

# MOVING TO ENTERPRISE SQL

U.S. \$8.99 (Canada \$9.99)

# COLD FUSION Developer's Journal

ColdFusionJournal.com

April, 1999 Volume: 1 Issue: 2

**From the Editor**  
**Hitting the Ground Running**  
Chad Sittler page 5

**ColdFusion Tag Gallery**  
**A Simple Custom ColdFusion Tag**  
Jeb Beasley page 16

**Slick Tricks**  
**Hidden Frames and JavaScript**  
Dan Chick page 32

**Tips & Techniques**  
**Form to Mail Applications**  
Jim Esten page 38

**CFDJ News**  
page 42

**User Group News**  
page 15

**A Look Inside ColdFusion 4.0 Developer Enhancements**  
Charles Arehart page 44

**SYS-CON PUBLICATIONS**



**CFDJ**  
**FEATURE**  
page 10

## <BF on CF>: Caching in on Performance

*When used properly, query caching can dramatically improve application performance and response time*



6

Ben Forta

## CFDJ Special Feature: Introduction to the Problem

*Building collaborative workspaces on the Web*



10

Hal Helms

## Feature: So You've Outgrown Access Part 1

*Making the move to enterprise-capable SQL Server*



18

Richard Schulze

## Feature: Who Says You Can't Take It with You?

*Keeping your e-mail accessible when you're away from your personal computer*



26

Rich Rosen

## CFDJ Feature: What's In It for Me?

*Taking advantage of new features in ColdFusion 4.0*



34

Michael J. Murdy

## Product Review: NetObjects Fusion 4.0

*Designing great-looking Web pages couldn't be easier with this invaluable tool*



48

Shawn Cowman

## IMHO: New Business Models on the Web

*How to get to the next level from here*



50

Jeremy Allaire

# Who Says You Can't Take It With You?

*How to keep your e-mail accessible  
— even when you're not*

by Rich Rosen

Have you ever been off on a trip, away from your home or office computer, and wanted (no, needed!) to check your e-mail?

Or have you ever been stuck at a location where firewalls or other network limitations prevented you from accessing your mail or sending mail to others?

E-mail has become a lifeline for many people. For some, it's the main mechanism for staying in touch with others. (Many have been known to check their e-mail more frequently than they check their voicemail or answering machine.) For an increasing number of people, being able to access e-mail is not just a luxury, it's an absolute necessity.

How do you check your e-mail when you don't have access to your own computer? Of course, you need a computer with an Internet connection, but that's just the beginning. In the olden days, before the existence of the Web, it was rather simple. You could telnet to your UNIX host and run an e-mail program like elm or pine.

Nowadays, the process is much more complicated. Most people use POP3-based clients like Eudora, Outlook or Netscape Messenger to retrieve and send e-mail, including attachments. If you don't want to go to the bother of setting yourself up as a "user" of one of these e-mail client programs on someone else's computer, or if you simply can't make use of such programs because a firewall blocks POP3 traffic, another solution is needed.

A Web-based e-mail system is a perfect solution. Users can connect to a Web site



that allows them to login, supply their e-mail password (and other relevant information) and view a list of messages for them to read. Unlike the typical POP3 client program that deletes your messages from the server as they're downloaded, this application would leave your messages on the server so that you'd have access to them when you got back to your home base. In addition, you could download the messages you wanted to view selectively, skipping spam and other unwanted e-mail.

This article describes how I used ColdFusion to build a Web-based e-mail system called popART, for "POPmail Access and Retrieval Tool." (I considered calling it "popTART," but feared my lawyer would get a phone call from Battle Creek, Michigan.) The goal was to create a robust, full-featured application for people who are used to more sophisticated e-mail clients like Netscape Messenger, Outlook or Eudora. It may not replace these programs, but it can serve as a viable alternative when they're not available, for whatever reason.



## Why Reinvent the Wheel?

The question arises: Why build a Web-based e-mail system at all? Many portal services already provide a variant of this kind of service. You can use these services to access your e-mail easily over the Web, provided you sign up for an e-mail address on their site. Some of them even let you access mail from your home or office e-mail addresses. But outgoing mail sets your e-mail address to "so-and-so@portal-mail.com" rather than to your original e-mail address. Unless you're looking for anonymity (or extra work for yourself), why get yet another e-mail address?

People who just want access to their home or office e-mail simply need a system that will provide them with that access without requiring them to sign up for a new e-mail address or use proprietary software.

Such a system would be an extremely useful, generalized tool for anyone on the Net to use. It could also be configured as a customized application for a particular group of people who get their e-mail from one particular server, such as the employees of a company or the customers of an ISP.

## Why Use ColdFusion?

ColdFusion is a perfect platform choice for building a Web-based e-mail system. Two of ColdFusion's tags, <CFPOP> and <CFMAIL>, are specifically tailored for this very sort of task. But throwing together a ColdFusion module that uses these tags doesn't necessarily make it a viable e-mail system. The key is to maintain "state" between pages so the necessary connection information is known at all

times without requiring the retransmission of that information every time the user goes from page to page. (Since the password is part of that connection information, ideally it should be transmitted as little as possible -- and preferably not as part of a URL query string!)

I've provided snippets of ColdFusion code from some of the core modules to illustrate the process of building this application. However, these snippets are simplified versions of the code that I eventually used in my own application, and additional work would be required to create a solid and robust e-mail system based on them.

## Requirements

The basic requirements for an application like this are as follows:

1. *A login page.* This page would provide a place for users to enter all the information necessary to connect them to their POP mail server. This would include their POP user name, their password and the name/address of their POP mail server.

(This page doesn't need to be a ColdFusion module. It can be a simple HTML page with a form on it. The form's ACTION parameter should point to a ColdFusion module that uses the <CFPOP> tag to connect to the server to retrieve mail, using the POST method so that sensitive information isn't included in the URL. The password would be entered here and used as the basis for secure access to e-mail throughout the application, but wouldn't be retained or recorded.)

2. *A message list page.* Once connected, a page should be displayed that lists the "headers" of the messages waiting for you on the server. The header is the part of an e-mail message containing "metadata" information about the message, such as its subject, the address of the person who sent it to you and when it was sent. You'll use the <CFPOP> tag to return the headers of all messages and format them as an HTML table with links to the "bodies" of the individual messages.
3. *A message display page.* Each entry on the message list page should include a link to a page that will display the contents ("body") of that individual message. You can use the <CFPOP> tag here to retrieve the header and body of an individual message.
4. *A means of replying to a message.* By clicking on a link found on the message display page, a user should be able to send a reply to the message. Ideally, similar links should be available to do "reply all" (reply to everyone who got or sent the message) and "forward" (send the message on to someone else). The application can open a window using JavaScript that includes a form with a <TEXTAREA> for entering the response message. This form can link to a page that makes use of the <CFMAIL> tag to actually send the e-mail.
5. *A means of generating a new message.* Similarly, there should be a way to create a new e-mail message from scratch. This function should be usable from everywhere in the application. If you play your cards right, you can reuse some of the functionality you built for replying to messages.

In addition, it would be useful for the application to have a means of viewing or downloading attachments to messages, of deleting messages and of retrieving new mail from the server. This article won't cover these additional features.

## Use of Session Variables

A fundamental part of this application is its use of ColdFusion's session variables to maintain "state" as the user jumps from page to page. Under the covers, session variables make use of cookies to identify a pool of values associated with a given application. For this application, the session variables hold the POP server connection information (user name, password, etc.) so that they can persist across pages. This means the application doesn't repeatedly retransmit this information between the ColdFusion server and the browser (though it does transmit the information between the ColdFusion server and the user's POP mail server for each request).

The global ColdFusion module Application.cfm should be used to establish the session variable scope. It can also be used to set other global parameters.

```
<CFAPPLICATION NAME="popart"
SESSIONMANAGEMENT="Yes"
SESSIONTIMEOUT=600>
<CFSET SMTPHost = "smtphost.myisp.com">
```

Once session variables are set, they don't need to be included as parameters in links to subsequent pages, either within the query string portion of the URL (for GET) or within the standard input (for POST). The less this information (which includes the password) is transmitted across the wire, the better.

## POP3 Protocol and the <CFPOP> Tag

To understand the structure of the application you're building, you need some basic knowledge of the POP3 protocol used to retrieve e-mail on the Internet (see Figure 1).

A very simplified POP3 session goes something like this. By telnetting to port 110, you can "talk" to the POP3 server directly. After the server acknowledges your connection, you can send a line that says "USER username" (substituting your user name, of course). The server will respond by telling you that you need a password, so you send another line that says "PASS password." Once you have passed these authorization tests, you can interrogate the POP3 server about your e-mail by sending additional commands.

The STAT command will tell you how many e-mail messages are waiting. You can use that number as an upper bound for successive applications of the RETR or TOP command, which can retrieve individual messages (header and body) or just the headers from messages. If desired, the DELE command can permanently delete specific messages from the server. (The deletion becomes permanent only when you "commit" by explicitly entering a QUIT command. Messages won't be purged if a session is terminated prior to issuing the QUIT command.)

- **USER username** = Enter user name
- **PASS password** = Enter password
- **STAT** = Obtain number of messages
- **RETR n** = Get message #n
- **TOP n 1** = Get header and first line of message #n
- **DELE n** = Delete message #n
- **QUIT** = End POP session and commit mailbox changes

Figure 1: Basic POP3 commands chart



Figure 2: The login page

ColdFusion's <CFPOP> tag can perform sophisticated operations by automatically grouping these commands together. It returns results in a manner similar to <CFQUERY> or <CFDIRECTORY>, so "rows" returned can be iteratively processed via the <CFOUTPUT> tag.

The <CFPOP> tag's required parameters are a USERNAME, a PASSWORD, a SERVER name, an ACTION and a NAME for the <CFPOP> query, which will be used by subsequent <CFOUTPUT> tags. Optional parameters include the MESSAGENUMBER (which could be a range or list of numbers). If the MESSAGENUMBER isn't supplied, it's assumed that the requested operation applies to all messages rather than to just one.

The ACTION parameter can be "getall" (get the entire message including header), "getheaderonly" (get just the header) or "delete" (delete the message from the server). Using ACTION="getheaderonly" without supplying a MESSAGENUMBER returns all of the headers. This format of the <CFPOP> tag is used on the message list page. Alternatively, using ACTION="getall" and supplying a MESSAGENUMBER will return the entire contents (header and body) of a single message. This format is used on the message display page.

## The Login Page

The HTML form on the login page will probably need four entry fields: POP user name, POP user password, POP server name or IP address and the user's e-mail address (see Figure 2). In an ideal world, the POP user name and the server name could be derived from the e-mail address. (e.g., "joe@myisp.com" would have a POP user name of "joe" and a POP server name of "myisp.com"). This isn't always the case. Some ISPs give their customers POP user names that are different from the name in their e-mail address, and more often than not the POP server isn't simply "myisp.com," but rather a specific system name like "mailhost.myisp.com" or "popserver.myisp.com".

This page could be customized so that some parameters are hard-coded. This is useful if you're designing this application for employees of a company or customers of an



Figure 3: The main popART window

ISP, all of whom use the same POP3 server and all of whom have their e-mail addresses associated with the same domain. You can also create defaults to use if some information isn't entered. For example, the name and domain in the e-mail address can act as the default POP user name and POP server name if those fields aren't supplied.

## Message List Page

My version of this application, popART (see Figure 3), makes use of a master page containing three frames. A frame-based architecture can be more efficient, cleaner and more user-friendly because there's less need to use the "Back" button to navigate between the message list and an individual message.

The top frame is a title page containing the popART logo, global buttons used by the application and an ad link generated by an ad server. The middle frame contains the message list page that displays the headers of the messages waiting on the server. The bottom frame starts out blank, but will contain the message display page when individual messages are selected for reading.

If you use a similar design, the form on the login page should connect to this master page. Otherwise it can connect directly to the message list page. First, the parameters passed from the login form are used to set the values of session variables.

```
<CFSET Session.user = #user#>
<CFSET Session.password = #password#>
<CFSET Session.servername = #servername#>
<CFSET Session.address = #address#>
```

Once these values are set, the <CFPOP> tag is used to retrieve the message headers. The output of this tag acts much like a <CFQUERY>, as you'll see below.

```
<CFPOP NAME="email"
ACTION="getheaderonly">
```

```
PORT=110 TIMEOUT=600
USERNAME="#Session.user#"
PASSWORD="#Session.password#"
SERVER="#Session.servername#">
```

Note that the message bodies (and attachments) aren't retrieved as part of this process, just the headers. This should make the process of retrieving the list faster, but it also affords the user some additional flexibility. If you can see the list of messages before downloading them in their entirety, you can selectively choose *not* to download messages that you don't want to see. You can appreciate this if you've ever used a POP3 client program and waited an eternity for several long spam messages or messages with attachments to download in order to read the one genuinely important message that followed.

First, the parameters passed are used to set the values of session variables.

```
<CFSET Session.user = #user#>
<CFSET Session.password = #password#>
<CFSET Session.servername = #servername#>
<CFSET Session.address = #address#>
```

The message headers are displayed as part of an HTML table. (See Listing 1 for the code snippets from this module.) The first line is for column headings (From, Subject, Received). Subsequent lines are generated by the <CFOUTPUT> tag associated with the query named "email". There are columns for message number, sender (from), subject and date sent. The number displayed in the message number column is a link to the message.cfm module (see Listing 2), passing the message number as a parameter.

## Message Display Page

The message.cfm module's primary function is to display the message body (see Figure 4). It begins this process by retrieving the individual message using the <CFPOP> tag.

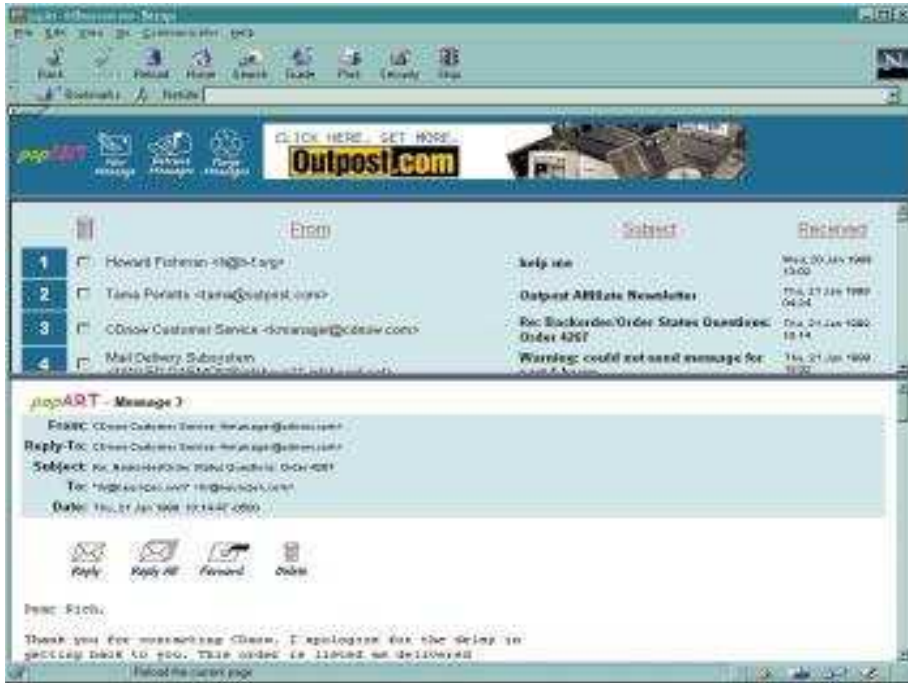


Figure 4: The main popART window with message display

```
<CFPOP NAME="message" ACTION="getall"
PORT=110 TIMEOUT=600
USERNAME="#Session.user#"
PASSWORD="#Session.password#"
SERVER="#Session.servername#"
MESSAGENUMBER="#num#">
```

It then redisplay the header information. Following the header, the body of the message is displayed. You could do this using the "<PRE>" tag, but long lines would roll off the edge of the screen. Instead, enclose the body text in a table, explicitly

use a monospaced font, and append hard line breaks in the message body with "<BR>" tags.

### Reply Function

This would be sufficient if all you wanted to do is display the message. But this module is also supposed to provide a mechanism for replying to the displayed message. To accomplish this task, you must first derive pertinent information from the original message.

You derive the subject and prepend the string "Re:" to it (if it isn't already there). You also derive the reply address, which is the "Reply-to" field (if it exists) or the "From" field.

Finally, you build a copy of the message body, indent each line with ">" and prepend the whole thing with an introductory line. ("On December 31, 1999, so-and-so@somewhere.com wrote:") The JavaScript variable "x" is set to the content of this bodycopy field. The contents of this variable can be retrieved by the subsequent module compose.cfm by referring to "window.opener.x". This is actually far less cumbersome than trying to pass this field as a parameter.

There must also be a link enabling the user to compose a response. In popART, I use an image for this link. The image links to another JavaScript function, com-

# Fusion FX


[www.fusionfx.com](http://www.fusionfx.com)



- designated as a directory that can be seen via the Web server. In this way, attachments can be viewed or downloaded only through this application, by users authorized to see only their own e-mail attachments.
5. Most of the existing e-mail client programs automatically highlight hyperlinks, such as URLs and e-mail addresses. By clicking on these hyperlinks, you can open up new browser windows or initiate e-mail messages. This is definitely a nice feature to have in an e-mail tool, but it's very tricky to figure out where such links begin and end (especially when they end in commas, periods, brackets or other punctuation). ColdFusion's REReplace function is powerful enough to do a lot of this work, but the endings of hyperlinks should be marked carefully. (Also, think about this: Do you want "mailto:" links to open the browser's internal e-mail response function or your application's?)
  6. More robust error handling is required. If the POP server doesn't accept the user name and password, the application displays a nasty error message from the ColdFusion server. Also, there's no indication of success or failure when e-mail is sent. Additional error checking and validation are necessary.
  7. More robust security is also required. Using ColdFusion 4.0's Encrypt and Decrypt functions might be useful for encrypting passwords. Additional code may be necessary to support "refreshing" of session variables if they've timed out or, if desired, the user can be required to sign in again after a timeout. Using this application over an SSL (Secure Sockets Layer) connection is another way to enhance security.
  8. The <CFPOP> tag in ColdFusion 3.0 has some known bugs. One rather serious bug prevents users from reading any message beyond the first one if that first message has an attachment. (This was fixed in ColdFusion 4.0.) It's recommended for this reason that ColdFusion 4.0 should be used as the basis of this application.
  9. In addition, the <CFMAIL> tag lacks a lot of the functionality one would want for sending e-mail (blind carbon copy, reply-to header, etc.). There are shareware and commercially available ColdFusion tags that can serve as plug-compatible replacements for <CFMAIL> (and for <CFPOP>), including Christopher Evans's CFX\_MAIL and Patrick Steil's CFX\_iiPOP3. These are available at Allaire's ColdFusion Developer's Exchange, formerly known as the Tag Gallery ([www.allaire.com/developer/gallery.cfm](http://www.allaire.com/developer/gallery.cfm)).

## Conclusion

I've barely scratched the surface in describing what's involved in building an application like this. The code snippets I've provided, for simplicity's sake, don't include certain finesses in laying out the pages the way they appear in the screen shots. Still, I hope this article serves as a useful introduction to the process of designing and building a Web-based e-mail system in ColdFusion.

If you'd like to take a look at the resulting application in its current state, visit [www.neurozen.com/popart](http://www.neurozen.com/popart). It's still in beta; however, it provides all the main features and a number of the additional enhancements described in this article. 

## About the Author

Rich Rosen has been on the Net since before there was a Net. He's been with Pencom Web Works ([www.pencomwebworks.com](http://www.pencomwebworks.com)) since 1997, building e-commerce, multimedia and Web/database connectivity solutions using NetDynamics, Macromedia Flash, RealAudio and ColdFusion. The popART e-mail system described in this article is one of the many showpieces at his personal Web site ([www.neurozen.com](http://www.neurozen.com)).

[rlr@neurozen.com](mailto:rlr@neurozen.com)

# Virtualscape

[www.virtualscape.com](http://www.virtualscape.com)