Building Dynamic WAP Applications with ColdFusion

Charlie Arehart, SysteManage

In this paper, you'll learn what dynamic content is, how it's useful, and how it applies to WAP applications. I'll also introduce ColdFusion, a leading, easy to use, web application development environment, and show you why it's a powerful tool for building dynamic WAP applications.

I'll cover several important tips and tricks for CF/WML development, as well as support for WML development in Allaire's two award-winning text editors: CF Studio, the integrated development environment (IDE) for ColdFusion, and the HomeSite HTML text editor.

Part 1: Preaching the Good News

Part 1 presents an introduction to the general concept and benefits of using dynamically generated WAP applications, as well as an introduction to ColdFusion. Those experienced in both dynamic WAP applications and ColdFusion will want to skip to *Part 2: Preaching to the Choir*.

What is a Dynamically Generated WAP Application?

A dynamically generated WAP application is one whose contents are created on the fly, perhaps based on data in some central database. An example might be a presentation to a client of their list of stocks in a stock tracking application. They only want to see their own list of stocks, and so the relevant information is obtained from a table of such data.

The deck and cards presented to the user to show them their choices are offered in WML (or HDML, if you prefer). There's nothing special about what is sent to the client/browser: it's the same as the WML that you would create by hand in a static WML page.

Code Generated by a Server-side Application

Rather than being a fixed set of choices coded in WML by hand, a dynamically generated WML deck will be created by a program running on the web server from which the user requested the information. This creates a kind of two-tier system, where the back end (or processing layer) is separated from the presentation layer. The program will run in a web application server environment, of the kind that has traditionally been used to create dynamic HTML applications.

Suitable web application servers include Microsoft's Active Server Pages (ASP), Allaire's ColdFusion, and JavaServer Pages (as provided by a tool such as JRUN). Perl is another traditional web application development environment. All of these environments have been employed by web application developers to create dynamic HTML applications, and each of them can be used similarly to create dynamic WAP applications.

The example offered above — generating content from a database — is perhaps the most obvious possibility for dynamically generated web applications. It opens the door both to making existing enterprise data available for display on browsers, and to creating databases specifically for use by dynamic web applications.

That leads naturally to another possibility: creating **two-way applications**.

Two-way Applications

In a nutshell, two-way applications prompt the user for input, and serve information in response to specific requests. Examples of two-way applications include search engines or data entry pages. In web pages, such capabilities are generally enabled by HTML forms. WML offers similar user input capabilities, but while WML does offer mechanisms to process and manipulate user input on the client, there's not nearly the power and versatility of processing input as there is on the server with a tool such as CF, ASP, JSP, or Perl.

When a WML page calls upon a server program written in one of these languages, the latter can perform processing that results in the generation of a web page (either HTML or WML). The server program is written in the language of the server development environment (Visual Basic, CFML, Java, Perl, etc.) but the result sent to the user in response is HTML or WML.

The server program might search for data based on user input, store that user input in a database, or cause records to be updated or deleted. Furthermore, it could send e-mail based on the WAP client's request, or communicate to another process on the server on the client's behalf. Finally, the user will receive a WML page that shows or explains the result of their request.

What Makes Dynamic WAP Applications So Great?

For a web site (HTML or WML) to be dynamically generated is a clear sign of its maturity. Dynamically generated web sites open up many possibilities for visitors and the site owner alike.

Personalization

One of the most important benefits is customization of content based on the user's preferences or their permitted access to data. Personalized sites are a worthwhile design goal because they are much more responsive to the needs of the end user, and serve to make the user feel better about the 'browsing experience'.

Personalization starts with the identification of the user in such a way that they can be uniquely associated with their profile of preferences or authorized access. In the development of HTML applications, the traditional means of identifying users is by way of a login form — and cookies are used to remember a user's authentication over multiple visits.

One difference with wireless browsers is that some mobile phones can offer a means by which to identify the user uniquely. Not all phones do, but this approach can provide assurance of the visitor's uniqueness without violating their privacy, making personalization even more powerful. If a phone doesn't offer such a facility, or the user has chosen to disable it, the more traditional means of login authentication (a login form) is available. Also, some phones (or gateways) now support cookies too, although the support is thin and not something you should presume just yet.

Site Security

An extension of the features enabled by personalization, where we determine the identity and authenticity of the user, is that we can also limit what parts of the site the user is authorized to see. This enables important data to be made available to the user without concern that it may be viewed by an unauthorized party, as might be the case with a static web site. (It's true that web server security can be used to protect static data, but there are times when developers are unable to control web server security configurations. In such cases, they can create programmatic security.)

Database Query and Update Processing

As mentioned earlier, another important benefit of dynamic sites is database query and update processing. A WML deck can provide a search interface, and a server program can search the appropriate database table (or site search index). Perhaps more powerful still is the ability to store, update, or delete database data based on user input. Mobile applications can provide substantial benefit when users can provide and update data from the field.

Generating e-Mail

Yet another benefit of dynamic sites is that the server program can generate e-mail based on actions performed on the client. Perhaps a form will simply lead to mail being sent as a direct result — giving the user the opportunity to provide site feedback, for example. Another use of e-mail is the generation of a message based on some activity taken by the user, such as the updating of a database in response to registering their details on a site.

Integrating Other Content

Still another possibility is to integrate content from other applications, or even from other sites. For example, the server program might request data from some other program operating on the server. Perhaps that other program is written in another programming language?

There are several mechanisms available for the mediation of such inter-program communications: Component Object Model (COM) and Distributed COM (DCOM) in the Windows environment, Common Object Request Broker Architecture (CORBA) in most other environments, and Enterprise JavaBeans (EJB) in Java programming environments.

Some server programming languages offer a means to gather information from other web sites. Such tools can parse the content gathered from another site for presentation to your visitors.

Scheduled Content

Finally, one of the biggest benefits of dynamic content is that you can use server-side scheduling features to cause generation of content at a particular time, on either a recurring or a once-only basis. This can cause the server to reflect data changed based on scheduled content, or in wireless browsers this can be more powerfully employed by taking advantage of WAP's 'notification' or 'push' capabilities.

With notification (a feature offered in the proprietary but widely supported Phone.com WAP architecture), content is *sent* to the browser, rather than the browser coming to a site to *request* content. Notification processes are most often associated with such features as messages sent to warn of dramatic changes in stock prices. More practical applications could include notifications based on changes in business processes (stock outages, site break-ins, etc.). The WAPForum's 1.2 standard is due to add a 'push' feature that could be used in similar fashion.

Introducing ColdFusion, a Stellar Tool for Building Them

The need for and benefits of interactive, two-way web applications became apparent a few years ago with HTML browsers, and one of the earliest web application servers to arrive was Allaire's ColdFusion. In fact, despite the ubiquity of Microsoft ASP in the Windows world, ColdFusion is the leading cross-platform web application server.

Running on both Windows and several versions of UNIX (including Linux), and with nearly every web server (including Apache), CF has continued to thrive due to its integrated security, scalability, broad integration with other technologies, and, perhaps most important, its ease of use. It doesn't hurt that it's inexpensive compared to other similar web application servers, ranging from free (for a limited version) to between $2,000 and $5,000 for its professional and enterprise versions, respectively.

ColdFusion's Ease of Use

As a demonstration of the ease of using ColdFusion, take a look at the following sample code fragment that shows a ColdFusion program (also referred to generically as a 'template') generating WML:

```
<card>
   <CFQUERY NAME="GetDepts" ...>
      Select DeptName from Depts
   </CFQUERY>
   <p>Depts Found:<br/>
      <CFOUTPUT QUERY="GetDepts">
         #DeptName# <br/>
      </CFOUTPUT>
   </p>
</card>
```

Although CF was originally designed and intended for creating HTML, there's absolutely nothing to stop it from creating WML code for display in wireless browsers and simulators.

ColdFusion's Flexibility

I mentioned previously CF's ability to run on multiple operating systems and web servers. One thing I didn't point out was that the *exact same* code written on one OS for a particular web server can be moved to another OS against another web server, with high probability that no changes will be required to the code. (Apart from some minor alterations such as case sensitivity on links when moving from, say, a Windows environment to a UNIX environment, if you've not programmed for that concern in advance.)

In fact, the flexibility is even more substantial than that. CF can run against many different *database* servers, including any ODBC-compliant database (typically desktop databases such as Access and Visual FoxPro), as well as enterprise-class databases such as SQL Server, Sybase, Oracle, Informix, and DB2.

Again, an extension of this flexibility is that a ColdFusion program written against one DBMS could easily be migrated to run against an entirely different database (going from, say, a test version of the database in Access to a production version in Oracle) without changing the ColdFusion code. As long as the database table and column names are the same, the CF code (and underlying SQL) often need not change. This is a powerful benefit.

The final element of this flexibility refers to the integration mentioned earlier, whereby a ColdFusion template can request information from another application on the server, including communication via COM/DCOM, CORBA, EJB, etc. ColdFusion supports gathering data from other web sites through its <CFHTTP> tag.

ColdFusion's Scalability

When people begin to consider dynamically generated content, a first concern is often whether a dynamic site can perform well under large loads, when thousands or even millions of visitors arrive at the site each day.

Allaire has refined each successive release of ColdFusion (now in version 4.5) to address scalability concerns. The enterprise version incorporates integrated

clustering, load balancing, and failover support, so that visitors to the site are spread over multiple (sometimes dozens of) servers in a seamless way, which makes it transparent if their session is swapped among servers in order to balance server load.

Even in the free and professional versions, there is built-in support for scalability by way of such things as web server integration, template code caching, and state management capabilities.

ColdFusion is used by several leading web sites, including ToysRUs.com, the Wall Street Journal's SmartMoney.com site, AutoByTel, and many more. (Even the SQL Server Magazine and Windows NT/2000 Magazine sites are ColdFusion sites, rather than ASP sites!)

Part 2: Preaching to the Choir

In this second part of the paper, I'll focus on the specifics of generating WAP content with ColdFusion, including the features available in CF and CF Studio/HomeSite for creating WAP code. I'll conclude with several useful tips and traps gathered from the experience of having developed CF/WAP applications.

Serving WAP Content via ColdFusion

The most important thing you need to know when trying to create WAP applications in ColdFusion is how to tell the browser calling a ColdFusion program that the code it will receive is the proper MIME type expected by that browser.

The All Important MIME Type, Set with CFCONTENT

In wireless applications — specifically, in WML applications — the MIME type should be `text/vnd.wap.wml`. Normally, one sets up the web server to get `.wml` pages sent to the browser with that MIME type.

When generating dynamic WML code with ColdFusion, however, the file type used is not `.wml` but `.cfm`. This tells the web server to pass the ColdFusion code to the CF server for processing. But whereas ColdFusion is installed by default to create output with a MIME type of `text/html`, we need specifically to send WML code with the proper WML MIME type.

To do that, we use the ColdFusion `<CFCONTENT>` tag, typically placed at the very top of the ColdFusion template that will be generating WML content. Specifically, the tag will be:

```
<CFCONTENT TYPE="text/vnd.wap.wml">
```

> *If you have some ColdFusion experience, you may know that this tag has been around for many releases — it's not a new addition for generating WAP content. Its sole purpose, which happens to be very useful here, is to change the MIME type of the page being generated.*

You can use a `<CFCONTENT>` tag in the same way for templates that generate HDML or WMLScript, substituting the appropriate MIME types.

A Skeletal CF/WAP Template

To put this tag in its proper context, here's a skeletal code template for creating a WML deck within CF. Notice that the only ColdFusion aspect of it is the very first line, with the <CFCONTENT> tag slipped in before the XML declaration:

```
<CFCONTENT TYPE="text/vnd.wap.wml"><?xml version=1.0"?>
<!DOCTYPE wml PUBLIC "-//PHONE.COM//DTD WML 1.1//EN"
                     "http://www.phone.com/dtd/wml11.dtd">

<wml>
   <card>
   </card>
</wml>
```

Because this is such a fundamental piece of code, necessary for any CF/WML development, Allaire has altered the latest release of ColdFusion Studio (the CF editing environment) — and its cousin HomeSite — to be able to create this sort of thing whenever you request to create a new template. I'll explain this in more detail later on.

If you're an experienced WML developer, you'll have noticed that this page is set up to use the Phone.com WML document type definition (DTD). You can just as easily choose to type in the WAP Forum's DTD declaration on the second line, if you prefer. (Later on, I'll show how you can change Studio/HomeSite's new page Wizard to do that as well.) One other minor point to note is that I've placed the <CFCONTENT> on the same line as the XML declaration, to avoid unnecessarily creating carriage returns before that tag, which some wireless browsers complain about.

From here, you're free to insert any ColdFusion code you want, either to create WML elements for sending to the user, or to perform server processing on behalf of the user. A variation on the classic "Hello World" example can be presented using simple ColdFusion variable and output processing, as in the following:

```
<CFCONTENT TYPE="text/vnd.wap.wml"><?xml version=1.0"?>
<!DOCTYPE wml PUBLIC "-//PHONE.COM//DTD WML 1.1//EN"
                     "http://www.phone.com/dtd/wml11.dtd">

<wml>
   <card>
      <CFSET Name="Charlie">
      <CFOUTPUT><p>Hello, #name#!</p></CFOUTPUT>
   </card>
</wml>
```

This is a very simplistic example, of course. There's nothing terribly dynamic about it (the very first fragment you saw was more realistic). But it does demonstrate the basics of creating and using variables in CF. If you're new to WML, you should take note that we surround any text sent for display to the wireless browser in <p> tags, and that WML tags are entered in lowercase.

Those new to ColdFusion should note that we are intermixing CF tags and WML elements, but the CF tags are generating WML elements that will be sent to the browser. The CF tags will be stripped out after performing their assigned tasks on the server. All that the user will receive is WML. (Which is also why you need not worry about the CF tags being XML-compliant. You don't need to close the <CFCONTENT> or <CFSET> tags, because they're never sent to the browser.) Also, note that the page is a `.cfm` file (`hello.cfm`), rather than a `.wml` file.

A more practical example would be a page that processes a WML form, where the user is asked to enter data that we can process on the ColdFusion page on the server. First, I'll present a fragment of the form, which is just a pure WML file. The

only hint that something more is going on is the <go> element that passes control to a ColdFusion page when the accept action is triggered (by the user pressing their "OK" button [or equivalent] on the phone):

```
<card>
   <do type="accept">
      <go href="wml_action.cfm" method="post">
         <postfield name="symbol" value="$(symbol)"/>
      </go>
   </do>
   <p>
      Enter stock symbol:
      <input name="symbol" format="4A"/>
   </p>
</card>
```

This is clearly quite different from an HTML form. For a start, there is no <FORM> tag in WML, and the code used to display the form and that for sending the form to the server are distinct. You can also see the use of internal variables within WML (something HTML can't do) when we use the form data gathered by the <input> element and then passed by the <postfield> element. Finally, notice that the <input> element offers an opportunity to provide an expected format for the data entered, which is another benefit over HTML.

The ColdFusion page for processing the form (referred to above as wml_action.cfm) could be:

```
<CFCONTENT TYPE="text/vnd.wap.wml"><?xml version=1.0"?>
<!DOCTYPE wml PUBLIC "-//PHONE.COM//DTD WML 1.1//EN"
                     "http://www.phone.com/dtd/wml11.dtd">

<wml>
   <card>
      <p>
         The stock symbol selected was:
         <CFOUTPUT>#form.symbol#</CFOUTPUT>
      </p>
   </card>
</wml>
```

This is still a simplistic example, since we're not really doing much with the form data, but you can see how the form data (the symbol variable in WML) is passed to and made available within the ColdFusion program (as the CF variable form.symbol).

We could also have passed the form variable to a query, to find all the records in some database table that match the stock symbol.

Where to go From Here?

From here, it's a simple matter of placing valid CF tags in the page to create proper WML, or to perform server processing (updating databases, generating e-mail, communicating with other applications or web sites, etc.) on behalf of the user.

If you're new to WML, remember that you must put valid WML code on the page. You must learn WML, and the differences between it and HTML, before you can begin coding effectively. And if you're new to CF, you should really learn as much as you can about the subject. I've only brushed the surface in this paper, and there are dozens more tags and hundreds more functions at your disposal. Some of the more important CF components you should study for creating wireless applications include:

- CFMAIL (for generating e-mail)
- CFSCHEDULE (for creating notifications and other scheduled content)
- CFSEARCH (for performing text searches through document indexes)

- Session, client, application, and server variables (to perform state management)
- Query caching (to improve query processing for frequently referenced data)

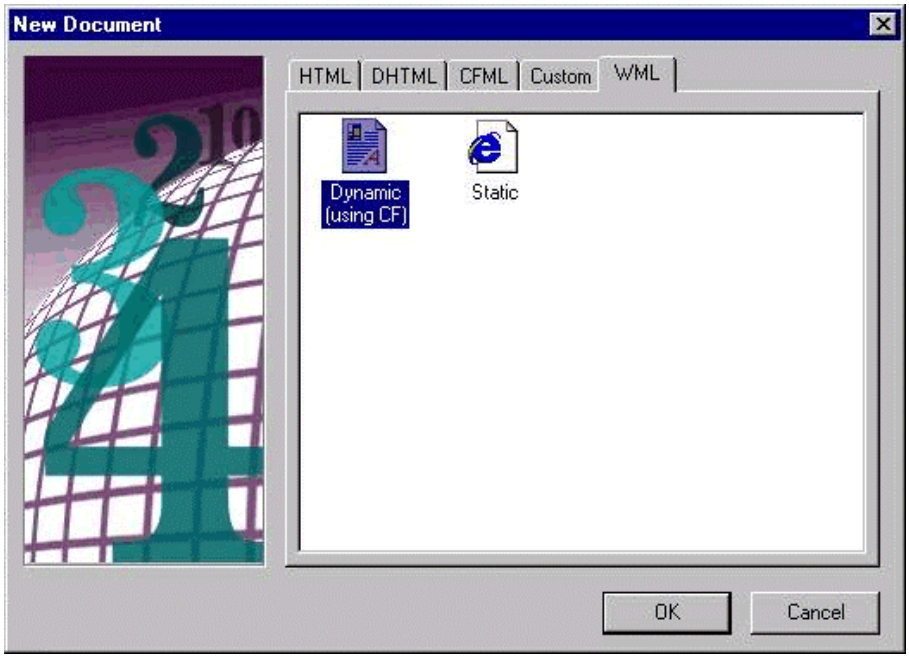
Combining WML and CF expertise will open up worlds of possibility for you. As well as the things I've mentioned so far, interesting combinations of CF and WML include creating table-formatted displays of data, dynamically generating pick lists, and much more.

Studio/HomeSite Features for Creating WAP Apps

As mentioned before, CF Studio is the IDE for ColdFusion coding. On the surface, Studio is a text editor, albeit one with many valuable features for web page developers. It's based on Allaire's award-winning HomeSite HTML editor. Studio simply adds many features specifically for ColdFusion developers. Both Studio and HomeSite, like ColdFusion, can just as easily be used to create WML pages if you know what to write. However, release 4.5 offers features that are specifically geared toward WML developers to help make coding easier, especially if you're new to WML. (From now on, I'll refer to simply Studio but all these features are in both Studio and HomeSite.)

New Page Wizard

To make Studio create a new, skeletal page for coding WML applications with CF, use the File | New menu command and choose the WML tab from the dialogue offered. From there, you have the choice of creating a new, blank template with skeletal WML code and either with or without the <CFCONTENT> tag filled in for you (referred to as dynamic or static, respectively):

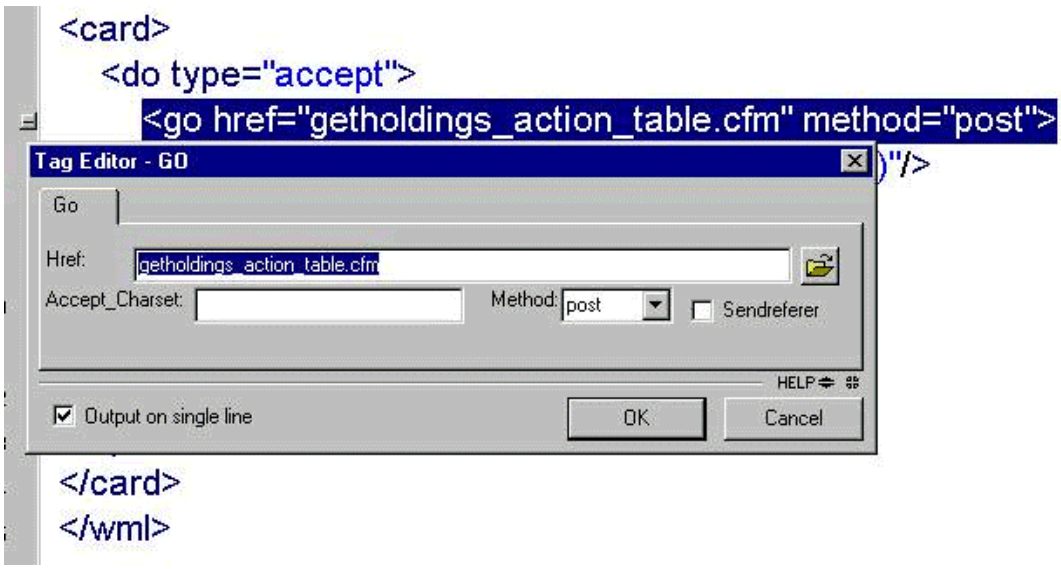


Tag Insight

Even more valuable, when entering tags in Studio (whether HTML or WML), is a feature called Tag Insight. If, after entering the tag name and a space, you wait a second, you'll be presented with a list of all the valid attributes for the given tag:

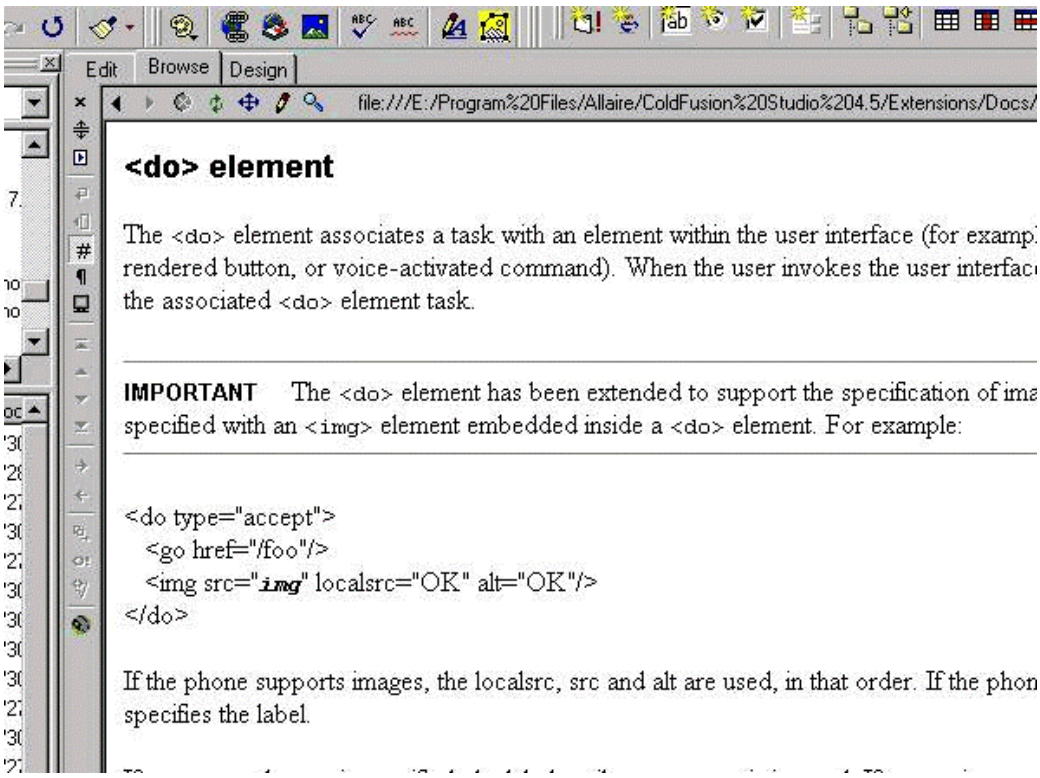
Tag Editors

Another useful feature for editing WML code is Studio's Tag Editor. With this, you can place the cursor on a tag and either right-click and choose Edit Tag, or press *Ctrl-F4*, which pops up a dialog that displays the available attributes and their appropriate values (where they are predefined). Also, this feature often presents the attributes in the order of their most likely usefulness, whereas Tag Insight simply presents them in alphabetical order:



Tag Help

Yet another useful feature is Studio's built-in help. Sometimes, you may not want to see the available attributes in a dialog (like the Tag Editor) or in a list (as in the Tag Insight feature). You may just want to read about the available attributes and their predefined values (if any). To do that, simply press the *F1* key while the cursor is on an element (CF, HTML, or WML). Studio will open a help window for that element:



To return to editing your code, simply press *F12* or click the Edit button offered above the window showing the help.

Studio's tag editing features are powerful; they make it easy to enter code while typing (Tag Insight) and provide assistance when you're not sure how to proceed (Tag Insight, Tag Editor, and Tag Help).

Tricks and Traps

So: we've been glorifying the wonders of dynamically generated WAP content, especially within ColdFusion. Is it perfect? Of course not. There are several traps, as well as a few more tricks, that you should know about before setting off to code CF/WML.

When CFCONTENT is Unavailable

First and foremost, you may find that the all-important `<CFCONTENT>` tag is rejected as an invalid CF tag. There are two situations in which this could occur. First, if you're using the free version of ColdFusion (CF Express), the tag is simply not supported. Perhaps that will change in the future, but for now it does mean that you can't create WML content within CF Express.

The other situation where `<CFCONTENT>` is unavailable is when the CF administrator has disabled the tag for security reasons. The security risks have nothing to do with the way we're using `<CFCONTENT>`, but rather with ways it can be used for other purposes (via its `FILE` attribute). Sadly, this too is a showstopper until Allaire

provides a means to restrict the use of the FILE attribute, while permitting our more generic use of the tag. (In my CF/WML chapter in *Professional WAP Programming*, Wrox Press, I offer a possible workaround.)

Studio Traps (and Tricks)

Another problem for some is that Studio support for creating WML pages and elements is all based on the Phone.com specification for WML, rather than the more generic WAP Forum standard. As such, it is possible to create WML code that will not work in all browsers (specifically, those that do not support the Phone.com specification).

I also recommended earlier that you should place the <CFCONTENT> tag on the same first line as the XML declaration. Unfortunately, Studio's new page Wizard for creating a CF/WML skeleton page does not follow that recommendation.

Both of the above problems can be solved by modifying the file that Studio uses to create that skeletal page — it's simply a file in the directory Program Files\Allaire\ColdFusion Studio 4.5\Wizards\WML, on the drive where Studio is installed. The file is called (literally) Dynamic (using CF).cfm. Modify it to suit your tastes, and the next time you use the File | New | WML | Dynamic feature, the new untitled page shown will reflect your changes.

A subtler problem in Studio is that there is an option that causes all tags created by features like Tag Insight and Tag Editor to be in uppercase. Since WML requires lowercase tags, this feature should be turned off. (See Options | Settings | HTML | Lowercase all inserted tags.)

CF's HTML Heritage

A more pernicious problem is one based on CF's heritage as a tool for creating HTML pages. While we've simply coded WML and there's been no problem, there are features in CF that create output for us, and those features tend to create output in HTML. Specifically, CF error messages and debugging information include HTML tags. Sadly, a WML browser will generate an error if HTML code is sent to it. But again, there are a couple of workarounds for this.

CF provides for a special file called application.cfm to be executed before any other CF page. My recommendation is to place such a file in the same directory as your WML code (preferably in a directory that has only CF templates that create WML code), so it can execute several CF tags that solve some of these problems. The details of this approach are explained in *Professional WAP Programming*, but here's the listing for those who just want working code without explanation:

```
<CFSETTING ENABLECFOUTPUTONLY="Yes">

<!- the line above prevents this file from sending any code to
    the browser until the end. I've placed the comment after it to prevent
    even a needless carriage return being sent to the browser ->

<!- Setup to test the version of CF server currently running. ->

<CFSET version=listgetat(Server.ColdFusion.ProductVersion,1,", ")
   & "." & listgetat(Server.ColdFusion.ProductVersion,2,", ")>

<!- Turn on use of the wml_error.cfm with new CFERROR TYPE="exception"

    Before 4.5, the CFERROR TYPE="exception" option was not allowed.
    The tags and attribute existed, but the options did not. We can't
    even list the option if before 4.5, as the compiler will generate
    an error. But we can play a trick to get it to work if ge 4.5,
    while still allowing this code to compile successfully in a 4.0 server ->
```

```
<CFIF version ge 4.5>
   <CFSET type="exception">
   <CFERROR TYPE="#type#" template="wml_error.cfm">
</CFIF>

<!- Use CFSETTING to turn off CF server-side debugging, if turned
    on, since it, too, is HTML that will choke a WML browser.

    Before 4.0, the CFSETTING SHOWDEBUGOUTPUT attribute was not
    allowed, but there's no way to trick the server with this one,
    since you can't use variables for either tags or their
    attributes. If you're running a release older than 4.0, simply
    remove the following line ->

<CFSETTING showdebugoutput="no">

<!- Restore the setting that prevented this file sending any text
    to the browser. Must be done or subsequent pages will send no
    output to the browser unless it's enclosed in CFOUTPUT - generally
    not desirable ->

<CFSETTING ENABLECFOUTPUTONLY="no">
```

You can't run `application.cfm` directly. Instead, you place it in the directory where it is to affect other programs, and whenever one of those other programs is executed, the `application.cfm` is run first.

The `application.cfm` file above refers to another `.cfm` file called `wml_error.cfm`, which offers an alternative means of formatting the output of CF errors when they occur. Here is that code:

```
<CFCONTENT TYPE="text/vnd.wap.wml"><?xml version="1.0"?>
<!DOCTYPE wml PUBLIC "-//PHONE.COM//DTD WML 1.1//EN"
                     "http://www.phone.com/dtd/wml11.dtd">
<wml>
   <card>
      <CFOUTPUT>
         <p><b>An error has occurred in this application. Please
         share this diagnostic information with the system
         administrator<CFIF cferror.mailto is not ""> at
         #cferror.mailto#</CFIF>.</b></p>

         <p><b>Error Text:</b>
            #htmleditformat(cferror.diagnostics)#</p>

         <p><b>DateTime:</b> #cferror.datetime#</p>

         <p><b>Error Template:</b>
            <CFIF cferror.Template is "">
            (unavailable)<CFELSE>#cferror.Template#</CFIF></p>

         <p><b>Referrer:</b>
            <CFIF cferror.httpreferer is "">
            (unavailable)<CFELSE>#cferror.httpreferer#</CFIF></p>

         <p><b>Query String:</b>
            <CFIF cferror.querystring is "">
            (unavailable)<CFELSE>#cferror.querystring#</CFIF></p>

         <p><b>Browser:</b>
            <CFIF cferror.browser is "">
            (unavailable)<CFELSE>#cferror.browser#</CFIF></p>

      </cfoutput>
   </card>
</wml>
```

Another element of CF's HTML heritage is that several CF tags are designed to generate HTML (and/or JavaScript code) for you automatically, such as `CFFORM` (and its child tags `CFINPUT`, `CFSELECT`) and `CFTABLE`. As useful as these are for HTML-oriented CF developers, they must not be used when creating WML decks from within CF, since they do not generate valid WML.

Serving Multiple Clients

Another problem confronting developers considering dynamic WAP applications is how to serve multiple clients. How do you know if your visitor is an HTML or WML browser? How do you code your program to serve appropriate content?

You have a few choices. Any will work, and the choice will depend on your application, skill set, available resources, and priorities.

The first thing to be aware of is that you can indeed detect the type of browser that is visiting your site and running a given program. Browser detection in ColdFusion is discussed in the next section. Assuming you know the type of browser, then, what are your choices?

One approach is to have a given server program tailor its output to a range of different markup languages, depending on the type of browser being used to access it. In other words, a single program might be designed to provide a list of stocks. It could be programmed to provide the list in HTML, to HTML browsers, and in WML to WML browsers.

As elegant as it may seem, this approach may not always be appropriate. Developers who have already written an application in HTML may hope to save development time by creating multi-purpose programs to serve both clients in a single program, but given that the output requirements for WAP browsers are generally quite different from those for HTML browsers, the effort to ensure appropriate display for both types of browsers may negate the potential time savings of having a single program.

A better approach is to have a front-door page that detects the type of browser visiting your site, and then directs the user to a set of pages that serve the appropriate result. If the results being sent to the browser really are quite different depending on the type of browser, this may be the more appropriate approach.

Another still more elegant result would be to split programs into processing and presentation logic, and have the processing component pass results to the presentation component for display to the user as appropriate. As ColdFusion is more a procedural language than an object-oriented one, this sort of segregation may, again, be more trouble than it's worth. It's worth investigation, though.

The bottom line is that there's nothing in ColdFusion to prevent you choosing any of these three approaches. There are even specific features that can be used to make each approach easier depending on your preferences, some of which are discussed in this paper (browser detection and redirection) while others are left to the reader to investigate (custom tags, calling objects).

Browser Detection

Many people are interested in doing browser detection, to recognize if the visitor is a WML or an HTML browser. While I recommend that you think twice about trying to serve WML and HTML from within the *same* template (for the reasons explained above in *Serving Multiple Clients*), it's perfectly reasonable to want to detect the browser type and send the user to a page suitable for processing by that kind of browser.

One of the most reliable means of doing that is to detect what MIME type the browser is expecting. HTML browsers will expect an HTML MIME type, and WML browsers will expect a WML MIME type. (There's a chance that someday, HTML browsers will permit and emulate processing of WML decks, but until then this is a suitable approach.)

Fortunately, the browser reports the type of pages it expects in a CGI variable called `http_accept`, and CF can access that variable. The CF code to detect and redirect a page based on the browser would be just:

```
<CFIF cgi.http_accept contains "text/vnd.wap.wml">
   <CFLOCATION url="/wml/index.cfm">
</CFIF>
```

Notice that the code is also using a CF tag called `<CFLOCATION>`, which can be used to tell any CF program to pass control to another page or program. Placing this code inside a page at the front of your site, such as `index.cfm`, would allow that page to detect and redirect WML browsers to a suitable WML-encoded page.

For More Information

Of course, there are many more CF tricks, traps, and additional examples that I could offer, but space prevents further elaboration here. Each of these, and lots more, is offered in the chapter on "Developing WML Applications with ColdFusion" in the Wrox Press title, *Professional WAP Programming*.

Dynamically generated WML is a powerful tool, and ColdFusion is especially well suited to the task. The combination of these two can take your web sites to the next stage, adding tremendous value along the way.