Recipe 4

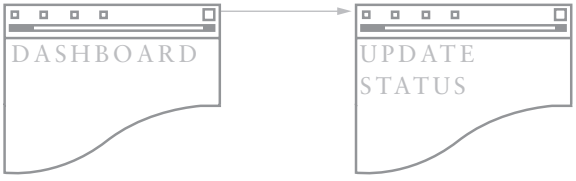
In/Out Dashboard

One of the first rules of business is to keep a close eye on all your assets—and employees are among an organization's primary assets. The In/Out Dashboard application provides a visual representation of the location for all intranet employees. With the Dashboard, you can instantly tell if an employee is in the office, in a meeting, on the road, or wherever. With minor modifications, this application can also serve to keep track of virtually any asset, such as training tapes, laptop computers, or even company cars.

Because the Dashboard is a natural employee-gathering place, it's a good location to display company-wide announcements. Such announcements need to be timely and well managed; corporate messages released before their time can be damaging to business, and those left too long on display lessen the impact of the other notices.

To support the Dashboard and its announcement section, this application uses four pages in all. One page provides the main interface for the in/out board and message section; a bullet character in a particular status column designates an employee's status. A necessary function of any such application is the ability to easily alter an employee's status; in our application, each employee's name on the in/out board is linked to a detail record where the status updates can be made. Two pages are used to administer the announcement section—one to manage the announcements in general, designating which ones should be displayed and which should be deleted; and another page for adding announcements.

User Recipes



Administrator Recipes

Ingredients

4 APPLICATION PAGES:

- 2 User Pages
  - In/Out Dashboard
  - Update Status
- 2 Administration Pages
  - Manage Announcements
  - Add Announcement

1 DATA SOURCE:

- 1 Database with:
  - 3 Tables
    - **Dashboard**—Dashboard tracks each entry in the Dashboard by employee ID, the current status, and each time the status is changed.
    - **Announcements**—Announcements maintains the announcement text, its active date, and whether or not it should be displayed.
    - **Status**—Status contains each of the available status options, by ID and name. This table is used primarily to populate list elements.
    - **Employees**—Reference is made to this table, so it needs to be included here for clarity.

- 1 View
  - **EmployeeDashboard**—This is a virtual table joining the `Announcements`, `Dashboard`, and `Status` tables.

Prep Work

Before you begin to build this application, make sure your prep work has been completed.

1. Create the data source containing the necessary tables.
   - ASP and ColdFusion users should use the **Recipes.mdb** data source found in the downloaded files while PHP users should work with the **Recipes** data source. For assistance in locating and installing these files, choose **Help > Web Application Recipes**.
2. Set up a connection to the data source. If you're unsure how to do this, see the "Connecting to Data Sources" section of Chapter 1.
   - Name the connection **Recipes**.
3. Create the template to be used for the application.
   - If you're not using the prepared files, use the template for your server model named **inoutboard**.

End User Recipe: Dashboard

Our electronic version of the old In/Out Dashboard gives an immediate visual clue as to the whereabouts of any employee. Instead of pushpins in a chart behind a secretary's desk, we use bullet symbols in a multicolumn table, available from anywhere on the intranet. The example shows seven possible status locations. These columns are, of course, completely customizable. Each employee's name acts as a link to change the status.

An announcement section is included at the bottom of the in/out board. For an announcement to be shown, it must not be dated in the future and it must be cleared for posting by an administrator. The announcements are sorted in descending date order, so the newest are displayed on top.

Step 1: Implement Dashboard Design

The basic `Dashboard` page consists of two main display areas: one for the Dashboard itself and one for the announcements section.

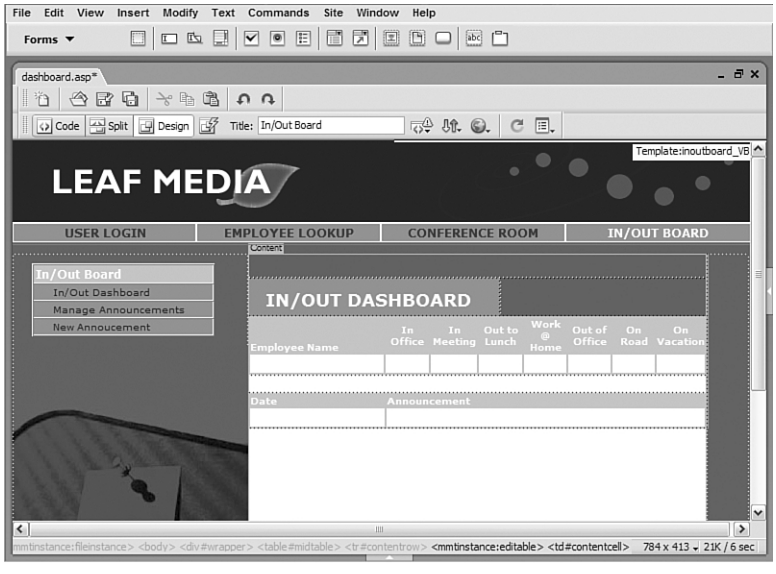
1. Create a basic dynamic page, either by hand or derived from a template.
   - 🕮 In the **InOutBoard** folder, locate the folder for your server model and open the `dashboard` page found there.
2. Add a table to the Content region of your page to contain the interface elements for the application.
   - 🕮 From the Snippets panel, drag the **Recipes > InOutBoard > Wireframes > InOut Dashboard - Wireframe** snippet into the Content editable region.
3. Within the table, nest another HTML table to display the In/Out Dashboard results. The list should have a column for the employee name as well as one for each status you want to display; our example table has room for seven such fields. Make sure your table has a border of at least 1 pixel to separate the various status fields and employees.
   - 🕮 Place your cursor in the first row below the words IN/OUT DASHBOARD and insert the **Recipes > InOutBoard > ContentTables > InOut Dashboard - Content Table** snippet.

NOTE

By using the same color for the border as the background color of the header row, you can displays lines where they're vital—in the employee rows—and keep them out of the areas where they would distract, such as the header area.

4. Below the table with the In/Out Dashboard display, add another to hold the current announcements. This table only needs two columns: one for the date and another for the announcement.
   - 🕮 Place your cursor in the bottom row of the wireframe and insert the **Recipes > InOutBoard > ContentTables > Announcements - Content Table** snippet [r4-1].

r4-1



Step 2: Add Database Components (Part 1)

With two different display areas, you might expect that we'd need two different recordsets, and you'd be right. The first recordset contains information that will be used to fill out the In/Out Dashboard area with each employee's name and their status. The second recordset gathers all the data for the announcements area. Each recordset is somewhat involved and worthy of a bit of explanation.

Let's start by adding the recordset for the announcement section because the process is similar for all server models. What distinguishes this recordset is the `WHERE` clause of the SQL statement. To ensure that an announcement is not displayed before it is supposed to be, a two-part `WHERE` clause is used. In ASP and ColdFusion, the `WHERE` clause is as follows:

```
WHERE AnnouncementDate <= Date() AND AnnouncementDisplayed <> 0
```

For PHP/MySQL, the same functionality is achieved like this:

```
WHERE AnnouncementDate <= CURDATE() AND AnnouncementDisplayed!=0
```

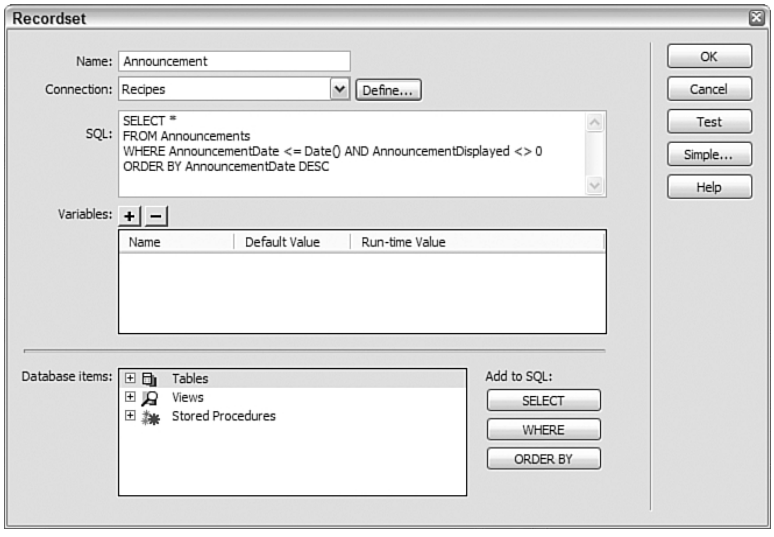
The first half of the statement ensures that the announcement date is less than or equal to today (as returned by the `Date()` or `CurDate()` function), and the second half checks to see whether the `AnnouncementDisplayed` field—which is a boolean field controlled on the `Manage Announcements` page—is not equal to `false`. You should note that the date functions are related to the databases Access and MySQL; other data sources might require a different syntax.

Here are the steps for building the `Announcement` recordset:

- To prepare for this step, copy the snippet to the clipboard by first navigating to the **Recipes > InOutBoard > SQL** folder. Then right-click (Control-click) on either the **Dashboard – Announcement RS** or the **Dashboard – Announcement RS PHP** snippet and choose Copy Snippet.

1. From the Bindings panel, choose Add (+) and select **Recordset (Query)**.
2. Switch to the advanced view and enter an appropriate name for the recordset [r4-2].
   - Enter **Announcement** in the Name field.

r4-2



3. Select the connection (data source) for the recordset.
   - Choose **Recipes** from the Connections (Data Source) list.
4. In the SQL area, enter the following code:
   - Press Ctrl-V (Command-V) to paste the copied snippet into the SQL area.

VB JS CF
```
SELECT *
FROM Announcements
WHERE AnnouncementDate <= Date() AND AnnouncementDisplayed <> 0
ORDER BY AnnouncementDate DESC
```

PHP
```
SELECT *
FROM Announcements
WHERE AnnouncementDate <= CURDATE() AND AnnouncementDisplayed != 0
ORDER BY AnnouncementDate DESC
```

5. Verify your code and click OK to close the dialog.
6. Save the page.

Step 3: Add Database Components (Part 2)

The second recordset required by the `Dashboard` page is used to populate the main in/out board interface.

For ASP and ColdFusion

NOTE

To accommodate the different dialogs for the various server models, the steps are presented separately here and when necessary throughout this recipe.

At the heart of the `In/Out Dashboard` recordset for ASP and ColdFusion is an Access view that combines three tables: `Employees`, `Dashboard`, and `Status`. To obtain a listing of all the employees—even those who might not have been assigned a status yet—you use particular kinds of SQL `JOIN`s: the `RIGHT JOIN` and the `LEFT JOIN`. You'll remember that an `INNER JOIN` returns only matching results between two tables. A `RIGHT JOIN` returns all the records found in the table on the right side of the SQL clause. (That is, `FROM Dashboard RIGHT JOIN Employees` would return all the Employees records.) Similarly, a `LEFT JOIN` returns all those records in the table to the left of the keywords. To make the SQL statement that comprises the view a bit more readable, I've separated the main clauses:

```
SELECT [Dashboard].[DashboardID], [Employees].[EmployeeFirst],
➥[Employees].[EmployeeLast], [Employees].[EmployeeID],
➥[Status].[StatusID], [Status].[StatusName]

FROM (Dashboard RIGHT JOIN Employees ON
➥[Dashboard].[DashboardEmployee]=[Employees].[EmployeeID])
➥LEFT JOIN Status ON [Dashboard].[DashboardStatus]=
➥[Status].[StatusID]

WHERE Dashboard.DashboardID =  (SELECT TOP 1 DashBoardID
➥FROM Dashboard AS DB2 WHERE DashboardEmployee =
➥Employees.EmployeeID ORDER BY DashboardID DESC)
➥OR  (SELECT TOP 1 DashBoardID FROM Dashboard AS DB2
➥WHERE DashboardEmployee = Employees.EmployeeID
➥ORDER BY DashboardID DESC) IS NULL

ORDER BY [EmployeeLast] & ' ' & [EmployeeFirst];
```

Because the SQL statement is a prebuilt view—and Macromedia Dreamweaver treats views as tables—you can insert all this complexity with point-and-click simplicity.

1. From the Bindings panel, choose Add (+) and select **Recordset**.
2. In the simple view of the Recordset dialog, enter an appropriate name.
   - Enter **Dashboard** in the Name field.
3. Select a connection (data source) to use.
   - Choose **Recipes** from the Connections (Data Source) list.
4. Choose the table or view to work with.
   - Select **EmployeeDashboard** from the Table list.
5. Leave both the Filter and Sort options unchanged, and click OK to close the dialog.

NOTE

The Table list is not totally alphabetical. Dreamweaver presents the tables first in the list and then the views.

For PHP

PHP users need to take a more circuitous route to build the needed recordset because MySQL does not support views. (If you've built the other applications in this book in order, you've seen a similar technique used before in the EmployeeLookup recipe.) To simulate the use of views, data from a table—`employeedashboard`—is initially flushed to make sure we start with a clean slate. Then that table is filled with data derived from the `employees` and `dashboard` tables, thus creating a view-like structure. The newly filled table is then referenced by this SQL statement:

```
SELECT DISTINCT employeedashboard.*
FROM employeedashboard, employeedashboard employeedashboard2
WHERE employeedashboard.DashboardID > employeedashboard2.DashboardID
➥OR employeedashboard.DashboardID IS NULL
GROUP BY employeedashboard.EmployeeID
ORDER BY employeedashboard.EmployeeLast,
➥employeedashboard.EmployeeFirst
```

Because of the way that Dreamweaver inserts code, we'll first add a new recordset using the preceding SQL and then put in the custom code needed to populate the `dashboardemployee` table.

- 🕮 Rather than enter the complex SQL statement by hand, open the Snippets panel and use the Copy Snippet command to copy it from the **Recipes > InOutBoard > SQL > Dashboard - Dashboard RS PHP SQL Statement** snippet.

1. From the Bindings panel, choose Add (+) and select **Recordset** from the list.
2. If necessary, switch to the advanced Recordset dialog.
3. Enter an appropriate name for the recordset.
   - 🕮 Enter **Dashboard** in the Name field.
4. Choose a proper connection from the list.
   - 🕮 Select **Recipes** from the Connections list.
5. In the SQL field, insert the following code:
   - 🕮 Press Ctrl-V (Command-V) to paste the code copied from the snippet into the SQL area:

```
SELECT DISTINCT employeedashboard.*
FROM employeedashboard, employeedashboard employeedashboard2
WHERE employeedashboard.DashboardID >
➥employeedashboard2.DashboardID
➥OR employeedashboard.DashboardID IS NULL
GROUP BY employeedashboard.EmployeeID
ORDER BY employeedashboard.EmployeeLast,
employeedashboard.EmployeeFirst
```

6. Click OK to close the dialog and insert the recordset.
7. Save the page.

Now that Dreamweaver has inserted our recordset code, we're ready to add the custom code that initially clears data from the `dashboardemployee` table and then refills it with the current values.

1. In Code view, place your cursor after the code that starts `<?php require_once` near the top of the page and make a new line.
2. Insert the following code:
   - From the Snippets panel, insert the **Recipes > InOutBoard > CustomCode_PHP > Dashboard - Temporary Table** snippet.

```
<?php
// Temporary Table population
mysql_select_db($database_Recipes_PHP, $Recipes);
// Delete current contents of employeedashboards table
$tempSQL = "DELETE FROM employeedashboard";
$tempRES = mysql_query($tempSQL,$Recipes);
$tempSQL = "INSERT INTO employeedashboard
➥SELECT dashboard.DashboardID, employees.EmployeeFirst,
➥employees.EmployeeLast, employees.EmployeeID, status.StatusID,
➥status.StatusName AS StatusName
➥FROM (dashboard RIGHT JOIN employees
➥ON dashboard.DashboardEmployee=employees.EmployeeID)
➥LEFT JOIN status ON dashboard.StatusID=status.StatusID
➥ORDER BY employees.EmployeeLast, employees.EmployeeFirst";
$tempRES = mysql_query($tempSQL,$Recipes);
?>
```

3. Save your page.

Step 4: Data Binding Process for Dashboard

The first aspect of this step should seem fairly familiar as we drag dynamic data for the employee's name into the first column. However, the balance of the step requires a bit of hand-coding to insert values for each of the status columns.

In each column, we'll insert code to display a bullet character if the employee's status ID matches that column. For example, if an employee is in the office—the first column—his status ID is set to 1 and a bullet is displayed in the first column. Anyone in a meeting (the second column) has a status ID of 2, and the bullet is displayed in the second column, and so on. Only one value is changed for each snippet of code placed in a different column.

Let's take care of the dynamic text for the employee name first:

1. From the Bindings panel, expand the **Dashboard** recordset.
2. Drag the **EmployeeFirst** data source field onto the row under the Employee Name column.
3. Press the right arrow key to move away from the selection and add a non-breaking space by pressing Ctrl-Shift spacebar (Command-Shift spacebar).
4. Select the **EmployeeLast** data source field and choose Insert; if you're feeling extremely dexterous, you can try dragging the data source field into position.

Now let's add the code necessary for displaying the bullet character.

1. Place your cursor in the row underneath the first status column and add the following code:
   - From the Snippets panel, open the **Recipes > InOutBoard > Custom Code** folder for your server model and insert the **In Office Status - Dynamic Text** snippet.

VB
```
<% if (Dashboard.Fields.Item("StatusID").Value = 1) then
Response.Write("<strong>&#8226;</strong>") else
Response.Write(" ") end if%>
```

JS
```
<%=(Dashboard.Fields.Item("StatusID").Value == 1)?"<strong>&#8226;
</strong>":" "%>
```

CF
```
#IIf(Dashboard.StatusID EQ 1, "'<strong>&bull;</strong>'",
"' '")#
```

PHP
```
<?php echo ($row_Dashboard['StatusID'] == 1)?"<strong>&#8226;
</strong>":" "; ?>
```

2. Repeat step 1 for each status column, incrementing the value in the code by one for each column (snippets are provided for those following the recipe). For example, in the second column, the number 1 would change to 2, like this:

VB
```
<% if (Dashboard.Fields.Item("StatusID").Value = 2) then
[Response.Write("<strong>&#8226;</strong>") else
Response.Write(" ") end if%>
```

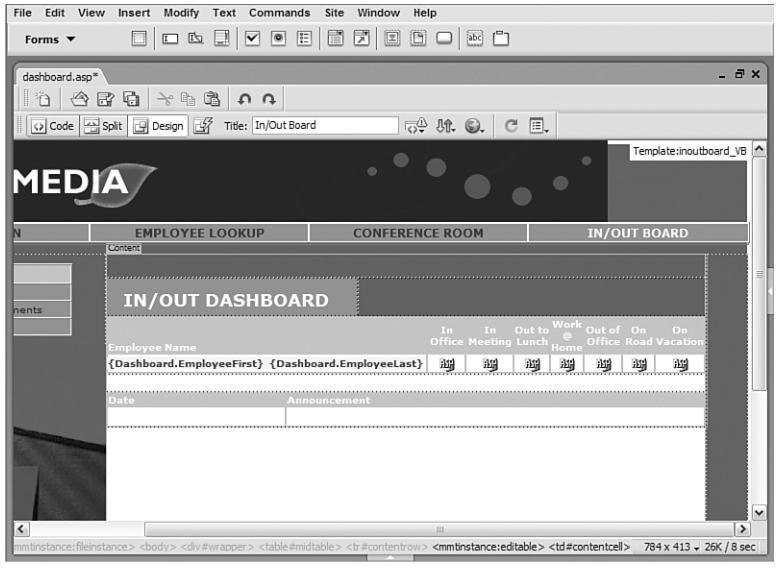
JS
```
<%=(Dashboard.Fields.Item("StatusID").Value == 2)?"<strong>&#8226;
</strong>":" "%>
```

CF
```
#IIf(Dashboard.StatusID EQ 2, "'<strong>&bull;</strong>'",
➥ "' '")#
```

PHP
```
<?php echo ($row_Dashboard['StatusID'] == 2)?"<strong>&#8226;
</strong>":" "; ?>
```

- Drag the corresponding snippet into its proper column:
  - Drag the **In Meeting Status - Dynamic Text** snippet to the `In Meeting` column.
  - Drag the **Out to Lunch Status - Dynamic Text** snippet to the `Out to Lunch` column.
  - Drag the **Work at Home Status - Dynamic Text** snippet to the `Work @ Home` column.
  - Drag the **Out of Office Status - Dynamic Text** snippet to the `Out of Office` column.
  - Drag the **On Road Status - Dynamic Text** snippet to the `On Road` column.
  - Drag the **On Vacation Status - Dynamic Text** snippet to the `On Vacation` column.

When you're done, you should see an entire row of symbols for server-side code—that is, if you have Invisible Elements enabled. ColdFusion users see the code or code indicator (`{text}`) [r4-3].



r4-3

3. Make sure all the columns have an incrementing value in the code. If, during testing, two columns show a bullet at the same time, the values are duplicates.

Creating Custom Server Behaviors

Rather than hand-code the display bullet routine seven times, Dreamweaver offers another methodology—custom server behaviors—and a tool for creating them: the Server Behavior Builder. Because this code is used repeatedly with different parameters, it's a prime candidate for a custom server behavior. Let's walk through the process:

1. Copy the code snippet you want to insert into your server behavior.

   📖 To create a server behavior that would show the column bullet, copy the **In Office Status - Dynamic Text** snippet for your server model.

2. From the Server Behaviors panel, choose Add (+) and select **New Server Behavior**.

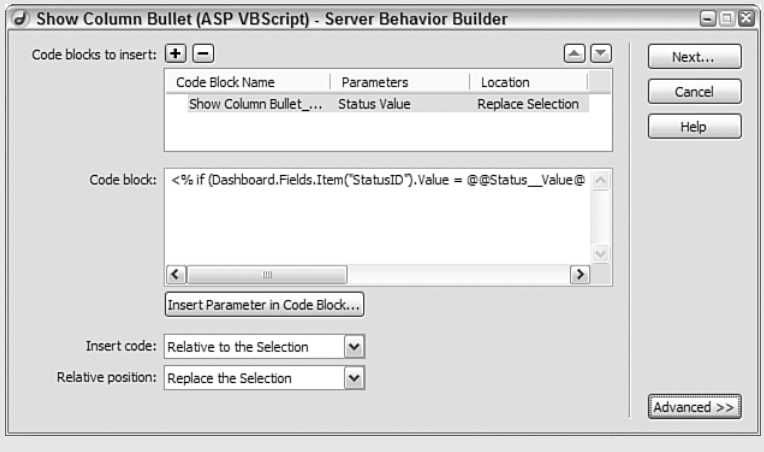
3. In the New Server Behavior dialog, select your server model and language from the Document Type list and give the custom server behavior a meaningful name unique to other server behaviors, such as `Show Column Bullet`.

   The name you enter appears in the Server Behaviors panel.

4. If this is to be a totally new server behavior, leave the Copy Existing Server Behavior option unchecked. The Copy Existing Server Behavior option allows you to build on other custom server behaviors.

After you click OK, the main Server Behavior Builder interface appears [r4-4].

r4-4



5. In the Server Behavior Builder dialog, select Add (+) to create a new code block.

6. Accept the default name or enter a custom one in the Create a New Code Block dialog, and click OK to close the pop-up dialog.

7. In the Code Block area, select the placeholder text and paste your copied code.

   If you were to stop now, the code would just go in as is without parameters and would, in essence, be a code snippet. Let's add a parameter next.

8. In the code you just pasted, select the value you want to turn into a parameter and then choose **Insert Parameter in Code Block**.

   📖 Select the number value (1, 2, 3, and so on) in the code that was previously incremented.

9. Enter a label for the parameter in the Insert Parameter in Code Block dialog and click OK to close the pop-up dialog.

   📖 This label appears in the Custom Server Behavior dialog and should indicate a direction to the user, such as `Status__Value`.

   Note: A double underscore is used in the label name. The underscores maintain the name as a single string in the code, and the Server Behavior Builder displays the double underscores as a space.

10. From the Insert Code list, choose where you want the code to go.

    📖 Because we want the code to go into the current cursor position, select **Relative to the Selection** from the list.

11. From the Relative Position list, choose where the code should go more specifically.

    📖 Select **Replace the Selection** from the list.

12. Make sure all your choices are correct and click Next.

    With the code to be inserted complete, all that's left to do is to specify what type of parameters are expected in the Generate Behavior Dialog Box dialog.

Continues

13. In the Generate Behavior Dialog Box dialog, select the parameter, and from the drop-down list under the Display As column, choose the most appropriate form control for that parameter.

    - From the Display As list, select **Numeric Text Field**.

14. Click OK to create the custom server behavior.

If you've already inserted the code, you'll see that Dreamweaver identifies that code as a server behavior now and lists all the code blocks in the Server Behaviors panel with the parameters in parentheses.

Step 5: Link to Employee Update Page

Although the code is in place to show the current status of each employee, we need a way to change that status. In this step, we create a link from the employee's name to an `update status` page.

1. Select the text or graphic you want to use as a link.
   - Select **Dashboard.EmployeeFirst** and then Shift-select **Dashboard.EmployeeLast**.
2. From the Property inspector, choose **Browse for File**, which is the folder icon next to the Link field.
3. In the Select File dialog, choose the file you want to handle the update.
   - Choose **update_status**.
4. Select **Parameters** from the Select File dialog.
5. In the Parameters dialog, enter **ID** under the Value column.
6. Under the Value column, select the lightning symbol to open the Dynamic Data dialog, and then choose **EmployeeID**.
7. Click OK to close the Dynamic Data, Parameters, and Select File dialogs.
8. Save the page.

Step 6: Data Binding Process for Announcements

Let's flesh out the announcement section a bit. We need to add two dynamic text elements: one for the date and one for the announcement.

1. From the Bindings panel, expand the **Announcements** recordset.
2. Drag the data source fields into their proper positions on the page.
   - Drag the data source field **AnnouncementDate** to the row under the Date column.
   - Drag the data source field **AnnouncementText** to the row under the Announcement column.
3. Save your page.

Step 7: Add Repeat Regions

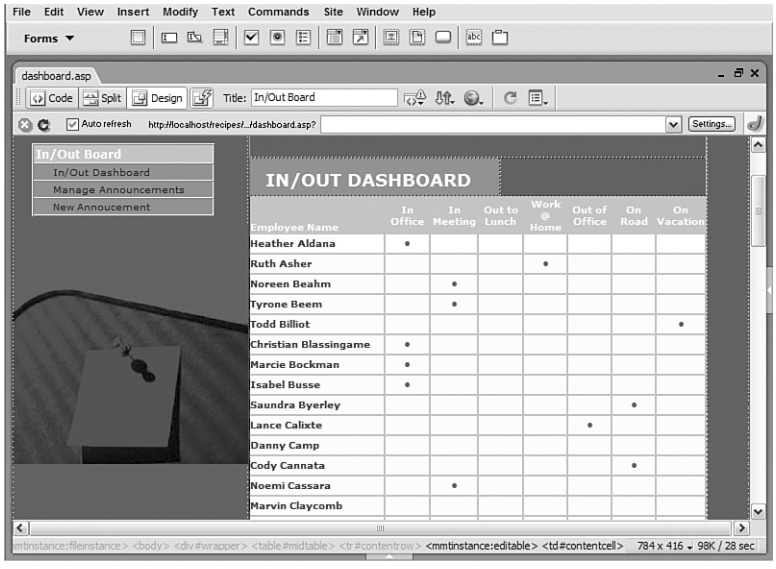
The final step on this page is to add two Repeat Region server behaviors, one for each of the dynamic data rows. Let's work with the in/out board section first.

1. Place the cursor in any of the table cells containing the dynamic data in the `Dashboard` table.
2. From the Tag Selector, select the **<tr>** tag, located to the left of the current <td> tag.
3. From the Server Behaviors panel, choose Add (+) and select **Repeat Region** from the list.
4. In the Repeat Region dialog, make sure the **Dashboard** recordset is selected.
5. Choose the **All Records** option and click OK to close the dialog.

   Now let's do the same thing for the Announcements section.
6. Select either of the dynamic data elements in the second row of the Announcements table.
7. Choose the **<tr>** tag from the Tag Selector.
8. From the Server Behaviors panel, choose Add (+) and select **Repeat Region** from the list.
9. In the Repeat Region dialog, make sure the `Announcements` recordset is selected.
10. Again, choose the **All Records** option and click OK to close the dialog.
11. Save the page.

Enter into Live Data view to see a list of all the employees, including those with and without a current status [r4-5]. In the next section, we'll create an `update status` page so that we can modify the status for any employee.

r4-5



Recipe Variations: Employee Records

As currently designed, the `Dashboard` page shows all the employees. If your organization is fairly large, you might want to show only a few employees at a time. You can do this by changing the Repeat Region server behavior to anything other than All Records and adding recordset navigation controls as demonstrated in the Employee Results page in Recipe 2, "Employee Lookup."

Another variation would be to limit the employees by department. One way to do this would be to include a list element tied to the `department` recordset and filter the `Dashboard` recordset based on the list selection. When a different department is selected, the page is resubmitted to the server and the new filter is applied. A similar technique was used in the New Job page in Recipe 2.

End User Recipe: Update Status

As we've seen in the `Dashboard` page, a link from the employee's name connects to an `update status` page, where we can modify the employee status. All the available status options are contained in a dynamically generated drop-down list.

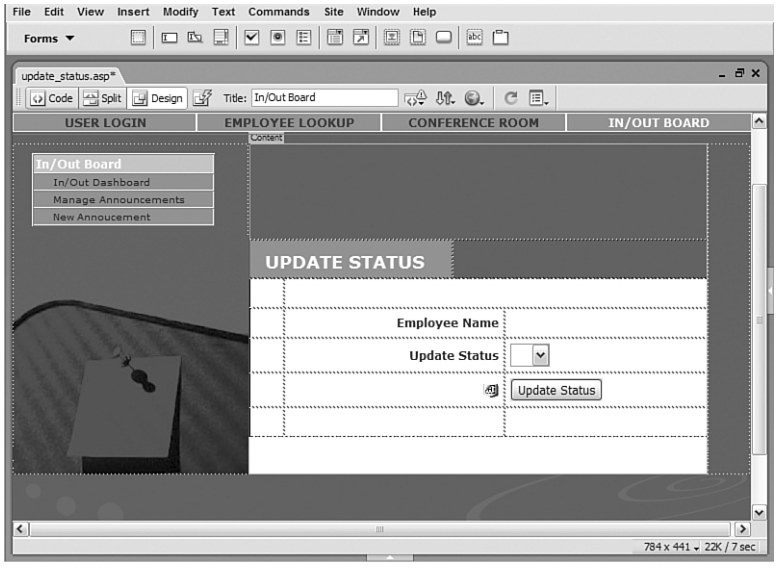
After a new status option has been chosen and the form submitted, the `Dashboard` table is updated. Because we're keeping track of each update, we're actually inserting a new record.

Not only does this have the expected effect of moving the bullet on the In/Out Dashboard to the proper column, but it also records when the status was altered. This opens the door to a possible enhancement for this application that displays a log of status changes.

Step 1: Implement Update Status Design

The requirements for an `update status` page are pretty straightforward: a form with a `dynamic list` element and a submit button.

1. Create a basic dynamic page, either by hand or derived from a template.
   - In the **InOutBoard** folder, locate the folder for your server model and open the `update_status` page found there.
2. Add a table to the Content region of your page to contain the interface elements for the application.
   - From the Snippets panel, drag the **Recipes > InOutBoard > Wireframes > Update Status - Wireframe** snippet into the Content editable region.
3. Within the table, place another HTML table to hold the form and its three elements: a dynamic list, a hidden form element to hold the employee's ID, and a submit button. Make sure you leave a labeled space to hold the employee's name, dynamically generated.
   - Place your cursor in the row below the words UPDATE STATUS and insert the **Recipes > InOutBoard > Forms > Update Status - Form** snippet [r4-6].



r4-6

Step 2: Add Database Components

Two recordsets are needed for this page: one to hold the employee's name, and one for the status values that will populate the list element. Both are straightforward and can be inserted without going into the Recordset dialog advanced view. We'll extract the employee data first.

1. From the Bindings panel, choose Add (+) and select **Recordset** (**Query**).
2. In the dialog's simple view, enter an appropriate name for the recordset.
   - Enter **Employee** in the Name field.
3. Choose a connection (data source) to use.
   - Select **Recipes** from the Connection list.
4. Choose the table in the data source to work with.
   - Select **Employees** (**employees** for PHP) from the Table list.
5. From the Columns option, choose **Selected** and select only those fields you'll need.
   - From the Columns list, choose **EmployeeID**, **EmployeeFirst**, and **EmployeeLast** by selecting one and then holding the Ctrl (Command) key and selecting the others.
6. In the Filter area of the Recordset dialog, set the four Filter list elements like this:

`EmployeeID`	`= (Equals)`
`URL Parameter`	`ID`

7. Leave the Sort option set to **None** and click OK to close the dialog.

Now let's make a recordset to populate the Status drop-down list.

1. From the Bindings panel, choose Add (+) and select **Recordset**.
2. In the dialog's simple view, enter an appropriate name for the recordset.

   Be careful not to use a reserved word such as Status for your recordset.
   - Enter **rsStatus** in the Name field.
3. Choose a connection (data source) to use.
   - Select **Recipes** from the Connection list.
4. Choose the table in the data source to work with.
   - Select **Status** (**status** for PHP)from the Table list.
5. Leave the Columns, Filter, and Sort options set to their respective defaults and click OK to close the dialog.
6. Save the page.

Step 3: Data Binding Process

It's time to bind the results of our recordsets to elements on the page. Aside from the relatively standard operations of adding dynamic text and populating a list element dynamically, we'll also link our hidden form element to the EmployeeID field of the `Employee` recordset. As you'll see later, values in hidden form elements can be passed back to the data source just like user-supplied values.

First, we'll bring in the employee name as dynamic text elements:

1. From the Bindings panel, expand the **Employee** recordset.
2. Drag the **EmployeeFirst** data source field onto the row next to the Employee Name label.
3. Press the right-arrow key to move off the selection and add a space.
4. Select the **EmployeeLast** data source field and choose **Insert**.

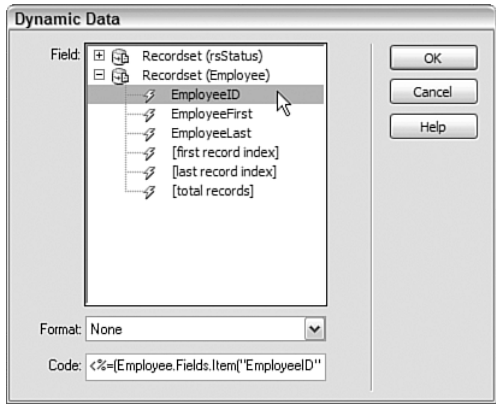
Now let's apply the `Status` recordset to populate the list element.

1. Select the **UpdateStatus** list element.
2. From the Property inspector, click the **Dynamic** button.
3. In the Dynamic List/Menu dialog, select the **rsStatus** recordset from the Options From Recordset list.
4. In the Values field, choose **StatusID**.
5. In the Labels field, select **StatusName**.
6. Leave the Select Value Equal To field blank.
7. Click OK to close the Dynamic List/Menu dialog.

Finally, let's associate the hidden form element with a field from one of our recordsets. Make sure you have Invisible Elements enabled so that you can easily locate the hidden form element's icon.

1. Select the hidden form element on the page.
   - 🕮 Choose the **EmployeeID** hidden form element located in the cell next to the Update Status button.
2. From the Property inspector, select the lightning bolt next to the Value field to open the Dynamic Data dialog.
3. In the Dynamic Data dialog, select **EmployeeID** from the `Employee` recordset [r4-7] and click OK to close the dialog when you're done.
4. Save the page.

r4-7



Recipe Variation: Selecting the Current Status

As designed, the `Status` list element initially always displays the first of the status elements in the list (in our example, In the Office)—regardless of the employee's current status. Some developers might want to bind the selection of the list element to the employee's current status. To do this, you'd need to create another recordset based on the `Dashboard` table (which contains the `DashboardStatus` field) and filtered by setting `EmployeeID` equal to the URL parameter ID. Then, in the Dynamic List/Menu dialog, select the lightning icon next to the Select Value Equal To field and choose **DashboardStatus** from the `Dashboard` recordset just established.

Step 4: Insert Record to Update Status

All that remains for this page is to store the modified information: the change in the status list. This is handily accomplished with Dreamweaver's standard Insert Record server behavior.

Why are we using Insert Record to modify a value rather than Update Record? By taking this route, we're able to keep track of all the status changes, even when they are made. If you take a close look at the `Dashboard` table in the Access database, you'll notice that the `DashboardUpdate` field is formatted as a date/time field and—most importantly—the default value is set to a function, `Now()`, which returns the current time and date.

For ASP

1. From the Server Behaviors panel, choose Add (+) and select **Insert Record**.
2. In the Insert Record dialog, select your connection from the list.
   - Choose **Recipes** from the Connections list.
3. Select the table in the data source to modify from the list.
   - Choose **Dashboard** from the Insert into Table list.
4. Enter the path to the file you want the user to visit after the record has been updated in the After Inserting, Go To field.
   - Choose Browse and locate the **dashboard.asp** file.
5. Choose the form to use.
   - Select **UpdateStatus** from the Get Values From list.
6. With the current form selected in the Get Values From list, set the form elements to their corresponding data source fields.
   - Set the `UpdateStatus` form element to the `StatusID` data column and submit it as Numeric.
   - Set the `EmployeeID` form element to the `DashboardEmployee` data column and submit it as Numeric.
7. Verify your choices and click OK to close the dialog.
8. Save the page.

For ColdFusion and PHP

1. From the Server Behaviors panel, choose Add (+) and select **Update Record**.
2. In the Update Record dialog, choose the current form.
   - Select **UpdateStatus** from the Submit Values From list.
3. Select your data source from the list.
   - Choose **Recipes** from the Data Source list.
4. Enter your username and password, if needed.
5. Select the table in the data source to insert into from the list.
   - Choose **Dashboard** (**dashboard** for PHP) from the Insert into Table list.
6. Set the data source fields to their corresponding form elements.
   - Make sure the `DashboardID` data column is set to be an unused Primary Key.

     Set `DashboardEmployee` to the `FORM.EmployeeID` form element and submit it as Numeric (Integer in PHP).

     Set `StatusID` to the `FORM.updateStatus` form element and submit it as Numeric (Integer in PHP).

     Set `DashboardUpdate` so that it does not get a value.

7. Enter the path to the file you want the user to visit after the record has been inserted in the After Inserting, Go To field.
   - Choose Browse and locate the **dashboard** file for your server model.
8. Verify your choices and click OK to close the dialog.
9. Save the page.

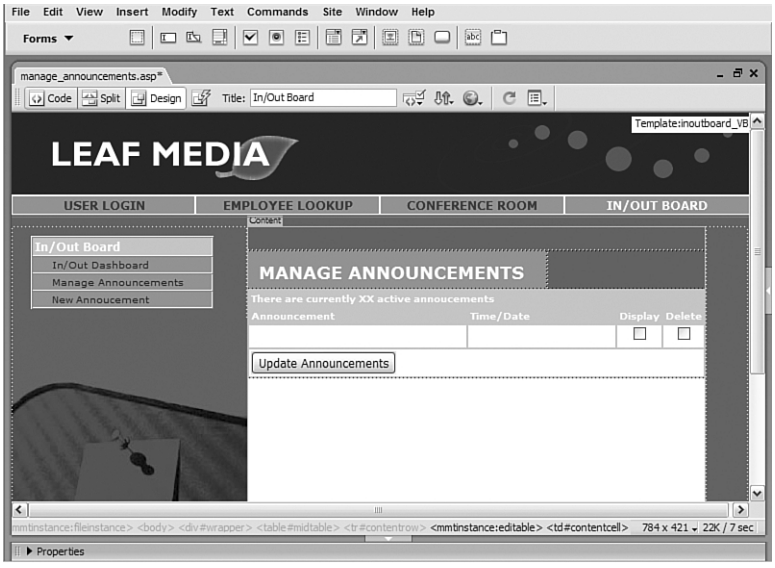
Administrator Recipe: Manage Announcements

For an effective announcement system, there needs to be a central page to control the announcements. Ideally, this page would display all the available announcements and give the administrator the option to display the currently applicable ones and delete those that are obsolete. What makes this application different from a standard update record page is that potentially every record could be updated. To accomplish this task, we'll need to set specific parameters for the recordset as well as add custom code to handle the multiple updates and deletions.

Step 1: Implement Manage Announcements Design

The first step in the application is to build the static elements of the page.

1. Create a basic dynamic page, either by hand or derived from a template.
   - In the **InOutBoard** folder, locate the folder for your server model and open the `manage_announcements` page found there.
2. Add a table to the Content region of your page to contain the interface elements for the application.
   - From the Snippets panel, drag the **Recipes > InOutBoard > Wireframes > Manage Announcements - Wireframe** snippet into the Content editable region.
3. Add a form-wrapped table with four columns. The table should have room for the announcement text, date, and two checkboxes. The only other form element needed is a submit button. It's also a good idea to add some placeholder text that will eventually show the number of announcements.
   - Place your cursor in the row below the words Employee Search MANAGE ANNOUNCEMENTS and insert the **Recipes > InOutBoard > Forms > View Announcements - Form** snippet [r4-8].



r4-8

Step 2: Add Database Components

Although there is only one recordset on this page, it has to be handled in a special fashion. Because the page will include multiple updates, in ASP, two default recordset properties need to be adjusted: the *cursor* and the *lock type*. ColdFusion and PHP are capable of handling this degree of manipulation without attributes being altered.

Let's start by creating a simple recordset.

1. From the Bindings panel, choose Add (+) and select **Recordset**.
2. In the simple Recordset dialog, enter an appropriate name.
   - Enter **Announcements** in the Name field.
3. Choose the proper connection or data source.
   - Select **Recipes** from the Connections list.
4. Select the table that contains the announcements data.
   - Choose **Announcements** from the Table list.
5. Leave the Column, Filter, and Sort options at their respective defaults and click OK to close the dialog.

With our recordset set up, ASP users are ready to modify the properties.

The following steps pertains to ASP only.

NOTE

1. From the Server Behaviors panel, select the **Announcements** recordset.
2. On the Property inspector, change the Cursor Type to **Static** and the Lock Type to **Pessimistic**. Make sure that Cursor Location remains at the default setting, **Server**.

A recordset cursor serves the same basic function as a screen cursor: Both indicate position. By default, a recordset cursor moves forward through a recordset and is known as a forward-only cursor. To update and delete multiple records, the SQL operation must be able to move forward and backward through the recordset, and that requires a *static* cursor. By the way, it's called a static cursor not because the recordset navigation is locked, but because a static copy of the recordset from the data source is used.

The lock type controls if or how the records are prevented from being updated by others. The default lock type is read-only. Although the read-only mode prevents changes from being made, it doesn't offer the needed control over the recordset. To accomplish our goal, the lock type should be changed to Pessimistic. With a Pessimistic lock type in place, the records remain locked until an `update` command is issued.

Step 3: Data Binding Process

In addition to using dynamic text to show the announcement and its associated time and date, we'll add a repeat region to show all the available announcements. To make it easier for the administrator to know how many announcements are in the system, the total number of records is also shown, using slightly different techniques for the various server models.

1. From the Bindings panel, expand the **Announcements** recordset.
2. Drag the **AnnouncementText** data source field onto the row in the Announcement column.
3. Drag the **AnnouncementDate** data source field onto the row in the Time/Date column.

Now let's add the Repeat Region.

1. Select either of the dynamic elements just placed on the page.
2. From the tag selector, choose the **<tr>** tag to the left of the current selection in the tag selector.
3. From the Server Behaviors panel, choose Add (+) and select **Repeat Region**.
4. In the Repeat Region dialog, make sure the `Announcements` recordset is chosen, and set the option to show All Records. Click OK when you're done.

For ASP

Now we're ready to replace the placeholder with the dynamic record count code.

1. Select the **XX placeholder** text in the top table row.
2. From the Bindings panel, drag the [**total records**] data source item onto the page over the selection.
3. Save the page.

Preview the page in Live Data view to get a count of the number of announcements registered.

For ColdFusion and PHP

Dreamweaver's ColdFusion and PHP server models provide a server behavior for displaying the total number of records easily.

1. Select and delete the **XX placeholder** text.
2. From the Server Behaviors panel, choose Add (+) and then select **Display Record Count > Display Total Records**.
3. In the Display Total Records dialog, choose the **Announcements** recordset and click OK.
4. Save your page.

Preview the page in Live Data view to get a count of the number of announcements registered.

Step 4: Insert Dynamic Checkbox Options

It's time to add in the dynamic checkboxes with our custom code. The custom code is needed to supply two attributes with proper values: `name` and `checked`. For ASP and ColdFusion, the `name` attribute combines the root word `Display` with the entry's `AnnouncementID` value, resulting in names such as `Display1`, `Display2`, and so on. PHP takes a slightly different tack, where the name indicates an array (`Display[]`) and the value contains the proper `AnnouncementID`. These unique names are important because they will be used during the update process to modify or delete the records. The same procedure is applied to both checkboxes found under the Display and Delete columns. To insert the necessary dynamic values, we'll use Dreamweaver's Tag Inspector.

The `checked` attribute reads the `AnnouncementDisplayed` value for the record and, if `true`, includes the attribute; if `false`, the attribute is left out. The checked state is set with the Dynamic Checkbox server behavior.

Let's start by setting the name (and, for PHP, the value) dynamically for both checkboxes. We'll handle the Display checkbox first:

1. Select the checkbox under the Display column, temporarily named **Display**.
2. Choose **Window > Tag Inspector** to display the Tag Inspector panel; make sure the Attributes category is visible.
3. If necessary, switch to List view by selecting the A–Z icon.

   The two different views for the Tag Inspector (Category and List) were added in Dreamweaver MX 2004; Dreamweaver MX only offers the List view.
4. Select the field next to the `name` attribute, which currently contains the word "Display."
5. Give the `name` attribute a dynamic value according to your server model:

   (VB) (JS) (CF) Choose the lightning bolt symbol to open the Dynamic Data dialog and select **AnnouncementID** from the `Announcements` recordset. Position your cursor at the front of the Code field and add the word **Display** in addition to the code already present. Click OK when you're done to close the dialog.

   (PHP) Add an opening and closing bracket after the Display name so that the value reads **Display**[].
6. PHP only: Select the field next to the `value` attribute and choose the lightning bolt symbol to open the Dynamic Data dialog; select **AnnouncementID** from the `Announcements` recordset. Click OK to close the dialog when you're done.

The same process is now applied to the Delete checkbox to provide unique names for each of those checkboxes.

1. Select the checkbox under the Delete column, temporarily named **Delete**.
2. In the Tag Inspector, select the field next to the `name` attribute, which currently contains the word "Delete."
3. Give the `name` attribute a dynamic value according to your server model:

   (VB) (JS) (CF) Choose the lightning bolt symbol to open the Dynamic Data dialog and select **AnnouncementID** from the `Announcements` recordset. Position your cursor at the front of the Code field and add the word **Delete** in addition to the code already present. Click OK when you're done to close the dialog.

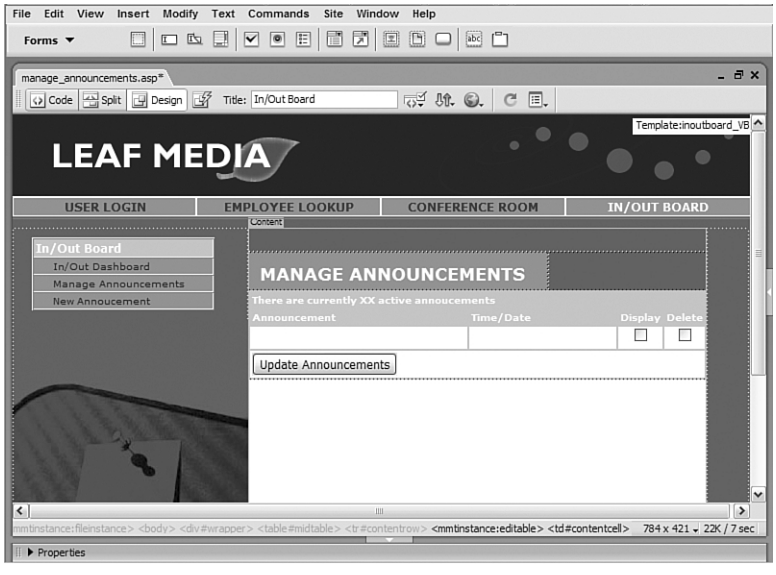
   (PHP) Add an opening and closing bracket after the Delete name so that the value reads **Delete**[].

4. PHP only: Select the field next to the `value` attribute and choose the lightning bolt symbol to open the Dynamic Data dialog; select **AnnouncementID** from the `Announcements` recordset. Click OK to close the dialog when you're done.

The Display checkbox requires an additional step to toggle the `checked` attribute according to the `AnnouncementDisplayed` value:

1. Select the Display checkbox.
2. From the Property inspector, choose **Dynamic**.
3. In the Dynamic Checkbox dialog, choose the field you want to evaluate.
   - Select the lighting bolt in the Check If field and choose **AnnouncementDisplayed** from the `Announcements` recordset.
4. Set the condition that will mark the `form` element with a check.
   - In the Equal To field, enter the proper value for your server model:
     - VB **True**
     - JS **1**
     - CF **1**
     - PHP **1**
5. When you're done, click OK to close the Dynamic Checkbox dialog.
6. Save your page.

After the custom checkboxes are in place, enter Live Data view to see which announcements currently are set to be displayed [r4-9].



r4-9

Step 5: Update Record

The final step for the `Manage Announcements` page is to add the supporting logic to update or delete the records as indicated. There are two basic operations here. The first looks for entries in which the Delete checkbox is selected and deletes the record. The second compares the Display checkbox value (whether it is checked or not) to the `AnnouncementDisplayed` value in the data source. If the two are the same, the user made no changes, and the routine moves to the next record to avoid unnecessary updates. However, if the two differ, the record is updated to reflect the value of the checkbox. The code block needs to be placed beneath the recordset declaration.

1. From the Server Behaviors panel, choose the **Announcements** recordset.
2. Switch to Code view to see the recordset code block highlighted. Create a new line after the recordset code block and before the next code block and place your cursor on that line.
3. Insert the following code:
   - From the Snippets panel, open the **Recipes > InOutBoard > Custom Code** folder for your server model and insert the **Update Delete Multiple Announcement Records** snippet.

VB
```
<%
if (cStr(Request("UpdateAnnouncements"))<>"")  then
  while (NOT Announcements.EOF)
    if (cStr(Request("Delete"&Announcements.Fields
    ➥("AnnouncementID").value))<>"")  then
        Announcements.Delete()
        Announcements.Update()
     else
        Dim Display
        Display = (cStr(Request("Display"&Announcements.Fields
        ➥("AnnouncementID").value))<>"")
        if  (Display <> Announcements.Fields
        ➥("AnnouncementDisplayed").value)  then
          Announcements.Fields("AnnouncementDisplayed").value =
          ➥NOT Announcements.Fields("AnnouncementDisplayed").value
          Announcements.Update()
        end if
      end if
      Announcements.MoveNext()
  wend
  if (Announcements.RecordCount > 0)  then
    Announcements.MoveFirst()
  end if
end if
%>
```

JS
```
<%
if (String(Request("UpdateAnnouncements"))!="undefined")  {
  while (!Announcements.EOF)  {
    if (String(Request("Delete"+Announcements.Fields
    ➥("AnnouncementID").value))!="undefined")  {
    Announcements.Delete();
    Announcements.Update();
  }
  else  {
    var Display = (String(Request("Display"+Announcements.Fields
    ➥("AnnouncementID").value))!="undefined")
    if  (Display !=
Announcements.Fields("AnnouncementDisplayed").value)  {
      Announcements.Fields("AnnouncementDisplayed").value =
     ➥ !Announcements.Fields("AnnouncementDisplayed").value;
      Announcements.Update();
      }
  }
  Announcements.MoveNext();
  }
  if (Announcements.RecordCount > 0)
    Announcements.MoveFirst();
}
%>
```

CF
```
<cfif IsDefined("Form.UpdateAnnouncements")>
  <cfloop query="Announcements">
    <cfif isDefined("Form.Delete"&Announcements.AnnouncementID) >
    <cfquery datasource="Recipes">
      DELETE FROM Announcements WHERE AnnouncementID =
     ➥ #Announcements.AnnouncementID#
    </cfquery>
  <cfelse>
    <cfif (isDefined("Form.Display" & Announcements.AnnouncementID)
    ➥NEQ Announcements.AnnouncementDisplayed)>
          <cfquery datasource="Recipes">
          UPDATE Announcements SET AnnouncementDisplayed =
          ➥#isDefined("Form.Display" & Announcements.AnnouncementID)#
          </cfquery>
    </cfif>
  </cfif>
  </cfloop>
  <cfquery name="Announcements" datasource="Recipes">
    SELECT * FROM Announcements
  </cfquery>
</cfif>
```

PHP
```
<?php
mysql_select_db($database_Recipes_PHP, $Recipes_PHP);
if (isset($_POST['UpdateAnnouncements'])) {
  // First Displays
  if (count($_POST['Display'] > 0)) {
        $bipassArr = array();
        for ($k=0; $k < count($_POST['Display']); $k++) {
              // First the items to display
              if ($_POST['Display'][$k]!="") {
                    $sql = "UPDATE announcements
                    ➥SET AnnouncementDisplayed=1
                    ➥WHERE AnnouncementID = " . $_
                    ➥POST['Display'][$k];
                    $bipassArr[] = "AnnouncementID !=
                    ➥ ".$_POST['Display'][$k];
                    mysql_query($sql,$Recipes_PHP);
              }
        }
        $sql = "UPDATE announcements SET AnnouncementDisplayed=0";
        if (count($bipassArr) > 0) {
              $sql.= " WHERE " . implode(" AND ",$bipassArr);
        }
        mysql_query($sql,$Recipes_PHP);
  }
  // Now Deletes
  if (count($_POST['Delete']) > 0) {
        for ($k=0; $k < count($_POST['Delete']); $k++) {
              if ($_POST['Delete'][$k]!="") {
                    $sql = "DELETE FROM announcements
                    ➥WHERE AnnouncementID=".$_POST['Delete'][$k];
                    mysql_query($sql,$Recipes_PHP);
             }
        }
  }
}
?>
```

4. Save your page.

Your page is now ready for testing, although you might want to wait until the next and final page of the recipe is completed so that you can add dummy announcements for deletion.

Administrator Recipe: Add Announcements

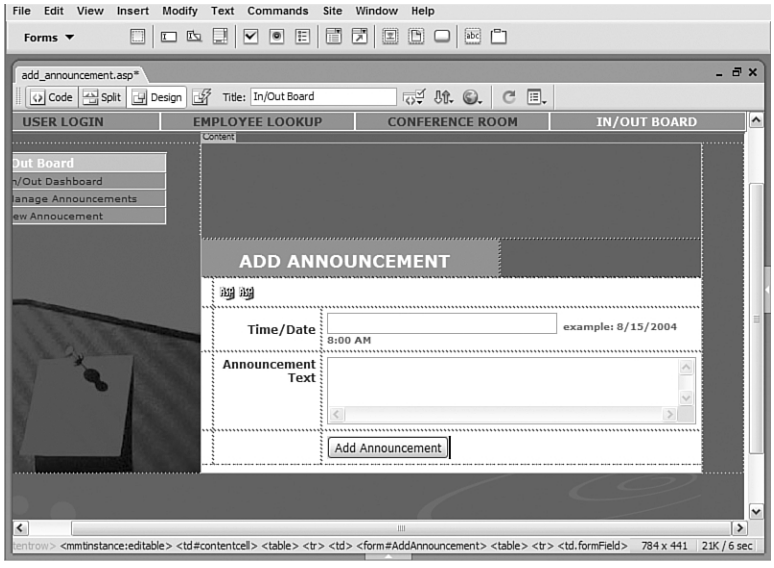
In addition to managing existing announcements, another administrative task is to add new ones. The records for announcements are straightforward and contain only two fields: one for the text and one for the date. The date information is stored in a date/time field in the data source; therefore, it must be properly formatted before it can be inserted. We'll use a bit of server-side validation to make sure our date is in the proper format; if it is not, we include code to trigger an error message.

Step 1: Implement Add Announcement Design

The first step is, of course, to create the basic page to hold the form and its elements.

1. Create a basic dynamic page, either by hand or derived from a template.
   - 🕮 In the **InOutBoard** folder, locate the folder for your server model and open the `add_announcement` page found there.
2. Add a table to the Content region of your page to contain the interface elements for the application.
   - 🕮 From the Snippets panel, drag the **Recipes > InOutBoard > Wireframes > Add Announcement - Wireframe** snippet into the Content editable region.
3. Within the table, place another HTML table to hold the form and three elements: a text field, a text area, and a submit button. Be sure you leave space to hold a date validation error message.
   - 🕮 Place your cursor in the row below the words ADD ANNOUNCEMENT and insert the **Recipes > InOutBoard > Forms > Add Announcement - Form** snippet [r4-10].

r4-10



Step 2: Insert Record for Announcement

After the user has entered the new announcement information and pressed the submit button, the Insert Record server behavior logic—which we are about to apply—takes over.

For ASP

1. From the Server Behaviors panel, choose Add (+) and select **Insert Record**.
2. In the Insert Record dialog, select your connection from the list.
   - Choose **Recipes** from the Connections list.
3. Select the table in the data source to modify from the list.
   - Choose **Announcements** from the Insert into Table list.
4. Enter the path to the file you want the user to visit after the record has been updated in the After Inserting, Go To field.
   - Choose Browse and locate the **manage_announcement.asp** file.
5. Choose the form to use.
   - Select **AddAnnouncement** from the Get Values From list.
6. With the current form selected in the Get Values From list, set the form elements to their corresponding data source fields.

- Set the `TimeAndDate` form element to the `AnnouncementDate` data column and submit it as Date.
- Set the `AnnouncementText` form element to the `AnnouncementText` data column and submit it as Text.

7. Verify your choices and click OK to close the dialog.
8. Save the page.

For ColdFusion and PHP

1. From the Server Behaviors panel, choose Add (+) and select **Update Record**.
2. In the Update Record dialog, choose the current form.
   - Select **AddAnnouncement** from the Submit Values From list.
3. Select your data source from the list.
   - Choose **Recipes** from the Data Source list.
4. Enter your username and password, if needed.
5. Select the table in the data source to insert into from the list.
   - Choose **Announcements** (**announcements** for PHP) from the Insert Into Table list.
6. Set the data source fields to their corresponding form elements.
   - Make sure the `AnnouncementID` data column is set as an unused Primary Key.

     Set `AnnouncementDate` to the `FORM.TimeAndDate` form element and submit it as Date.

     Set `AnnouncementDisplayed` to not get a value.

     Set `AnnouncementText` to the `FORM.AnnouncementText` form element and submit it as Text.
7. Enter the path to the file you want the user to visit after the record has been inserted in the After Inserting, Go to field.
   - Choose Browse and locate the **manage_Announcement** file for your server model.
8. Verify your choices and click OK to close the dialog.
9. Save the page.

NOTE

Although the announcement record will be entered into the data source, it will not immediately appear on the `In/Out Dashboard` page. To ensure that only properly cleared announcements are seen, this application requires that the Display checkbox show as checked on the `Manage Announcements` page—thus, as it is described in Web jargon, pushing the announcement live.

Step 3: Server-Side Form Validation

Two separate code blocks need to be inserted to validate the user-entered date. One code block handles the processing and makes sure a valid date is received, and a second code block outputs an error message if a problem is found. The trick here is to place the error message code block first, so that the page is checked when it is first loaded.

1. Place your cursor in the row below the Add Announcement label and above the Time/Date label and text field.
2. Insert the following code:
   - From the Snippets panel, open the **Recipes > InOutBoard > Custom Code** folder for your server model and insert the **Date Error - Dynamic Text** snippet.

VB
```
<% if (cStr(Request.QueryString("badDate")) <> "") Then
Response.Write("The date you entered was not in the proper format.
Dates should be in the format mm/dd/yyyy.  <br>(Use your browser's
back button to edit your previous entry).")%>
```

JS
```
<%=(String(Request.QueryString("badDate")) != "undefined")
?"The date you entered was not in the proper format.
Dates should be in the format mm/dd/yyyy.  <br>
(Use your browser's back button to edit your previous entry).":""%>
```

CF
```
<cfif isDefined("URL.badDate")>The date you entered was not in
the proper format.  Dates should be in the format mm/dd/yyyy.  <br>
(Use your browser's back button to edit your previous
entry).</cfif>
```

PHP
```
<?php echo (isset($_GET['badDate']))?"The date you entered was not
in the proper format.  Dates should be in the format mm/dd/yyyy.
<br>(Use your browser's back button to edit your previous
entry).":""; ?>
```

Now, we're ready to add the second code block to perform the date validation.

3. In Code view, place your cursor at the top of the page, just after the connection code, if any.
4. Insert the following code:
   - From the Snippets panel, open the **Recipes > InOutBoard Custom Code** folder for your server model and insert the **Server-side Date Validation** snippet.

VB
```
<%
if (cStr(Request.Form("AddAnnouncement"))<> "")  then
  dim DateString
  DateString = cStr(Request.Form("TimeAndDate"))
  if not (isDate(DateString)) then
```

```
    Response.Redirect("add_announcement.asp?badDate=true;")
  end if
end if
%>
```

JS

```
<%
function isADate(DateString)  {
  var DateRegex = /^(\d{1,2})\/(\d{1,2})\/(\d{4})$/
  var theMatch = DateString.match(DateRegex);
  var validDate = false;
  if (theMatch)  {
    var DateObj = new Date(DateString);
  if ((DateObj.getMonth()+1 == parseInt(theMatch[1])) &&
  ➥(DateObj.getDate() == parseInt(theMatch[2])) &&
  ➥(DateObj.getFullYear() == parseInt(theMatch[3])))
      validDate = true;
  }
  return validDate;
}
%>
<%
if (String(Request.Form("AddAnnouncement"))!="undefined" &&
➥(!isADate(String(Request.Form("TimeAndDate"))) ¦¦
➥String(Request.Form("TimeAndDate"))=="")
  Response.Redirect("add_announcement.asp?badDate=true");%>
```

CF

```
<cfif isDefined("FORM.AddAnnouncement")>
  <cfif NOT IsDate(FORM.TimeAndDate)>
    <cflocation url="add_announcement.cfm?badDate=true">
  </cfif>
</cfif>
```

PHP

```
<?php
function isADate($DateString) {
  $validDate = false;
  if (ereg("^[0-9]{1,2}\/[0-9]{1,2}\/[0-9]{4}$",$DateString)) {
        $today = date("Y");
        $submittedDate = explode("/",$DateString);
        $month = $submittedDate[0];
        $day = $submittedDate[1];
        $year = $submittedDate[2];
        if ($year >= $today) {
              $validDate = (checkdate($month,$day,$year))?true:false;
        }
  }
  return $validDate;
```

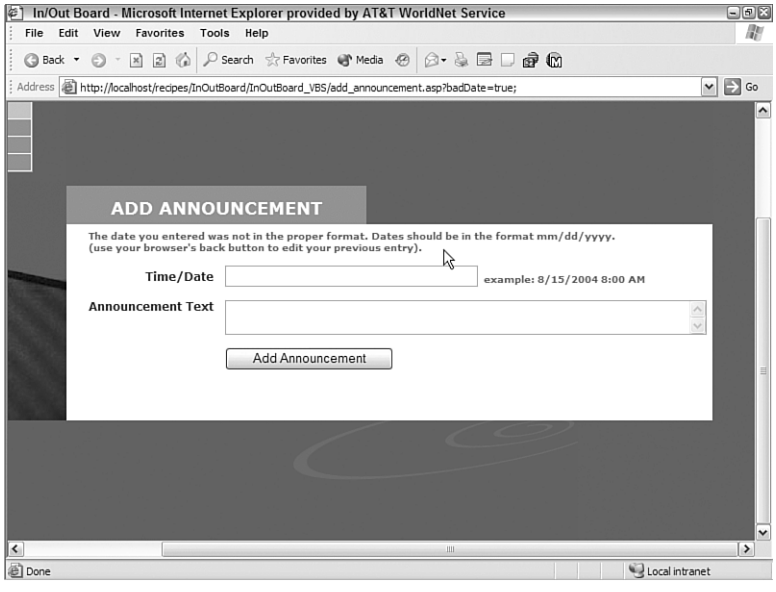
Continues

Continued

```
}
?><?php
if ((isset($_POST['TimeAndDate'])) & (!isADate
➥($_POST['TimeAndDate']))) {
   $url = "add_announcement.php?badDate=true";
   header("Location: $url");
}
?>
```

The server-side data validation is a powerful bit of code that could easily be adapted to different applications. Test your page either by entering into Live Data view and adding the URL parameter `badDate=true` or by previewing in the browser and entering an improper date format [r4-11].

r4-11



5. Save the page.

The entire application is now ready for testing and deployment.