displayed.

Further queries yield the relative value of the account's native currency to the U.S. dollar:

```php
$parameters = array();
$parameters['id'] = $subarray['currencyId'];
$soapclient = new soapclient('http://'. $accessorHost . '/acct/accessor/getSpecifiedCurrency.php');
$result = $soapclient->call('getSpecifiedCurrency', $parameters);
$xRate = $result[0]['xRate'];
```

This is then used to convert the account's balance to U.S. dollar terms:

```php
$usdBalance = $xRate * $result2[0]['balance'];
```

In the foreach loop, the program accumulates a running total U.S. dollar value for the account, which is then displayed at the bottom of the information table. A date is rendered, as well:

```php
echo date("d F Y H:i T");
```

The letters provided as an argument dictate how the date is rendered (Figure 8.1 shows a depiction and see the date() function documentation for other options). Figure 8.1 shows the accounts summary page.

### 8.2.2 Graphing the Balance of a Single Account over Time

In order to provide Currawong Accounting users with a bar graph that shows the balance of a specified account every seven days for a designated period of time, `blBarGraphAccountWeekly.php` exists. It takes input from `viewBarGraphSingleAccount-Weekly.php` on the presentation layer. Specifically, it receives three values:

- **account.** An integer corresponding to the id value of a row in the `ACCT_bank_account` table.
- **startDate.** A date, in YYYY-MM-DD form (the MySQL standard form) that represents the first day in the period to be graphed.
- **endDate.** A date, in YYYY-MM-DD form (the MySQL standard form) that represents the last day in the period to be graphed.

A bit of background about dates: MySQL date columns (such as the one that appears in the `ACCT_bank_account` table) use the YYYY-MM-DD format. However, it's hard to do calculations on dates in that form. It's much easier to do math on dates in the Unix timestamp form, in which a date is represented as a number of seconds since midnight on
January 1, 1970. Because this program (and the other graphing programs in the business-logic layer) need to do date math, we need some utility functions for converting back and forth between Unix timestamp format and MySQL format. Those functions appear in dateConv.php.

```php
function mysql_datetime_to_timestamp($dt) {
    $yr=strval(substr($dt,0,4));
    $mo=strval(substr($dt,5,2));
    $da=strval(substr($dt,8,2));
    $hr=strval(substr($dt,11,2));
    $mi=strval(substr($dt,14,2));
    $se=strval(substr($dt,17,2));

    return mktime($hr,$mi,$se,$mo,$da,$yr);
}

function timestamp_to_mysql_date($ts) {
    $d=getdate($ts);
```
It's all done with substring and date-manipulation functions, but the key thing to know is that `mysql_datetime_to_timestamp()` makes one conversion, and `timeStamp_to_mysql_date()` goes the other way. We'll need both functions to make the graph we want.

With that background out of the way, we can examine `blBarGraphAccountWeekly.php` itself.

```
/* 8.2 Reporting 177

     $yr=\$d["year"];
     $mo=\$d["mon"];
     $da=\$d["mday"];

     return sprintf("%04d-%02d-%02d",\$yr,\$mo,\$da);

*/

It's all done with substring and date-manipulation functions, but the key thing to know is that `mysql_datetime_to_timestamp()` makes one conversion, and `timeStamp_to_mysql_date()` goes the other way. We'll need both functions to make the graph we want.

With that background out of the way, we can examine `blBarGraphAccountWeekly.php` itself.

```
include 'graph.php';
require_once('configuration.php');
require_once('nusoap-0.6/nusoap.php');
require_once('dateConv.php');
require_once('rounder.php');
require_once('standardGraphFormat.php');

// Instantiate a graph object.

$graph = new graph(700,480);

// Initialize relevant dates...

$startDate = mysql_datetime_to_timestamp($_POST['startDate']);
$endDate = mysql_datetime_to_timestamp($_POST['endDate']);

$midDate = $startDate;

$xLabelArray = array();
$yDataArray = array();
$minValue = 0;
$maxValue = 0;

$graph->y_order=array();

// For every chosen account, get the current balance...
```
foreach ($_POST['accounts'] as $account)
{
    $arrayOfValues = array();
    while ($midDate < $endDate)
    {
        $formattedMidDate = timestamp_to_mysql_date($midDate);
        $xLabelArray[] = $formattedMidDate;
        $parameters = array('id' => $account, 'date' => $formattedMidDate);
        $soapclient = new soapclient('http://' . $accessorHost . '/acct/accessor/getSpecifiedBankAccountBalance.php');
        $result = $soapclient->call('getSpecifiedBankAccountBalance', $parameters);
        $balance = $result[0]['balance'];

        // Do some range-setting, so the graph is the right size...
        if ($balance < $minValue)
        {
            $minValue = $balance;
        }
        elseif ($balance > $maxValue)
        {
            $maxValue = $balance;
        }

        // Add balance to arrayOfValues (which is to be graphed)
        // Requires round() below, or no sub-zero range in graph.
        $arrayOfValues[] = round($balance);

        // Advance $midDate by seven days for another check
        $midDate += 604800;
    }
}
// Assign arrayOfValues to y_data (an associative array) under the $account key...

$graph->y_data[$account] = $arrayOfValues;

}$graph->y_order = array();

$availableColours = array('maroon','olive','green','navy','purple','gray',
'red','lime','yellow','blue','fuchsia','aqua','black');
$useColour=0;

// For each account being graphed, get the description for the legend, and associate a color.

foreach ($_POST['accounts'] as $account)
{
$parameters2 = array();
$parameters2['id'] = $account;
$soapclient2 = new soapclient('http://' . $accessorHost .
'/acct/accessor/getSpecifiedBankAccount.php');
$result2 = $soapclient2->call('getSpecifiedBankAccount',
$parameters2);

$accountName = $result2[0]['description'];
$accountCurrency = $result2[0]['currency'];

$graph->y_format[$account] = array('colour' =>
$availableColours[$useColour], 'bar' => 'fill', 'legend' =>
"$accountName ($accountCurrency)");
$graph->y_order[] = $account;
$useColour++;
}// Set the size and granularity of the graph...

$minValue = round($minValue*1.5);
$maxValue = round($maxValue*1.5);

$minValue = abs($minValue);
$minValue = round_to_nearest($minValue, 50);
$minValue = $minValue * -1;
$maxValue = round_to_nearest($maxValue, 50);

// ************************

// Do final formatting and draw the graph...

formatGraph();
$graph->parameter['title'] = 'Weekly Balance of Account (in local currency)';
$graph->parameter['y_label'] = 'Balance';

$graph->x_data = $xLabelArray;

$graph->parameter['y_min_left'] = $minValue;
$graph->parameter['y_max_left'] = $maxValue;

$graph->draw();

The program begins with a long series of variable initializations, among them the instantiation of the graph object (available from the imported graph.php) that will ultimately represent an image file containing the bar graph:

$graph = new graph(700,480);

The business of getting values to graph takes place inside a pair of loops. The outer foreach loop guarantees that we get data for each account specified in the HTTP POST request that kicked off execution:

foreach ($_POST['accounts'] as $account)

And the inner while loop performs the weekly sampling on the balance of that account:

while ($midDate < $endDate)

Bear in mind that there should be only one account in the $_POST array in this case, and that this program is designed to be adaptable to multiaccount applications covered later in this chapter.

Within the inner while loop, we add a label to $xLabelArray, which contains the dates that appear along the x-axis:

$xLabelArray[] = $formattedMidDate;

We also make a call to the accessor layer to get the balance for the current account on the date specified by $formattedMidDate:

$parameters = array('id' => $account, 'date' => $formattedMidDate);
$soapclient = new soapclient('http://'. $accessorHost .

A bit of ranging comes next, to make sure that the y-axis is high and low enough to accommodate the most extreme balance values (positive and negative):

```php
if ($balance < $minValue)
{
    $minValue = $balance;
}

elseif ($balance > $maxValue)
{
    $maxValue = $balance;
}
```

Then, arrayOfValues has the balance appended to it as a new element:

```php
$arrayOfValues[] = round($balance);
```

The variable $midDate is then advanced by 684,800, which equals 60*60*24*7, or the number of seconds in a week:

```php
$midDate += 604800;
```

Before leaving the loop, the two-dimensional associative array $y_data gets a new element, which has the current account id number as its key and $arrayOfValues as its value:

```php
$graph->y_data[$account] = $arrayOfValues;
```

The process repeats for all accounts contained in the $_POST array.

The program then loops through the specified accounts one more time to populate the y_format property of the $graph object with details about the color in which each account's bars are to be rendered. Note that there's a call to getSpecifiedBankAccount.php on the accessor layer to convert the account's id number to a name (description, actually) and get its native currency for the graph legend, as well.

```php
foreach ($_POST['accounts'] as $account)
{
    $parameters2 = array();
    $parameters2['id'] = $account;
    $soapclient2 = new soapclient('http://'. $accessorHost . '/acct/accessor/getSpecifiedBankAccount.php');
    $result2 = $soapclient2->call('getSpecifiedBankAccount', $parameters2);
```
Figure 8.2: Currawong Accounting’s single-account bar graph performs weekly balance sampling over a specified time range.

```php
$accountName = $result2[0]["description"]; $accountCurrency = $result2[0]["currency"];

$graph->y_format[$account] = array("colour" => $availableColours[$useColour], "bar" => "fill", "legend" => "$accountName ($accountCurrency)"); $graph->y_order[] = $account; $useColour++; }
```

A bit more ranging takes place—the code makes the top and bottom of the y-axis 50 percent greater than each actual extreme, just so the graph looks more comfortable—and the settings are applied to the $graph object with a series of assignments. Finally, there’s a call to $graph->draw(), which causes a PCX graphic file to be generated based on the calculated specifications. Figure 8.2 shows the single-currency bar graph.

8.2.3 Graphing the Balance of a Single Account over Time, with a Moving Average

A bar graph of account values every seven days during a specified time period is useful, but it would be better if there were a line showing the changes in a 60-day moving average
each week. With such a line visible, the user of Currawong Accounting could see whether the balance of the account was, overall, tending to increase or decrease. In blBarGraphSingleAccountWeeklyWithMA.php, some modifications are made to the program discussed in the preceding section to enable the addition of such a trend line.

This program takes its input from viewBarGraphSingleAccountWeeklyWithMA.php on the presentation layer. That program provides three values in an HTTP POST request:

- **account.** An integer corresponding to the id value of a row in the ACCT_bank_account table.
- **startDate.** A date, in YYYY-MM-DD form (the MySQL standard form) that represents the first day in the period to be graphed.
- **endDate.** A date, in YYYY-MM-DD form (the MySQL standard form) that represents the last day in the period to be graphed.

Here's the program that takes those values.

```php
blBarGraphSingleAccountWeeklyWithMA.php

include 'graph.php';
require_once('configuration.php');
require_once('nusoap-0.6/nusoap.php');
require_once('dateConv.php');
require_once('rounder.php');
require_once('standardGraphFormat.php');
// Instantiate a graph object.

$graph = new graph(700,480);

// Initialize relevant dates...

$startDate = mysql_datetime_to_timestamp($_POST['startDate']);
$endDate = mysql_datetime_to_timestamp($_POST['endDate']);

$midDate = $startDate;

$xLabelArray = array();
$yDataArray = array();
$minValue = 0;
$maxValue = 0;

$graph->y_order=array();

// For every chosen account, get the current balance...
```
foreach ($_POST['accounts'] as $account)
{

    $arrayOfValues = array();
    $arrayOfMovingAverageValues = array();

    // ...for each specified $midDate value...

    while ($midDate < $endDate)
    {

        $formattedMidDate = timestamp_to_mysql_date($midDate);

        $xLabelArray[] = $formattedMidDate;

        $parameters = array('id' => $account, 'date' => $formattedMidDate);
        $soapclient = new soapclient('http://' . $accessorHost . '/acct/accessor/getSpecifiedBankAccountBalance.php');
        $result = $soapclient->call('getSpecifiedBankAccountBalance', $parameters);
        $balance = $result[0]['balance'];

        if ($balance < $minValue)
        {
            $minValue = $balance;
        }

        elseif ($balance > $maxValue)
        {
            $maxValue = $balance;
        }

        // Requires round() below, or no sub-zero range in graph.

        $arrayOfValues[] = round($balance);

        // Calculate 60-day moving average as of current $midDate...

        $parameters = array('id' => $account, 'date' => $formattedMidDate, 60);

        $soapclient = new soapclient('http://' . $accessorHost . '/acct/accessor/getAverageBalance.php');
$movingAverage = $soapclient->call('getAverageBalance', $parameters);

// Add moving average value to its own array...
$arrayOfMovingAverageValues[] = round($movingAverage);
$endDate += 604800;
}

// Add each array of account-balance values to the y_data (associative) array, using the account as a key...

$graph->y_data[$account] = $arrayOfValues;

// Add the array of overall moving-average values values to the y_data array, using 'Moving Average' as a key...

$graph->y_data['Moving Average'] = $arrayOfMovingAverageValues;

$graph->y_order = array();

$availableColours = array('maroon','olive','green','navy','purple','gray','red','lime','yellow','blue','fuchsia','aqua','black');
$useColour=0;

// For each graphed account, choose a color and get a description to use in the legend.

foreach ($_POST['accounts'] as $account)
{
    $parameters2 = array();
    $parameters2['id'] = $account;
    $soapclient2 = new soapclient('http://' . $accessorHost . '/acct/accessor/getSpecifiedBankAccount.php');
    $result2 = $soapclient2->call('getSpecifiedBankAccount', $parameters2);
    $accountName = $result2[0]['description'];
    $accountCurrency = $result2[0]['currency'];
The chief innovation of this program is a second call to the accessor layer to get the 60-day moving average for every \$formattedMidDate looped through:

\[
\text{\$parameters = array('id' => \$account, 'date' =>
\$formattedMidDate, 60);}
\]

\[
\text{\$soapclient = new soapclient('http://'. \$accessorHost .}
\]
Figure 8.3: The 60-day moving average line adds a lot to the value of the account-balance graph.

```
'/acct/accessor/getAverageBalance.php');
$movingAverage = $soapclient->call('getAverageBalance', $parameters);
$arrayOfMovingAverageValues[] = round($movingAverage);
```

When the loop is done, $arrayOfMovingAverageValues is another data series that can be formatted for depiction as a line:

```
$graph->y_format['Moving Average'] = array('colour' =>
    $availableColours[$useColour], 'line' => 'brush', 'legend' =>
    "60-Day Moving Average");
```

and graphed with the bar data showing the weekly balances. Figure 8.3 shows the graph with a moving average.

### 8.2.4 Graphing the Balance of Multiple Accounts over Time

The requirement to equip Currawong Accounting with a graphing function that shows a clustered bar graph of all accounts' values in U.S. dollar terms requires another adaptation of the graphing program used twice already. The module blBarGraphMultiAccountWeekly-USD.php has no moving average, but it converts each account's balance to U.S. dollars before adding it to an array of values to be rendered in graphical form.
This program receives input from viewBarGraphMultiAccountWeeklyUSD.php on the presentation layer. It sends two discrete data values and one array:

- **account**: An array of integers corresponding to the id value of a row in the ACCT_bank_account table, representing the accounts to be included in the graph.
- **startDate**: A date, in YYYY-MM-DD form (the MySQL standard form) that represents the first day in the period to be graphed.
- **endDate**: A date, in YYYY-MM-DD form (the MySQL standard form) that represents the last day in the period to be graphed.

A problem with Currawong Accounting in this area is its lack of any historical exchange rate information. As a result, the historical balances depicted by the graphs that result from this program are converted to U.S. dollars based on the latest available exchange rate. The extreme case would be a graph showing balances for a six-month period three years ago, with conversion to U.S. dollars according to today's exchange rate! It's a weakness that can be fixed with an additional database table.

```php
include 'graph.php';
require_once('configuration.php');
require_once('nusoap-0.6/nusoap.php');
require_once('dateConv.php');
require_once('rounder.php');
require_once('standardGraphFormat.php');

set_time_limit(0);

$graph = new graph(700,480);

$startDate = mysql_datetime_to_timestamp($_POST['startDate']);
$endDate = mysql_datetime_to_timestamp($_POST['endDate']);

// Get details of all selected bank accounts...
```
foreach ($_POST['accounts'] as $account) {

    $midDate = $startDate;

    $arrayOfValues = array();

    $parameters = array();
    $parameters['id'] = $account;
    $soapclient = new soapclient('http://' . $accessorHost . '/acct/accessor/getSpecifiedBankAccount.php');
    $result = $soapclient->call('getSpecifiedBankAccount', $parameters);
    $currency = $result[0]['currencyId'];

    $parameters = array();
    $parameters['id'] = $currency;
    $soapclient = new soapclient('http://' . $accessorHost . '/acct/accessor/getSpecifiedCurrency.php');
    $result = $soapclient->call('getSpecifiedCurrency', $parameters);
    $xRate = $result[0]['xRate'];

    $xLabelArray = array();

    // Get the balance as of the current $midDate...

    while ($midDate <= $endDate) {
        $formattedMidDate = timestamp_to_mysql_date($midDate);
        $xLabelArray[] = $formattedMidDate;

        // Get balance and convert to USD...

        $parameters = array('id' => $account, 'date' => $formattedMidDate);
        $soapclient = new soapclient('http://' . $accessorHost . '/acct/accessor/getSpecifiedBankAccountBalance.php');
        $result = $soapclient->call('getSpecifiedBankAccountBalance', $parameters);
        $balance = $result[0]['balance'] * $xRate;
    }
/ Do some range-setting...

if ($balance < $minValue)
{
    $minValue = $balance;
}

elseif ($balance > $maxValue)
{
    $maxValue = $balance;
}

// Requires round() below, or no sub-zero range in graph.
$arrayOfValues[] = round($balance);
$midDate += 604800;

$graph->y_data[$account] = $arrayOfValues;

$graph->y_order = array();

$availableColours = array('maroon','olive','green','navy','purple','gray',
    'red','lime','yellow','blue','fuchsia','aqua','black');
$useColour=0;

// For each account, get a description and choose a color...

foreach ($_POST['accounts'] as $account)
{
    $parameters2 = array();
    $parameters2['id'] = $account;
    $soapclient2 = new soapclient('http://'.$accessorHost .
        '/acct/accessor/getSpecifiedBankAccount.php');
    $result2 = $soapclient2->call('getSpecifiedBankAccount',$parameters2);
    $accountName = $result2[0]['description'];
    $accountCurrency = $result2[0]['currency'];

    $graph->y_format[$account] = array('colour' =>
        $availableColours[$useColour], 'bar' => 'fill', 'legend' =>
        "$accountName ($accountCurrency converted to USD)"__);
$graph->y_order[] = $account;
$useColour++;
}

// Do some more formatting...

$minValue = round($minValue * 1.5);
$maxValue = round($maxValue * 1.5);

$minValue = abs($minValue);

$minValue = round_to_nearest($minValue, 50);
$minValue = $minValue * -1;

$maxValue = round_to_nearest($maxValue, 50);

// ************************

// Assign titles and generate the image...

formatGraph();

$graph->parameter['title'] = 'Weekly Balance of Accounts (in current USD)';
$graph->parameter['y_label'] = 'Balance';

$graph->x_data = $xLabelArray;

$graph->parameter['y_min_left'] = $minValue;
$graph->parameter['y_max_left'] = $maxValue;

$graph->draw();

What's new in this program? Primarily, this, the call to getSpecifiedCurrency.php on the accessor layer to determine the rate of exchange to U.S. dollars:

$parameters = array();
$parameters['id'] = $currency;
$soapclient = new soapclient('http://'.$accessorHost.'/acct/accessor/getSpecifiedCurrency.php');
$result = $soapclient->call('getSpecifiedCurrency', $parameters);
$xRate = $result[0]['xRate'];
Figure 8.4: Currawong Accounting can standardize the value of several accounts in U.S. dollars for combined reporting.

With that available, the balance of each account is converted before being added to the two-dimensional y_data array:

```php
$balance = $result[0]["balance"] * $xRate;
```

Figure 8.4 shows an example of the graph with those results.

### 8.3 Questions and Exercises

1. What validation functions could you build into the simple business-logic modules covered in this chapter?
2. What are the relative merits of doing validation in PHP on the business-logic layer and doing it in JavaScript (or another client-side language) on the presentation layer?
3. The three graphing programs in this chapter bear a lot of similarity to one another. Try combining them into a single, multipurpose class or procedural program.
4. There are some classes, available under LGPL on the Web, that will generate pie charts (the French call them Camembert charts, no kidding). Create a new reporting module that shows the relative values of all accounts in such a format.