

# CHAPTER 12

---

## Adding a Discussion Room

### IN THIS CHAPTER:

- Discuss/Application.cfm**—Application.cfm page for the Discuss directory
- Discuss/FullThread.cfm**—Include that returns all messages in a single thread
- Discuss/Index.cfm**—Discussion home page
- Discuss/NewThreadI.cfm**—Input page for creating a new thread
- Discuss/NewThreadP.cfm**—Processing page for creating a new thread
- Discuss/ReplyThreadI.cfm**—Create a response message input
- Discuss/ReplyThreadP.cfm**—Create a response message display
- Discuss/ThreadDisplay.cfm**—Page to display one message



**W**eb-based message boards have become commonplace on many web sites. A message board is a place where your customers can go to converse with other customers. This chapter takes the message-board concept and expands upon it to create a threaded discussion room for your registered users.

---

## Structuring the Data

Discussion rooms on the Web are derived, in their simplest form, from message boards. A message board is a single web page that lists all the comments entered onto the board. There is no correlation between messages. Message boards also do not usually have any form of user tracking. When you add something to the page, it is appended to the bottom of the list. To take this concept to the next level, we want to implement a threaded discussion list.

A threaded discussion list is similar to a newsgroup. A *thread* is a list of messages, typically consisting of a main message and a list of responses to that original message. Messages are related to each other.

We can approach the implementation of our threaded discussion list in either of two ways. We could set it up so that each thread has its own page. If we do this, then a single page becomes similar to a message board, and our discussion list is a group of message boards. A more flexible, and more usable, implementation is to display only a single message on a page. To pull this off, each page has to contain a list of links to all other messages in the thread. This is the method we will use to implement our threaded discussion list.

What information do we want to store to create our discussion room? Well, the root of our discussion room is a single message. A single message contains these characteristics:

- ▶ **DiscussionID** An integer field that contains the primary key of the discussion thread
- ▶ **Subject** A text field that contains the topic of the message
- ▶ **Body** A text field that contains the content of the message
- ▶ **CreationDate** A date field that contains the date the message was created

As always, we need an ID for the table, so we have the DiscussionID field, as shown in Figure 12-1. We also have the topic of the message and the text of the message. The creation date is useful to have. It can be used for sorting or archiving

purposes. It also gives the reader of the discussion thread a sense of the time the discussion was written. You wouldn't want to read a five-year-old discussion thread if you are looking for information on a software bug in the most recent release of ColdFusion.

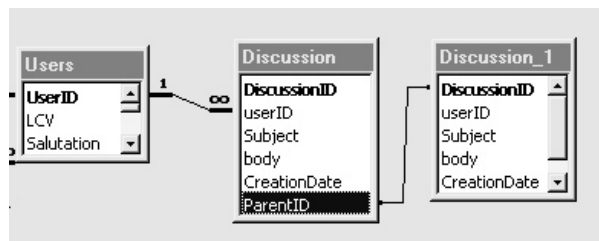
There are two things that we have yet to accomplish. We are making this message board available only to registered users, so we will want to add some user functionality to our table. We also need a way to handle threads. Integrating user functionality into our message table is easy. We have already created a user table for the Magonda application. To attach the user table to the message, we just have to add a UserID field. This is simple enough. Handling threads is a different story.

All messages have similar elements, so we do not need a separate table for responses and messages. We relate the two together by using a self-referencing table. First, let us examine the relationship between a response and a main message. A main message can have many responses to it, whereas a response can have only one main message. This is a one-to-many relationship. We accomplish this by adding a ParentID field into our message table. We can find a document's responses by using a SQL Select statement to find instances in which the ParentID is equal to the current DiscussionID. This method allows us to generate responses that go multiple levels deep. At each level, the response becomes the main page, and we can go deeper to find responses to responses. This implementation gives us great flexibility. To find the top-level document, the ParentID must be zero.

---

## Our Discussion Group Home Page

We'll start by coding the home page for our discussion group. Before examining the code, we have to do some data entry. In Chapter 4, we created database-driven navigation, and in Chapter 8, we rolled out a security scheme on the site. To add



**Figure 12-1** Discussion group table structure

our discussion group link to the navigation bar, we first need to add some data to the navigation table:

NavigationID	NavigationText	ParentID	NavigationLink
33	Discussion	21	Discuss/Index.cfm

Of course, once this data is added, we need to give a group access to it; otherwise, no one will see it. We add these rows to the SecurityGroupNavigation table:

GroupID	NavigationID	Rede	Edit	Delet
2 (Registered)	33	Yes	No	No
1 (Admin)	33	No	Yes	Yes

Now the discussion link will appear to all registered users. Our discussion group home page is shown in Figure 12-2.

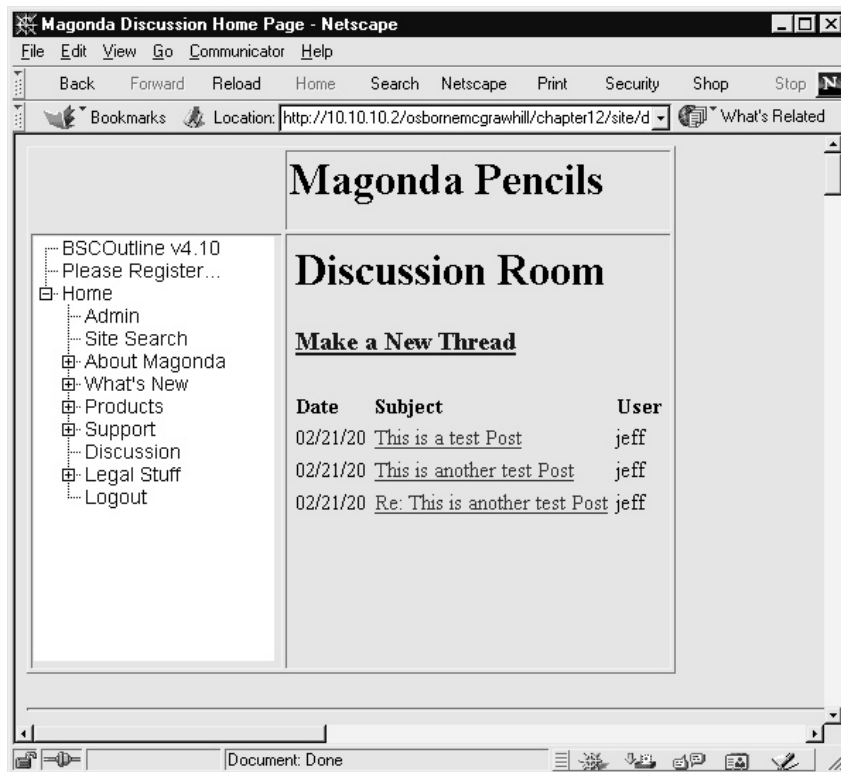


Figure 12-2 Discussion group home page

Next, we will examine the code for our home page. What do we want to display on this page? We want to start with a list of all the discussion threads. A link to create a new thread would be a nice addition. Otherwise, this page does not need more information on it. The code for our home page is shown in Figure 12-3.



### Site/Discuss/Index.cfm

```

<!---
Date: 1/21/01
Creator: Jeff Houser, DotComIt
Description: Discussion Home Page. This page will
             display an option to create a new
             thread and display a list of all
             available threads.

Entering: N/A
Exiting: N/A
Dependencies: N/A
Expecting: N/A
--->

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0
    Transitional//EN">

<HTML>
<HEAD>
  <TITLE>Magonda Discussion Home Page</TITLE>
</HEAD>
<BODY>
<CFQUERY datasource="#application.Datasource#"
           name="GetTopLevelDiscuss">
  SELECT Discussion.*, Users.Username
  FROM Discussion, Users
  WHERE Discussion.ParentID = 0 AND
         Discussion.UserID = Users.UserID
</CFQUERY>

<TABLE border="1">
  <TR>

```

---

**Figure 12-3** Discussion home page

```

<TD align="left" valign="top"></TD>
<TD align="left" valign="top">
  <H1>Magonda Pencils</H1>
</TD>
</TR>
<TR>
<TD align="left" valign="top">
  <!-- Navigation -->
  <CFINCLUDE template="#DirLevel#
    #application.IncludeDir#JavaNav.cfm">
</TD>

<TD align="left" valign="top">
  <TABLE>
  <TR>
  <TD colspan="2">
    <H1>Discussion Room</H1>
  </TD>
  </TR>
  <TR>
  <TD colspan="2">
    <H3><A HREF="NewThreadI.cfm?NavID=#NavID#">
      Make a New Thread</A></H3>
  </TD>
  </TR>
  </TABLE>
  <TABLE>
  <TR>
  <TD><B>Date</B></TD>
  <TD><B>Subject</B></TD>
  <TD><B>User</B></TD>
  </TR>
  <CFOUTPUT query="GetTopLevelDiscuss">
  <TR>
  <TD>#GetTopLevelDiscuss.CreationDate#</TD>
  <TD><A HREF="#DirLevel#Discuss/
    ThreadDisplay.cfm?NavID=#NavID#
    &DiscussionID=
    #GetTopLevelDiscuss.DiscussionID#">
    #GetTopLevelDiscuss.Subject#</A>

```

---

**Figure 12-3** Discussion home page (continued)

```

        </TD>
        <TD>#GetTopLevelDiscuss.Username#</TD>
    </TR>
</CFOUTPUT>
</TABLE>
</TD>
</TR>
</TABLE>
</BODY>
</HTML>

```

---

**Figure 12-3** *Discussion home page (continued)*

We start with a query to get all the main documents. Main documents are ones in which ParentID is set to zero. In this query, we also get the name of the user who has created the message. Next, we output the command to create a new thread. We will examine those pages in the next section. The last part of the page loops over the query and outputs all the main discussion threads, displaying the username, date, and subject of the message.

Because we do not want unregistered users to access this discussion room, I have set up some code in the Application.cfm file to redirect users to the home page if they are not in the registered users group. The code is as follows:



### Site/Discuss/Application.cfm

```

<CFSET Error = false>
<CFLOCK scope="SESSION" timeout="30"
    type="readonly">
    <CFIF #isDefined("session.Groups")#>
        <CFIF not listcontains(session.Groups,'2')>
            <CFSET Error = true>
        </CFIF>
    <CFELSE>
        <CFSET Error = true>
    </CFIF>
</CFLOCK>

<CFIF #Error# is true>
    <CFLOCATION url="#DirLevel#Index.cfm">
</CFIF>

```

In this code segment, we check the Groups session variable to make sure that it contains the necessary GroupID. If it does not, we set an error flag to true. If the error is true, then we automatically redirect the user out of this directory and back to the home page.

Now that we have our discussion home page, we want to examine the page that actually displays a thread. As shown in Figure 12-4, this page consists of two parts. The first part displays the current message. The second part displays all the other messages in the current thread. I coded the second part as an include, and it is shown in Figure 12-5, discussed shortly.



### Site/Discuss/ThreadDisplay.cfm

```

<!---
Date: 1/21/01
Creator: Jeff Houser, DotComIt
Description: Display a Discussion Thread

Entering: N/A
Exiting: N/A
Dependencies: N/A
Expecting: url.DiscussionID = DiscussionID of the
          thread we are displaying
-->

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0
  Transitional//EN">

<HTML>
<HEAD>
  <TITLE>Magonda Discussion Home Page</TITLE>
</HEAD>
<BODY>

<TABLE border="1">
  <TR>
    <TD align="left" valign="top"></TD>
    <TD align="left" valign="top">
      <H1>Magonda Pencils</H1>
    </TD>
  </TR>
  <TR>
    <TD align="left" valign="top">
      <!-- Navigation -->
      <CFINCLUDE template="#DirLevel#

```

---

**Figure 12-4** Code to display a message

```

#application.IncludeDir#JavaNav.cfm">
</TD>

<TD align="left" valign="top">
<TABLE>
<TR>
<TD colspan="2">
<H1>Discussion Room</H1>
</TD>
</TR>
<CFIF isDefined("url.DiscussionID")>
<TR>
<TD><H3>
<A HREF="NewThreadI.cfm">New Thread</A>
</H3></TD>
<TD><H3>
<A HREF="ReplyThreadI.cfm?DiscussionID=
<CFOUTPUT>#DiscussionID#</CFOUTPUT>">
Reply</A></H3>
</TD>
</TR>
</TABLE>
<TABLE>

<CFQUERY
datasource="#application.Datasource#"
name="GetThread">
SELECT Discussion.*, Users.Username
FROM Discussion, Users
WHERE Discussion.DiscussionID =
#url.DiscussionID# AND
discussion.UserID = Users.UserID
</CFQUERY>

<CFOUTPUT query="GetThread">
<TR>
<TD colspan="2"><B>User</B>:
#GetThread.Username#
</TD>
</TR>
<TR>
<TD colspan="2"><HR></TD>
</TR>
<TR>
<TD>

```

---

**Figure 12-4** Code to display a message (continued)

```

        <B>Date</B>: #GetThread.CreationDate#
    </TD>
    <TD>
        <B>Subject</B>: #GetThread.Subject#
    </TD>
</TR>
<TR>
    <TD colspan="2"><HR></TD>
</TR>
<TR>
    <TD colspan="2"><B>Message Body:</B></TD>
</TR>
<TR>
    <TD colspan="2">#GetThread.Body#</TD>
</TR>
</CFOUTPUT>
<TR>
    <TD colspan="2"><HR></TD>
</TR>
<TR>
    <TD colspan="2">Thread</TD>
</TR>
<TR>
    <TD colspan="2">
        <CFINCLUDE
            template="#DirLevel#Discuss/
                FullThread.cfm">
        </TD>
    </TR>
<CFELSE>
    <TR>
    <TD colspan="2">
        You got to this page in error.
    </TD>
    </TR>
</CFIF>
</TABLE>
</TD>
</TR>
</TABLE>

</BODY>
</HTML>

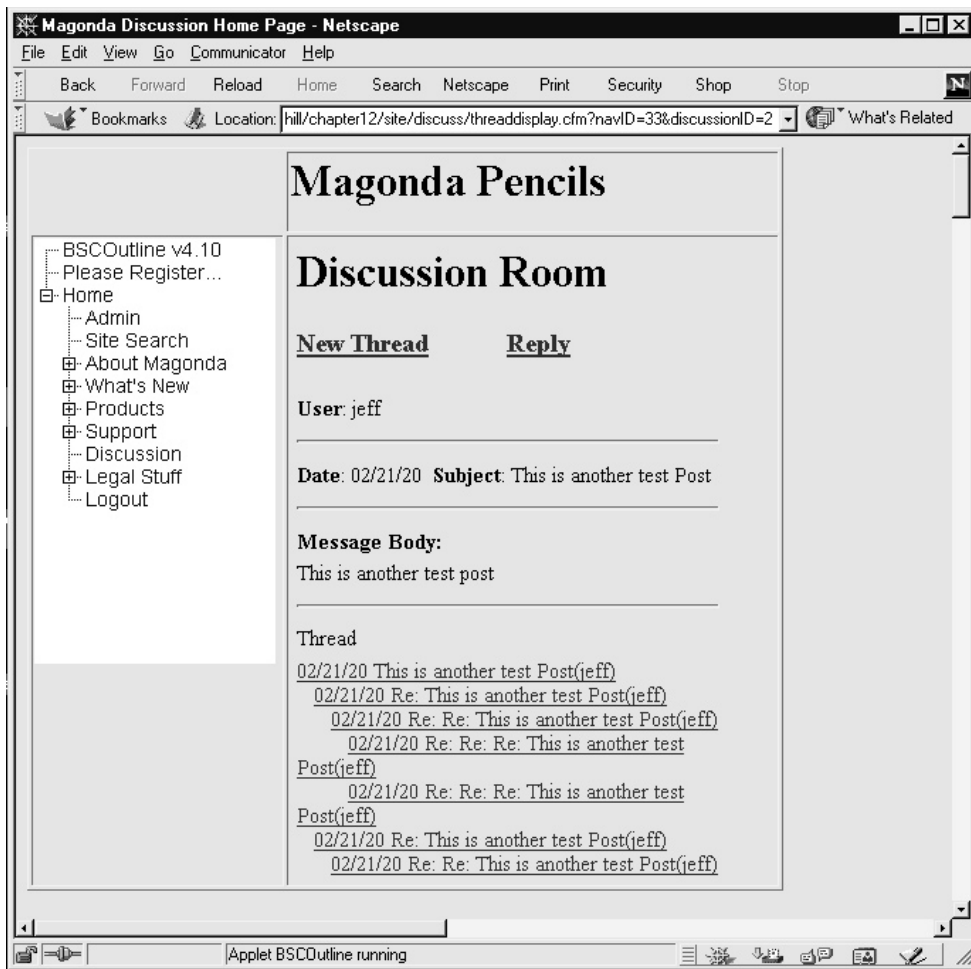
```

---

**Figure 12-4** Code to display a message (continued)

To display the thread, we first need to know which thread we are looking at. A DiscussionID variable must be passed into this template, so we start by checking for its existence. If it exists, we query the database and get the message information. This query grabs all the message data based on the DiscussionID. Otherwise, we output an error message and end the template.

Next, we give the user two command options: create a new thread or reply to the current thread. Then, we loop over the query, displaying all the message information, such as the creation date, the subject, the body, and the name of the user who created it. The template ends by displaying the full thread information, as shown in the following illustration.



We use an include file to create the list of all messages in the thread. The code is listed in Figure 12-5. Because threads can be buried multiple levels deep, the code here is similar to the code that we used when creating our Java navigation. The first step is to find the top-level thread. We copy the current DiscussionID into a temporary variable. We enter a loop and select from the discussion table where the CurrentID is equal to the table's DiscussionID. If the ParentID of our query is equal to zero, we have found the first message in the thread. We can set our Boolean variable to true and carry on with our execution. Otherwise, we set our TempDiscussionID to the current ParentID and loop through once more. This process loops until we find the top-level thread.



### Site/Discuss/FullThread.cfm

```
<!---
Date: 02/04/01
Creator: Jeff Houser, DotComIt
Description: Display a full thread

Entering: N/A
Exiting: N/A
Dependencies: N/A
Expecting: DiscussionID
--->
<CFSET TempDiscussionID = #DiscussionID#>

<!-- loop until the top level is found -->
<CFSET FoundTopLevel = false>
<CFLOOP condition="FoundTopLevel is false">
  <CFQUERY datasource="#application.Datasource#"
    name="GetTopLevel">
    SELECT Discussion.*, Users.Username
    FROM Discussion, Users
    WHERE
      Discussion.DiscussionID = #TempDiscussionID#
      AND Discussion.UserID = Users.UserID
  </CFQUERY>

  <CFIF GetTopLevel.ParentID is 0>
    <CFSET FoundTopLevel = true>
  </CFIF>
  <CFSET TempDiscussionID = #GetTopLevel.ParentID#>
</CFLOOP>
```

**Figure 12-5** Code to display all the messages in a thread

```

</CFLOOP>

<CFSET TempDiscussionID = #GetTopLevel.DiscussionID#>

<CFSET ThreadArray = ArrayNew(1)>
<!-- threadarray is an array of structures
Structure:
DiscussionID NavID for the current level
Level        How many levels deep we are
Subject      text for the link
CreationDate the link
Username     username
When we get to our processing loop, we use the
array like a stack, always putting new info at
the top/front of it and grabbing, processing
whatever is on the top data
-->

<!-- set up all the top level variables -->
<CFLOOP query="GetTopLevel">
<CFSCRIPT>
TempStructure = structnew();
structinsert(TempStructure,"DiscussionID",
             GetTopLevel.DiscussionID);
structinsert(TempStructure,"Level","");
structinsert(TempStructure,"Subject",
             GetTopLevel.Subject);
structinsert(TempStructure,"CreationDate",
             GetTopLevel.CreationDate);
structinsert(TempStructure,"Username",
             GetTopLevel.Username);
arrayprepend(ThreadArray,TempStructure);
</CFSCRIPT>
</CFLOOP>

<CFSET FinalThread = "">
<CFLOOP condition="arraylen(ThreadArray) is
not 0">
<CFSET FinalThread = FinalThread &
ThreadArray[1].level & "<A HREF="" &

```

---

**Figure 12-5** Code to display all the messages in a thread (continued)



```

        GetNextLevelDown.DiscussionID);
    structinsert(TempStructure,"Level",NextLevel);
    structinsert(TempStructure,"Subject",
        GetNextLevelDown.Subject);
    structinsert(TempStructure,"Creationdate",
        GetNextLevelDown.Creationdate);
    structinsert(TempStructure,"Username",
        GetNextLevelDown.Username);
    arrayprepend(ThreadArray,TempStructure);
</CFSCRIPT>
</CFLOOP>
</CFLOOP>

<CFOUTPUT>
    #FinalThread#
</CFOUTPUT>

```

---

**Figure 12-5** Code to display all the messages in a thread (continued)

Next, we create a stack structure. A stack has a first-in, last-out methodology. The first item that we put in the stack is the last one we take out of it. We used a stack when we created our Java navigation. Once again, we are using an array of structures for our stack. The structure contains the current DiscussionID, the subject of the message, the creation date, the username of the message creator, and a Level variable. The Level variable is used to keep track of how many levels deep we are. I filled the Level variable with HTML nonbreaking spaces: &nbsp;. We'll use the nonbreaking spaces to indent replies underneath their main document.

We enter into a loop until the stack size is zero, or until `arraylen(ThreadArray)` is not 0. We use a text variable to put together the list of links that will result in our final thread. Following that, we set up our variable in preparation for the next level of depth. We save the current DiscussionID, create the next-level variable, and remove the top element of the stack. Then, we perform a query to get all the responses to our current discussion message. This query selects all the messages where the ParentID is equal to the current DiscussionID. We add all of them to the top of the stack. We continue through the loop until we are out of messages in the current thread.

The final step of this template is to output the FinalThread variable. This displays the full thread, starting from the main document. With our display done, one topic remains to be discussed: creating messages in our board. First we'll look at creating a new thread, and then we'll look at how to create responses to that thread.

## Creating a New Discussion Thread

Creating a new discussion thread is a two-step process:

1. Create an input page to get the user's input.
2. Create a processing page to add the new message into the database.

We will start by examining the input page.

We need four pieces of data to create a new discussion page. The first is the UserID, which is stored in the session.UserID variable. The second is the creation date, which we can generate when we need it by using ColdFusion functions. The remaining two items are the ones that we need to get from the user. They are the subject and the body of the message. The code for the input page is shown in Figure 12-6.



### Site/Discuss/NewThreadI.cfm

```
<!---
Date: 11/18/00
Creator: Jeff Houser, DotComIt
Description: Create a new thread

Entering: N/A
Exiting: NewThreadP.cfm
Dependencies: N/A
Expecting: N/A
--->

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0
  Transitional//EN">

<HTML>
<HEAD>
  <TITLE>
    Magonda Discussion - Make a New Thread
  </TITLE>
</HEAD>
<BODY>

<TABLE border="1">
```

**Figure 12-6** *Making a new thread input page*

```

<TR>
  <TD align="left" valign="top"></TD>
  <TD align="left" valign="top">
    <H1>Magonda Pencils</H1>
  </TD>
</TR>
<TR>
  <TD align="left" valign="top">
    <!-- Navigation -->
    <CFINCLUDE template="#DirLevel#
      #application.IncludeDir#JavaNav.cfm">
  </TD>

  <TD align="left" valign="top">
    <TABLE>
      <TR>
        <TD><H1>Discussion Room</H1></TD>
      </TR>
      <FORM action="<CFOUTPUT>#DirLevel#</CFOUTPUT>
        Discuss/NewThreadP.cfm" method="post">
        <TR>
          <TD><B>Subject</B></TD>
        </TR>
        <TR>
          <TD><Input type="Text" name="Subject"></TD>
        </TR>
        <TR>
          <TD><B>Body</B></TD>
        </TR>
        <TR>
          <TD>
            <textarea name="body" cols="65" rows="20">
            </TEXTAREA>
          </TD>
        </TR>
        <TR>
          <TD>
            <input type="Submit" name="Submit">
            <input type="Reset" name="Reset">
          </TD>
        </TR>
      </TABLE>

```

---

**Figure 12-6** *Making a new thread input page (continued)*

```

    </FORM>
  </TABLE>
</TD>
</TR>
</TABLE>
</BODY>
</HTML>

```

---

**Figure 12-6** *Making a new thread input page (continued)*

This page is little more than a simple HTML form. The form accepts a standard input box for the subject and a textarea block for the body. The form submits to the thread processing page, `NewThreadP.cfm`. There isn't much else to say about this form, so let's move on to the processing page, shown in Figure 12-7.



### Site/Discuss/NewThreadP.cfm

```

<!---
Date: 11/18/00
Creator: Jeff Houser, DotComIt
Description: Create a new thread

Entering: NewThreadI.cfm
Exiting:
Dependencies: N/A
Expecting: form.Subject
         form.Body
--->

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0
  Transitional//EN">

<HTML>
<HEAD>
  <TITLE>
    Magonda Discussion - Make a New Thread

```

---

**Figure 12-7** *Making a new thread processing page*

```

</TITLE>
</HEAD>
<BODY>

<TABLE border="1">
  <TR>
    <TD align="left" valign="top"></TD>
    <TD align="left" valign="top">
      <H1>Magonda Pencils</H1>
    </TD>
  </TR>
  <TR>
    <TD align="left" valign="top">
      <!-- Navigation -->
      <CFINCLUDE template="#DirLevel#
        #application.IncludeDir#JavaNav.cfm">
    </TD>

    <TD align="left" valign="top">
      <TABLE>
        <TR>
          <TD><H1>Discussion Room</H1></TD>
        </TR>
        <TR>
          <TD>
            <CFIF isdefined("form.Subject") and
              isdefined("form.Body")>
              <CFLOCK type="READONLY" scope="SESSION"
                timeout="30">
                <CFQUERY
                  datasource="#application.Datasource#"
                  name="GetTopLevelDiscuss">
                  INSERT INTO Discussion (UserID, Subject,
                    Body, CreationDate, ParentID)
                  VALUES(#session.UserID#,
                    '#form.Subject#', '#form.Body#',
                    '#dateformat(createodbcdate(
                      now()), "mm/dd/yyyy")#', 0)
                </CFQUERY>

```

---

**Figure 12-7** *Making a new thread processing page (continued)*

```
</CFLOCK>
  Thanks for your submission.
<CFELSE>
  You got here in error.<BR>
</CFIF>
</TD>
</TR>
</TABLE>
</TD>
</TR>
</TABLE>
</BODY>
</HTML>
```

---

**Figure 12-7** *Making a new thread processing page (continued)*

Two values are expected upon entry into this page. The first is the subject, and the second is the body. These are the two items passed on from the input page. The only other item worthy of note on this page is the insert statement. It is a standard SQL insert statement, drawing information from the various sources. We get the UserID from a session variable. The subject and body of the message come from the form submission. We create the date with the createodbcdate and now functions. Since we have our datasource set up as an OLE datasource, the createodbcdate will not work. The function only works with ODBC datasources. We use the dateformat function to put the ODBC date into native access format. Finally, we give this message a ParentID of zero, to distinguish that this message is a main document, and not a response.

---

## Creating Responses to a Thread

The code to create responses on a thread, shown in Figure 12-8, is similar to the code for creating a main thread. The main difference is that we need the DiscussionID of the response's parent. The links to create a response pass this ID as a URL variable. If the variable is defined, we continue execution. If the variable is not defined, we give the user an error message and end the template processing.



## Site/Discuss/ReplyThread1.cfm

```

<!---
Date: 11/18/00
Creator: Jeff Houser, DotComIt
Description: Create a new thread

Entering: N/A
Exiting: NewThreadP.cfm
Dependencies: N/A
Expecting: url.DiscussionID
--->

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0
  Transitional//EN">

<HTML>
<HEAD>
  <TITLE>
    Magonda Discussion - Make a New Thread
  </TITLE>
</HEAD>
<BODY>

<TABLE border="1">
  <TR>
    <TD align="left" valign="top"></TD>
    <TD align="left" valign="top">
      <H1>Magonda Pencils</H1>
    </TD>
  </TR>
  <TR>
    <TD align="left" valign="top">
      <!-- Navigation -->
      <CFINCLUDE template="#DirLevel#

```

---

**Figure 12-8** *Reply input page*

```

    #application.IncludeDir#JavaNav.cfm">
</TD>

<TD align="left" valign="top">
<TABLE>
<TR>
<TD><H1>Discussion Room</H1></TD>
</TR>
<CFIF isdefined("url.DiscussionID")>

<CFQUERY
    datasource="#application.Datasource#"
    name="GetThread">
SELECT Discussion.*
FROM Discussion
WHERE Discussion.DiscussionID =
    #url.DiscussionID#
</CFQUERY>

<FORM
    action="<CFOUTPUT>#DirLevel#</CFOUTPUT>
        Discuss/ReplyThreadP.cfm"
    method="post">
<CFOUTPUT query="GetThread">
<INPUT type="Hidden" name="DiscussionID"
    value="#url.DiscussionID#">
<TR>
<TD><B>Subject</B></TD>
</TR>
<TR>
<TD>
    <Input type="Text" name="Subject"
        value="Re: #GetThread.Subject#">
    </TD>
</TR>
<TR>
<TD><B>Body</B></TD>
</TR>
<TR>
<TD>

```

---

**Figure 12-8** *Reply input page (continued)*

```

        <TEXTAREA name="Body" cols="65"
            rows="20"></TEXTAREA>
    </TD>
</TR>
<TR>
    <TD>
        <input type="Submit" name="Submit">
        <input type="Reset" name="Reset">
    </TD>
</TR>
</CFOUTPUT>
</FORM>
<CFELSE>
    <TR>
        <TD>You got to this page in error</TD>
    </TR>
</CFIF>
</TABLE>
</TD>
</TR>
</TABLE>
</BODY>
</HTML>

```

---

**Figure 12-8** *Reply input page (continued)*

The first step of our continued execution is to perform a query that retrieves the parent message of our new reply. We output a form similar to our new thread form. We add a hidden field for the ParentID, and prefill the Subject field. Both of these pieces of information come from the query. Using a standard in message responses, I append the “Re:” to the previous message’s subject, although there is nothing preventing the user from changing the subject line.

The input template, as expected, submits to a processing template, the code for which is shown in Figure 12-9. Once again, this code for the reply processing page is similar to the code for the main discussion processing page. We are expecting three variables to be defined upon entry into this page. All are passed in from the form: Subject, Body, and DiscussionID. If they are defined, we stop processing and display an error message. If they do not exist, we proceed with our insert statement.



## Site/Discuss/ReplyThreadP.cfm

```

<!---
Date: 11/18/00
Creator: Jeff Houser, DotComIt
Description: Create a reply to an existing thread

Entering: NewThreadI.cfm
Exiting:
Dependencies: N/A
Expecting: form.Subject
          form.Body
          form.DiscussionID
--->

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0
  Transitional//EN">

<HTML>
<HEAD>
  <TITLE>
    Magonda Discussion - Make a New Thread
  </TITLE>
</HEAD>
<BODY>

<TABLE border="1">
  <TR>
    <TD align="left" valign="top"></TD>
    <TD align="left" valign="top">
      <H1>Magonda Pencils</H1>
    </TD>
  </TR>
  <TR>
    <TD align="left" valign="top">
      <!-- Navigation -->
      <CFINCLUDE template="#DirLevel#
        #application.IncludeDir#JavaNav.cfm">

```

---

**Figure 12-9** Code for reply processing page

```

</TD>

<TD align="left" valign="top">
  <TABLE>
    <TR>
      <TD><H1>Discussion Room</H1></TD>
    </TR>
    <TR>
      <TD>
        <CFIF isdefined("form.Subject") and
          isdefined("form.Body") and
          isdefined("form.DiscussionID")>
          <CFLOCK type="READONLY" scope="SESSION"
            timeout="30">
            <CFQUERY
              datasource="#application.Datasource#"
              name="GetTopLevelDiscuss">
              INSERT INTO discussion (UserID, Subject,
                Body, CreationDate, ParentID)
              VALUES(#session.UserID#,
                '#form.Subject#', '#form.Body#',
                '#dateformat(createodbcdate(
                  now()), "mm/dd/yyyy")#',
                #form.DiscussionID#)
            </CFQUERY>
            </CFLOCK>
            Thanks for your submission.
          <CFELSE>
            You got here in error.<BR>
          </CFIF>
        </TD>
      </TR>
    </TABLE>
  </TD>
</TR>
</TABLE>
</BODY>
</HTML>

```

---

**Figure 12-9** Code for reply processing page (continued)

The SQL insert statement is a standard insert statement. We are grabbing information from multiple places for the insert statement. We use a session variable to get the UserID, and create the date dynamically, just as we did for the main document. The Subject, Body, and ParentID come from the form. Since we are accessing a session variable, the query is wrapped in a CFLOCK. The processing continues, with a confirmation message, and then the template is done.

---

## What to Take Away from This Chapter

In this chapter, we have done the following:

- ▶ Created a threaded discussion room
- ▶ Made our discussion room available only to registered users

Before leaving our Internet site to focus on some intranet applications, we want to examine user customization in the next chapter, which shows you how to allow each individual user to customize the information that they see on the site.