

## 5

# Portal Features in BVSN Financial Sample Application

One-To-One Financial enables financial institutions to add features to their Web sites that encourage customers to use their personal home pages as portals. These portal features allow customers to add links to their favorite Web sites on their personal home pages, and to have their home page remind them of important upcoming dates.

The One-To-One Financial link storing capability allows a simple, yet flexible implementation for saving the URL of external Web sites. Customers can save their favorite Web sites by entering the URL and a friendly name for the site. These links can be displayed, edited, removed, and sorted either alphabetically or arbitrarily.

The One-To-One Financial reminders feature provides a simple means for customers to enter important dates that they want to be reminded of and specify how soon before that date they should be reminded. This feature is an extension of the One-To-One Financial alerts feature.

The financial institution can specify the layout, color, font, and style of the page that holds the links, including the presence or absence of buttons, number of buttons, and the text on the buttons. The financial institution can also control the layout and style of the reminders pages and the text that is displayed to remind the customers.

---

## Portal operations

The BVSN Financial sample application demonstrates one approach to creating a financial institution site. One-To-One Financial provides support for the following link management operations for the customer:

Favorite Links Operation	Description
View links	Customers can choose to have their links displayed on the home page or only on the Favorite Links page.
Add new links	Customers can create new links.
Delete links	Customers can delete existing links.
Edit links	Customers can edit existing links, either by changing the URL or friendly name of the links.
Sort links	Customers can sort their links either arbitrarily with the Move Up and Move Down buttons, or alphabetically.

The BVSN Financial sample application also illustrates an implementation of the reminders features. One-To-One Financial provides support for the following reminders management operations for the customer:

Reminders Operation	Description
View reminders	Customer can view their existing reminders.
Add new reminders	Customers can create new reminders.
Edit reminders	Customers can edit existing reminders, either by changing the date, the description, or when the application should starting reminding the customer of the event.
Delete reminders	Customers can delete existing reminders.

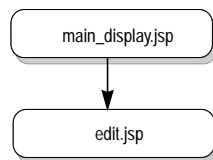
This chapter discusses the following portal management operations in greater detail.

- “Favorite links” on page 82
- “Reminders” on page 91

---

## Favorite links

The following diagram shows the scripts used for adding and managing favorite links. These pages are located in the `§BV1T01/finance/bvsnfi/scripts/favoritelinks` directory.



This section describes the following:

- “Viewing links” on page 82.
- “Updating links” on page 83.
- “Adding links” on page 85.
- “Editing links” on page 86.
- “Deleting links” on page 88.
- “Sorting links manually” on page 89.
- “Sorting links alphabetically” on page 91.

---

## Viewing links

The customer home page displays links in the upper right area of the page under the heading My Favorite Links. Customers can go to the Web link management page by clicking either the My Favorite Links heading or the Favorite Links navigation link in the left frame. In either case, the BVSN Financial sample application displays the `§BV1T01/finance/bvsnfi/scripts/favoritelinks/main_display.jsp` page. This page does the following:

main\_display.jsp

1. Initializes the page with `BVJS_PTFLInit()`, which sets the session variables for use in writing a form and for editing the Web sites.
2. Sets the value of `formatString`, which includes the `BVJS_PTFLColumnURL` and `BVJS_PTFLColumnName` constants.

▶ The constants used by the favorite links functions are described in “Favorite links constants” on page 324.

3. Displays the `formatString` from the previous step. This displays the Web links that the customer has saved.

```
Response . write(BVJS_PTFLShow(formatString, ""));
```



4. Uses the `makeScriptURL` function to add the session information from this page to the `edit.jsp` page. In this way the `edit.jsp` page has the customer’s session information should the customer chose to go to that page.

## Updating links

The functionality that allows the customers to add, edit, delete, and sort their favorite links is contained in the `edit.jsp` page. The functionality provided by this page is described in this section and “Adding links” on page 85, “Editing links” on page 86, “Deleting links” on page 88, “Sorting links manually” on page 89, and “Sorting links alphabetically” on page 91. The beginning of this script applies to all of the following operations. This page does the following:

edit.jsp

1. Initializes the page with `BVJS_PTFLInit()`, which makes sure that the session variables have been set for use in displaying the form and for editing the Web sites. `BVJS_PTFLInit()` does not take any explicit parameters. It returns zero if successful and non-zero for an error. This function sets the `portal` variable to a `BVI_Properties` stored in the Session object. After `portal` is set, it contains:
  - `linkCounter`, which is used to help generate a unique key for new links.
  - `linksTable`, which is a `BVI_MultiValueTable` that stores the customer’s links.
  - `nLinks`, which is an integer containing the number of links the customer has stored.

- **linkError\_blankURL**, which is a flag indicating that the customer has tried to create a link (through either addition or modification) without entering a URL. This is used as a control flag for displaying error information to the customer. This flag is set (set is 1, not set is 0), by one of **BVJS\_PTFLProcessForm()**, **BVJS\_PTFLAdd()**, or **BVJS\_PTFLUpdate()**.
- **linkError\_updateInvalid**, which indicates that the customer has tried to perform an update without selecting a link to update. This is set by the **BVJS\_PTFLUpdate()** method.
- **workingLink**, which is a **BVI\_MultiValueRow** corresponding to the link, if any, that the customer has chosen to modify. You can access the columns of the **BVI\_MultiValueRow** with its get method and the constants defined in **LinkConstants**.


2. Caches the value of **portal** by retrieving the **portal** value from the **Session** object.

```
var portal = Session . portal;
```

This value is used later in this script.

3. Checks to see if the customer has sent the cancel changes message. If cancel changes has not been sent, the page calls the **BVJS\_PTFLProcessForm()** method. This method processes the form and posts to itself, that is, updates the form. This **BVJS\_PTFLProcessForm()** method performs the updates to the form, whether the update is adding, editing, deleting, or sorting links.

The form is created by the combination of a JavaScript page, the session variables defined in the **BVJS\_PTFLInit()** method, and the previous version of this HTML page generated by the **BVJS\_PTFL>EditAll()** method.

 In the BVSN Financial sample application, the **BVJS\_PTFL>EditAll()** function is the interface to the **add**, **delete**, **update**, **sortAlphabetically**, and **sortManually** methods, none of which needs to be called directly. See “**BVJS\_PTFL>EditAll()**” on page 327.

4. Checks to see if the **Request** object contains a field named **BVJS\_PTFLFieldUpdateLink**. If it does, the page calls **clearWorkingRow**, which provides a clear row to enter a new link. (This method discards any unsaved changes in the current row, but it does not delete a row.) The **BVJS\_PTFLFieldUpdateLink** updates the database table, **BV\_PTFLLinks**, the name of which is stored in **BVJS\_PTFLTableName**.
5. Sets the value of **workingLink** by retrieving the value from the **portal** object that was set in [Step 2](#). This assigns the variable **workingLink** to a property of the **Session** object to make it more quickly accessible to this script.

```
var workingLink = portal . workingLink;
```

6. Stores **BVJS\_PTFLColumnID** in the HTML field named **BVJS\_PTFLFieldWorkingID**. Because this is a hidden field, the customers do not see the **BVJS\_PTFLFieldWorkingID**, but it does hold the name and value pair for use when this page posts to itself, that is, updates.
7. Retrieves the URL of the working link, and stores **BVJS\_PTFLColumnURL** in the HTML field named **BVJS\_PTFLFieldWorkingURL**. These values are not visible to the customer, but store value pair information.
8. Checks to see if the **linkError\_blankURL** has been set. If so, the page displays an error message.
9. Calls the **get** method of the **workingLink** object. This gets the value of the column named by the contents of **BVJS\_PTFLColumnName**.
10. Sets the value of **selectOptions**. This identifies the links (options) in the customer’s list of saved links for display.

11. Displays the HTML form that allows customers to add, edit, delete, and sort their links.

---

## Adding links

To add a link, this page performs all of the steps in [“Updating links” on page 83](#), then does the following:

1. Checks the value of the ID of **workingLink**. If the customer is adding a link, the **workingLink** variable is null. The script displays the Add New Link button so the customer can add a new link.

```
<%
if(workingLink . get(BVJS_PTFLColumnID) != "") {
%>
<INPUT TYPE=submit NAME="<%= BVJS_PTFLFieldUpdateLink %>"
      VALUE="Save Changes">
<INPUT TYPE=submit NAME="Cancel Changes" VALUE="Cancel">
<%
} else {
%>
<INPUT TYPE=submit NAME="<%= BVJS_PTFLFieldAddLink %>"
      VALUE="Add New Link">
<%
}
%>
```

2. Sets the value of **formatString**, which later uses the server-side JavaScript to dynamically generate the client-side JavaScript. This happens when the **formatString** is passed to the formatter by the [BVJS\\_PTFL>EditAll\(\)](#) method. This is the part of the script that displays the fields and buttons needed to add a link.

```
<%
var formatString =
  "<SCRIPT>selectOptions = selectOptions + " +
  "'<OPTION VALUE=\\'<%= " + BVJS_PTFLColumnID + " %s " +
  "encoding:html:std %>\\'> " +
  "<%= " + BVJS_PTFLColumnName + " %s encoding:html:std
  %>' ;</SCRIPT>\n";
%>
```

3. Displays the rest of the HTML form. The **selectOptions** here is taken from the code in [Step 2](#). (This section of the script comes after **moveUp** function.)

```
<select name="ReorderList" size="10">
<script> document . write(selectOptions); </SCRIPT>
```

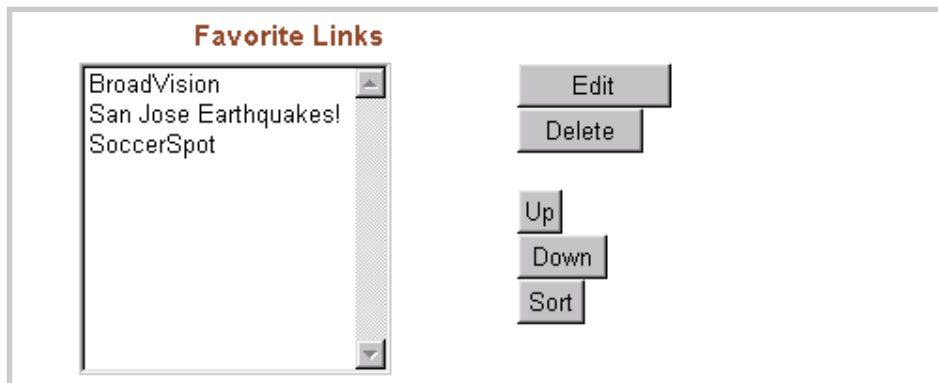
4. The customer enters the new link name and URL.



URL:

Name:

5. Makes sure that the selected information is passed to the `BVJS_PTFLProcessForm()` method correctly. Then displays the page again.
6. When the customer clicks the Add New Link button, the script runs the `BVJS_PTFLProcessForm()` method. This updates the page, adding the new link.



**Favorite Links**

BroadVision  
San Jose Earthquakes!  
SoccerSpot

---

## Editing links

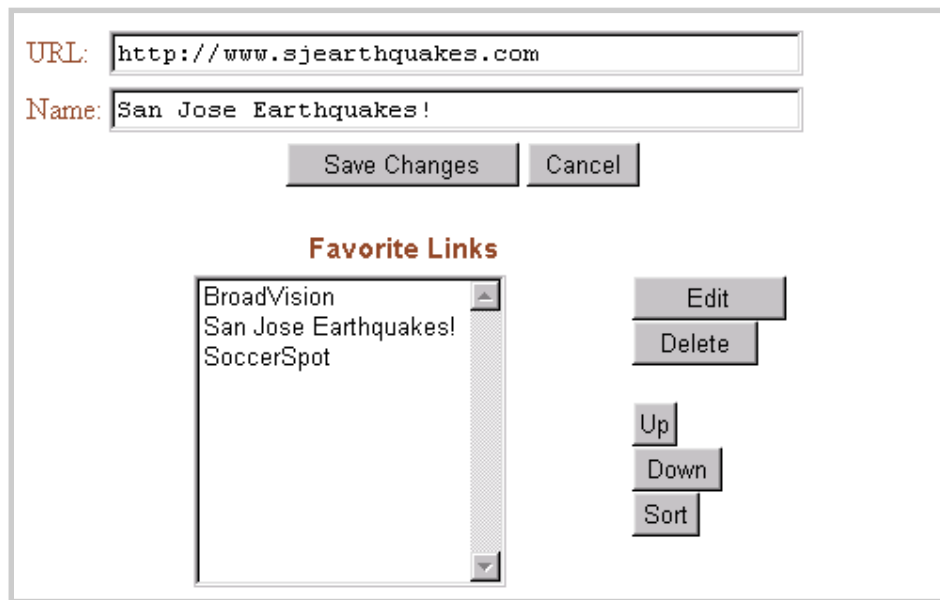
To edit an existing link, this page performs all of the steps in “Updating links” on page 83, then does the following:

1. The customer selects a link and clicks the Edit button. The page updates using the `BVJS_PTFLProcessForm()` method.
2. Checks the value of the ID of `workingLink`. If the customer is editing a link, the ID column of the `workingLink` row contains a value. The script displays the Save Changes and Cancel buttons so that it can be updated, or the customer can cancel their changes. (In order for the `workingLink` to contain a value, the `edit.jsp` page must have updated itself at least once.)

```
<%  
if(workingLink . get(BVJS_PTFLColumnID) != "") {  
%>  
<INPUT TYPE=submit NAME="<%= BVJS_PTFLFieldUpdateLink %>"  
    VALUE="Save Changes">  
<INPUT TYPE=submit NAME="Cancel Changes" VALUE="Cancel">  
<%  
} else {  
. . .
```

3. Sets the value of `formatString`, which later uses the server-side JavaScript to dynamically generate the client-side JavaScript. This happens when the `formatString` is passed to the formatter by the `BVJS_PTFL>EditAll` method. This is the part of the script that displays the fields and buttons need to edit a link.

```
<%
    var formatString =
        "<SCRIPT>selectOptions = selectOptions + " +
        "'<OPTION VALUE=\\'<%= " + BVJS_PTFLColumnID + " %s " +
        "encoding:html:std %>\\'> " +
        "<%= " + BVJS_PTFLColumnName + " %s encoding:html:std
        %>';</SCRIPT>\n";
%>
```



4. Makes sure that the selected information is passed to the `BVJS_PTFLProcessForm()` method correctly. (This section of the script comes after `moveUp` function.)

```
<input type="submit" value="    Edit    " onClick=
    "this.name='Edit'+document.LinkForm.ReorderList
    [document.LinkForm.ReorderList.selectedIndex].value;" >
```

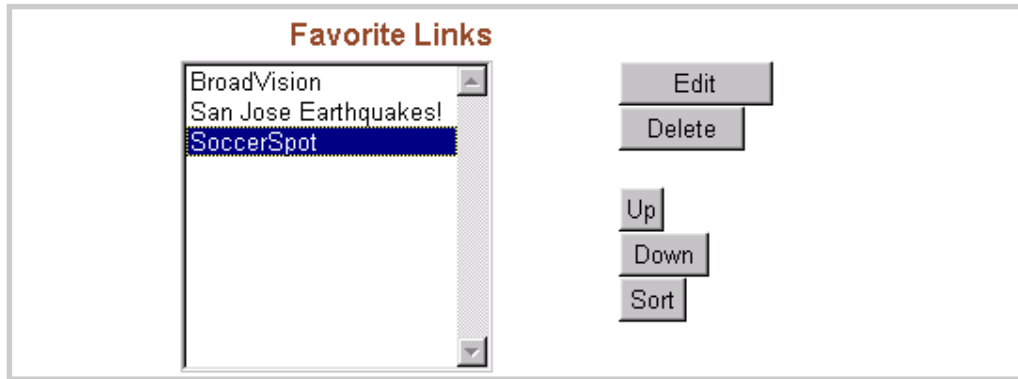
5. When the user clicks the `Save Changes` button the page updates again using the `BVJS_PTFLProcessForm()` method.

## Deleting links

To delete an existing link, this page performs all of the steps in “Updating links” on page 83, then does the following:

1. Sets the value of `formatString` using the server-side JavaScript to dynamically generate the client-side JavaScript. This is the part of the script that identifies the link to be deleted.

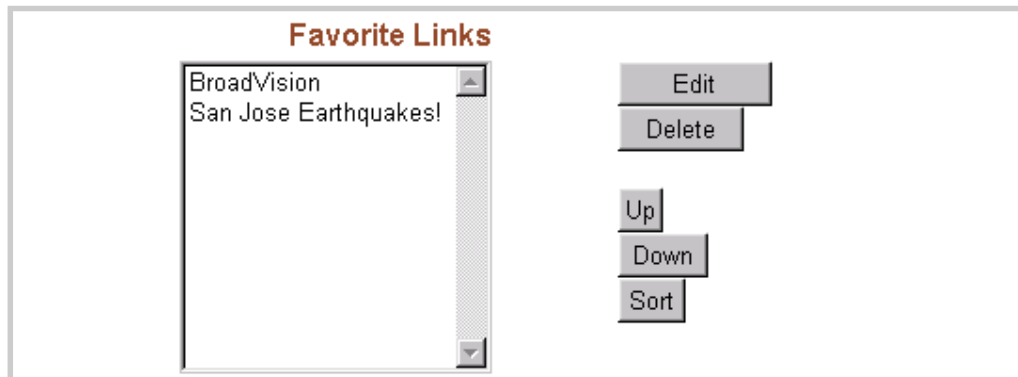
```
<%  
    var formatString =  
        "<SCRIPT>selectOptions = selectOptions + " +  
        "'<OPTION VALUE=\\'<%= " + BVJS_PTFLColumnID + " %s " +  
        "encoding:html:std %>\\'> " +  
        "<%= " + BVJS_PTFLColumnName + " %s encoding:html:std  
                                                %>' ;</SCRIPT>\n";  
%>
```



2. Makes sure that the selected information is passed to the `BVJS_PTFLProcessForm()` method correctly. (This section of the script comes after `moveUp` function.)

```
<input type="submit" value=" Delete "  
onClick="this.name='Delete'+document.LinkForm.ReorderList  
[document.LinkForm.ReorderList.selectedIndex].value;">
```

3. When the user clicks the Delete button the page updates using the `BVJS_PTFLProcessForm()` method.





---

## Sorting links manually

For a customer to sort links, there must be at least two links in the list. When the customer selects a link and clicks either the Up or Down button, this page performs all of the steps in “Updating links” on page 83, then does the following:

1. Checks the value of **workingLink**. If **workingLink** contains any value, the script displays the Save Changes and Cancel buttons so that it can be updated or the customer can cancel their changes. In order for the **workingLink** to contain a value the customer must have selected a link.

```
<%
if(workingLink . get(BVJS_PTFLColumnID) != "") {
%>
<INPUT TYPE=submit NAME="<%= BVJS_PTFLFieldUpdateLink %>"
      VALUE="Save Changes">
<INPUT TYPE=submit NAME="Cancel Changes" VALUE="Cancel">
<%
} else {
%>
```

2. Sets the value of **formatString** using the server-side JavaScript to dynamically generate the client-side JavaScript. This is the part of the script that identifies the link to be added, edited, deleted, or sorted (that is, moved up or down, see <sorting links>).

```
<%
var formatString =
"<SCRIPT>selectOptions = selectOptions + " +
"'<OPTION VALUE=\\'<%= " + BVJS_PTFLColumnID + " %s " +
"encoding:html:std %>\\'> " +
"<%= " + BVJS_PTFLColumnName + " %s encoding:html:std
 %>' ;</SCRIPT>\n";
%>
```

3. Defines the **moveDown()** function. This allows the customer to move the links down in the display list. The **moveDown()** function:
  - a. Sets the value of **selectedIndex** and **maxIndex**.
  - b. Checks the value of **selectedIndex** to see if it is **maxIndex**. If it is, the row is already at the bottom of the list and cannot be moved down any further. The function stops.
  - c. Sets the value of **selectedValue** and **nextValue**. These are used to move the rows in the list of links.
  - d. Initializes the value of **i** to zero. Hidden fields are used to identify the HTML fields as **id1**, **id2**, and so on. The links are re-ordered using these ids. The **i** represents the current HTML field; it could be any HTML field, a row in the list of links or a button. The function iterates

over all of the HTML fields to find the name of the field that has been selected. This function needs to find the selected link and the link below it in order to switch them. (This code is necessary to make the script browser independent.)

```
var i;  
for(i = 0; i < document . LinkForm . elements . length; ++i)  
    if(document . LinkForm . elements[i] . name == selectedValue)  
        ++(document . LinkForm . elements[i] . value);  
    else if(document . LinkForm . elements[i] . name == nextValue)  
        --(document . LinkForm . elements[i] . value);
```

4. Defines the **moveUp()** function. This allows the customer to move the links up in the display list. This function is almost identical to the **moveDown()** function described above. The **moveDown()** function:
  - a. Sets the value of **selectedIndex** and **maxIndex**.
  - b. Checks the value of **selectedIndex** to see if it is **maxIndex**. If it is, the row is already at the top of the list and cannot be moved down any further. The function stops.
  - c. Sets the value of **selectedValue** and **nextValue**. These are used to move the rows in the list of links.
  - d. Initializes the value of **i** to zero. Hidden fields are used to identify the HTML fields as **id1**, **id2**, and so on. The links are re-ordered using these ids. The **i** represents the current HTML field; it could be any HTML field, a row in the list of links or a button. The function iterates over all of the HTML fields to find the name of the field that has been selected. This function needs to find the selected link and the link below it in order to switch them. (This code is necessary to make the script browser independent.)

```
var i;  
for(i = 0; i < document . LinkForm . elements . length; ++i)  
    if(document . LinkForm . elements[i] . name == selectedValue)  
        --(document . LinkForm . elements[i] . value);  
    else if(document . LinkForm . elements[i] . name == previousValue)  
        ++(document . LinkForm . elements[i] . value);
```

5. Sets the value of **formatString**. The server-side JavaScript dynamically generates the client-side JavaScript in addition to reordering the form fields.
6. The customer selects a link to move and clicks either the Up or Down button.
7. Executes the **moveUp()** and **moveDown()** methods, if at all, with the following script:

```
<input type="submit" name"<%= BVJS_PTFLFieldSortManual %>"  
    value=" MoveUp " onClick="moveUp()">  
<input type="submit" name"<%= BVJS_PTFLFieldSortManual %>"  
    value=" MoveDown " onClick="moveDown()">
```

8. When the user clicks the Up or Down button the page updates using the **BVJS\_PTFLProcessForm** method.

---

## Sorting links alphabetically

In order for the customer to sort links, there must already be at least two links in the list. The script performs [Step 1](#) and [Step 2](#) of “[Sorting links manually](#)” on page 89. To sort links alphabetically, this page does the following:

1. Makes sure that the selected information is passed to the `BVJS_PTFLProcessForm()` method correctly. (This section of the script comes after `moveUp` function.)

```
<input type="submit" name="<%= BVJS_PTFLFieldSortAlpha %>"
value="Sort">
```

2. The customer clicks the Sort button. The form updates using the `BVJS_PTFLProcessForm()` method.

---

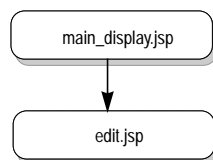
## Reminders

One-To-One Financial includes the capability for customers to maintain a list of date reminders on their personal home page of their financial institution. In this way, customers are encouraged to use their financial site home page as a portal to other sites.

The One-To-One Financial reminder storing capability allows a simple, yet flexible implementation for sending a reminder (an alert) to the customer about an important date. Customers can enter any date that is important to them: anniversary, appointment with the tax consultant, doctor’s appointment, birthday, and so on. The application allows customers to determine when they should be reminded of scheduled dates and how frequently.

The financial institution can specify the layout, color, font and style of the page that holds the reminders, including the presence or absence of buttons, number of buttons, and the text on the buttons.

The following diagram shows the scripts used for adding and managing favorite links. These pages are located in the `$BVLTO1/finance/bvsnfi/scripts/favoritelinks` directory.



This section describes the following:

- “[Viewing reminders](#)” on page 92.
- “[Managing reminders](#)” on page 92.
- “[Adding reminders](#)” on page 95.
- “[Editing reminders](#)” on page 96.
- “[Deleting reminders](#)” on page 98.
- “[Reminders configuration parameters](#)” on page 98

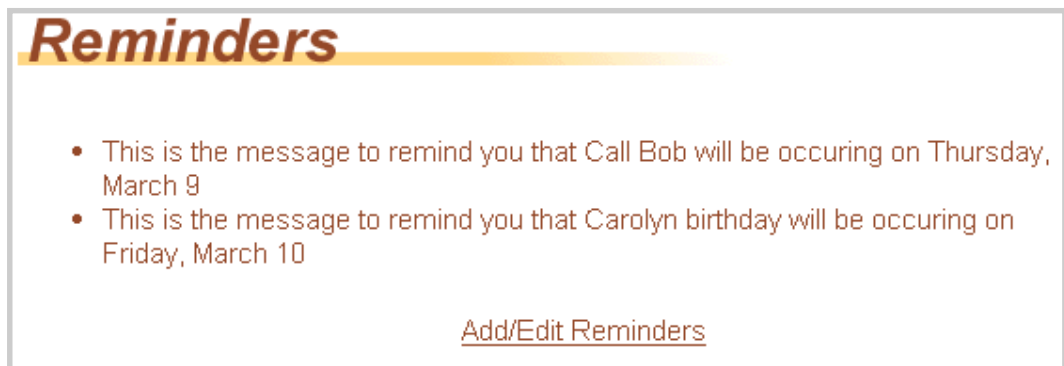
---

## Viewing reminders

The customer home page displays reminders in the upper left area of the page under the heading Reminders. Customers can go to the reminders management page by clicking either the Reminders heading or the Reminders navigation link in the left frame. In either case, the BVSN Financial sample application displays the `$BVT01/finance/bvsnfi/scripts/reminders/main_display.jsp` page. This page does the following:

main\_display.jsp

1. Sets the value of `remindersTable` to the value of `BVJS_PTRMGetPersonalInbox()`. The `BVJS_PTRMGetPersonalInbox()` function returns a `BVI_Table` containing the portion of the customer's alert inbox generated by personal reminders. If the customer's reminders have not triggered an alert, no reminders are displayed on the Reminders page or on the home page. If there are any reminders in the table (`rows > 0`) then the script uses the formatter to display the reminders.



2. This page also contains a link to the `edit.jsp` page.

---

## Managing reminders

As with favorite links, the functionality that allows the customers to add, edit, delete, and modify their reminders is contained in the `edit.jsp` page. The functionality provided by this page is described in the following sections. The beginning of this script applies to all of the following operations. This page does the following:

edit.jsp

1. Initializes the page with `BVJS_PTRMInit()`, which makes sure that the session variables have been set for use in writing the form and for editing the reminders.
2. Sets the value of `earlier` by retrieving the value of the `BVJS_PTRMFieldDaysBefore` constant from the `Request` object. By getting this value the script can determine if the page has been updated. The `BVJS_PTRMFieldDaysBefore` constant holds the value contained in the Start Reminding Me field. This field is passed whenever this page updates, that is, whenever the customer clicks on a button.

```
var earlier = Request . value(BVJS_PTRMFieldDaysBefore);
```

▶ The constants used by the reminders functions are described in “Reminders constants” on page 329.

3. Sets the value of **cancel** by retrieving the value "CancelUpdate" from the **Request** object.

```
var cancel = Request . value("Cancel Update");
```

Initially, this value is null. The value of **cancel** changes only when the customer clicks the Cancel Update button.

4. Sets the value of **dateInputError** to 0.
5. Runs a block of code to extract information from the user input. Because the Edit and Delete buttons are images in the BVSN Financial sample application, they can be assigned a name, but not a value. To work around this restriction, the BVSN Financial sample application appends identification information to the button name. This code segment extracts the value of the Delete and Edit buttons to determine which reminder the customer has selected.

```
if( (earlier != null) && (cancel == null) ) {
  var deleteKeys = Request . subkeys(BVJS_PTRMFieldDelete, true);
  if(deleteKeys . length() > 0) {
    var deleteInfo = deleteKeys . get(0);
    deleteInfo = deleteInfo . slice(BVJS_PTRMFieldDelete . length,
                                   deleteInfo . length - 2);
    Request . add_attribute(BVJS_PTRMFieldDelete, deleteInfo);
  }

  var editKeys = Request . subkeys(BVJS_PTRMFieldEdit, true);
  if(editKeys . length() > 0) {
    var editInfo = editKeys . get(0);
    editInfo = editInfo . slice(BVJS_PTRMFieldEdit . length,
                               editInfo . length - 2);
    Request . add_attribute(BVJS_PTRMFieldEdit, editInfo);
  }

  BVJS_PTRMProcessForm();

  dateInputError |=
    Session . portal . personalReminderError_dateInvalid;
}
```

6. Sets the value of several attributes to the empty string.

```
var workingReminderName = "";
var workingReminderMonth = "";
var workingReminderday = "";
var workingReminderID = "";
```

7. Sets the value of the **today** attribute to a new instance of **BVI\_DateTime**. (The default value is now, the current date and time.)
8. Sets the value of **portal** to the value of **Session . portal**. This is caching for improving performance.

9. Sets the value of **workingReminderDaysBefore** to 14 (the default value for the sample application). This is the variable that holds the value for when to start sending reminders. This variable is also held by the **BVJS\_PTRMFieldDaysBefore** constant.
10. Runs an if/else block that either cancels the object or sets the values to those in the **portal** object. If cancel is not null, that is, the customer has clicked the cancel button, then the script sets the working reminder values to null. If cancel is null, that is, the customer has not clicked the Cancel button, the script proceeds to the next part of the block.

```
if(cancel != null) {
    BVJS_PTRMClearWorkingReminder();
}
```

11. If the value of **earlier** is not null, the script sets the working reminder values to the values in the **portal** object. If earlier is null, the customer has not input any values yet. The script set the value of **workingReminderName** back to the empty string.

```
} else if(earlier != null) {
    workingReminderName = portal . workingReminderName;
    if(workingReminderName != null) {
        workingReminderMonth = portal . workingReminderEventDate .
                                                    month;
        workingReminderDay = portal . workingReminderEventDate . day;
        workingReminderID = portal . workingReminderID;
        workingReminderDaysBefore = portal .
                                                    workingReminderDaysBefore;
    } else
        workingReminderName = "";
}
```

12. Displays the HTML FORM.

Reminder Description	Month	Day	Start Reminding Me		
Carolyn birthday	3	10	Two weeks before	Add Reminder	Cancel

Description	Date	Start Reminding	Edit	Delete
Call Bob	03/09/00	03/08/00		

13. Checks for errors.

## Adding reminders

After following the steps in [“Managing reminders” on page 92](#), this page does the following:

1. Displays the drop-down month selector for the customer to chose a month. This step is the same for adding or editing a reminder.

```
<TD><SELECT NAME="<%= BVJS_PTRMFieldMonth %>">
  <OPTION <%= workingReminderMonth == 1 ? "SELECTED" : "" %>
    VALUE=1> 1
  <OPTION <%= workingReminderMonth == 2 ? "SELECTED" : "" %>
    VALUE=2> 2
  <OPTION <%= workingReminderMonth == 3 ? "SELECTED" : "" %>
    VALUE=3> 3
  <OPTION <%= workingReminderMonth == 4 ? "SELECTED" : "" %>
    VALUE=4> 4
  <OPTION <%= workingReminderMonth == 5 ? "SELECTED" : "" %>
    VALUE=5> 5
  <OPTION <%= workingReminderMonth == 6 ? "SELECTED" : "" %>
    VALUE=6> 6
  <OPTION <%= workingReminderMonth == 7 ? "SELECTED" : "" %>
    VALUE=7> 7
  <OPTION <%= workingReminderMonth == 8 ? "SELECTED" : "" %>
    VALUE=8> 8
  <OPTION <%= workingReminderMonth == 9 ? "SELECTED" : "" %>
    VALUE=9> 9
  <OPTION <%= workingReminderMonth == 10 ? "SELECTED" : "" %>
    VALUE=10> 10
  <OPTION <%= workingReminderMonth == 11 ? "SELECTED" : "" %>
    VALUE=11> 11
  <OPTION <%= workingReminderMonth == 12 ? "SELECTED" : "" %>
    VALUE=12> 12
</SELECT>
</TD>
```

2. Dynamically generates the list of available months. This step also is the same for adding or editing a reminder.

```
<TD><SELECT NAME="<%= BVJS_PTRMFieldDay %>">
<%
for(var i = 1; i <= 31; ++i)
  if(i == workingReminderDay)
    Response . write(" <OPTION SELECTED VALUE=" + i + "> " +
      i + "\n");
  else
    Response . write(" <OPTION VALUE=" + i + "> " + i + "\n");
%>
</SELECT>
</TD>
```

- Looks for the **workingReminderDaysBefore** value. That is, the script checks the drop-down list to see what value the customer has chosen for the Start Reminding Me field.

```
<TD><SELECT NAME="<%= BVJS_PTRMfieldDaysBefore %>">
  <OPTION VALUE=-1>Now
  <OPTION <%= workingReminderDaysBefore == 30 ? "SELECTED" : "" %>
    VALUE=30>Onemonth before
  <OPTION <%= workingReminderDaysBefore == 14 ? "SELECTED" : "" %>
    VALUE=14>Twoweeks before
  <OPTION <%= workingReminderDaysBefore == 7 ? "SELECTED" : "" %>
    VALUE=7>One week before
  <OPTION <%= workingReminderDaysBefore == 1 ? "SELECTED" : "" %>
    VALUE=1>One day before
  <OPTION <%= workingReminderDaysBefore == 0 ? "SELECTED" : "" %>
    VALUE=0>Day of the event
</SELECT>
</TD>
```

- Runs a code block to display the Add button. This same block also displays the Save Changes and Cancel buttons as shown in the procedures below.

```
<%
if( (portal . workingReminderOriginalName == "") ||
    (portal . workingReminderOriginalName == null) ) {
%>
  <INPUT TYPE=submit NAME="<%= BVJS_PTRMfieldAdd %>"
    VALUE="Add Reminder">&nbsp;<input type="submit"
      value="Cancel">
  . . .
```

- When the customer clicks the Add Reminder button, the script updates the HTML form by posting to itself with the [BVJS\\_PTFLProcessForm\(\)](#) method. ([BVJS\\_PTFLProcessForm\(\)](#) runs the [BVJS\\_PTRMAddPersonal\(\)](#) method.)
- If the customer clicks the Cancel button instead, the script updates by posting to itself with the [BVJS\\_PTRMProcessForm\(\)](#) method, but in this case it clears the fields. See [Step 10](#) of the [Managing reminders](#) procedure.

---

## Editing reminders

To edit an existing reminder, this page executes all of the steps in [“Updating links” on page 83](#) and [“Adding links” on page 85](#) through [Step 3](#), (the looking for the **workingReminderDaysBefore** value step) then does the following:

- Displays the Edit button. This code appears near the end of the script in the format string. The excerpt below is only that part concerned with the Edit button.

```
<input type='image' src='/bvsnfi/images/icons/edit.gif'
  width=22 height=22 border=0
  alt='Edit' name=' " + BVJS_PTRMfieldEdit + "<%= ALERT_ID %>:'
  <%= ALERT_NAME %s %>'></td>
```



▶ This is the button name and value that are extracted in [Step 5](#) of the [Managing reminders](#) procedure.

2. Runs else section of this code block to display the Save Changes button. This same block also displays the Add and Cancel buttons as shown in the other procedures.

```

if( (portal . workingReminderOriginalName == "") ||
    (portal . workingReminderOriginalName == null) ) {
    . . .
} else {
%>
    <INPUT TYPE=hidden NAME="<%= BVJS_PTRMFieldWorkingID %>"
        VALUE="<%= workingReminderID %>">
    <INPUT TYPE=submit NAME="<%= BVJS_PTRMFieldUpdate %>"
        VALUE="Save Changes">&nbsp;<input type="submit"
        name="Cancel Update" value="Cancel">
<%
}
%>

```

3. When the customer clicks the Edit button the script updates itself displaying the reminder to be edited in the HTML fields and changing the Add Reminder button to Save Changes. This is the result of the JavaScript in [Step 2](#) above.

## Reminders

Reminder Description	Month	Day	Start Reminding Me		
Quakes Home Opener	4	8	Two weeks before	Save Changes	Cancel

Description	Date	Start Reminding	Edit	Delete
Call Bob	03/09/00	03/08/00		
Carolyn birthday	03/10/00	02/25/00		
Quakes Home Opener	04/08/00	03/24/00		

4. When the customer clicks the Save Changes button, the script updates by posting to itself with the `BVJS_PTFLProcessForm()` method. (`BVJS_PTFLProcessForm()` runs the `BVJS_PTRMUpdatePersonal()` method.)
5. If the customer clicks the Cancel button instead, the script updates by posting to itself with the `BVJS_PTRMProcessForm()` method, but in this case it clears the fields. See [Step 10](#) of the [Managing reminders](#) procedure.


---

## Deleting reminders

To edit an existing reminder, this page executes all of the steps in the [Managing reminders](#) and [Adding reminders](#) procedures through [Step 3](#), (looking for the `workingReminderDaysBefore` value) then does the following:

1. Displays the Delete button. This code appears near the end of the script in the format string. The excerpt below is only that part concerned with the Delete button.

```
<input type='image' src='/bvsnfi/images/icons/delete.gif'  
      width=22 height=22 border=0  
      alt='Delete' name='" + BVJS_PTRMFieldDelete +  
      "<%= ALERT_ID %>:<%= ALERT_NAME %s %>'">
```

 This is the button name and value that are extracted in [Step 5](#) of the [Managing reminders](#) procedure.

2. When the customer clicks the Delete button the script updates itself deleting the selected reminder. (BVJS\_PTFLProcessForm()) runs the [BVJS\\_PTRMDeletePersonal\(\)](#) method.

---

## Reminders configuration parameters

Four `bv1to1.conf` parameters determine how reminders are managed at a site. These parameters determine the pace at which the One-To-One Notifications system processes reminders and how often reminder specifications and reminder messages get cleared from the One-To-One alerts tables and customer inboxes.

Configuration parameter	Description
ReminderRowsToRead	Number of reminder rows to be processed in a single pass by the One-To-One Notifications system. Once this number of rows is processed, the system waits the value of ReminderSleepTime (see next parameter) before it processes the next set of rows. The default is 500 rows.
ReminderSleepTime	Amount of time in seconds to wait between processing passes. The default is 1 second.
ReminderSpecGracePeriod	Determines the day a reminder specification is marked for deletion. The day the reminder is actually deleted depends on how often the deleter job is scheduled to run. For example, if the deleter job is scheduled to run once a week, there could be as much as a six-day delay between the day a reminder specification is marked for deletion and the day it is actually deleted. The default is 15 days.
ReminderMessageGracePeriod	Number of days after creation that reminder messages are marked for deletion. The day the message is actually deleted depends on how often the deleter job is scheduled to run. For example, if the deleter job is scheduled to run once a week (value "7" , there could be as much as a six-day delay between the day a specification is marked for deletion and the day it is actually deleted. The default is 5 days

The following code shows how the reminders configuration parameters are set up in the sample configuration file `$BVLTO1/finance/setup/bvlto1.conf.finance.us`.

```
ReminderRowsToRead="100"  
ReminderSleepTime="1"  
ReminderSpecGracePeriod="15"  
ReminderMessageGracePeriod="5"
```

For more information about deleting reminder specifications and messages, see [“bvptrm\\_personal\\_spec\\_deleter” on page 104](#) and [“bvptrm\\_personal\\_inbox\\_deleter” on page 105](#).

