Tau Beta Pi Conquers ColdFusion®
The Creation of a New Website

by Raymond H. Thompson, Karen C. Robnett, Kasi Miller, Stephen L. Cate, and James E. Goetze

INTRODUCTION
The Tau Beta Pi National Headquarters needed major revisions to its existing website. The old version was a nicely done site, but provided information only. The site had served its purpose and was ready for an evolutionary change. Becoming an interactive site able to adapt the output to the conditions and information stored on computer systems and in databases was the ultimate goal. Along with this improvement, changes were to be made in the infrastructure of the local Tau Beta Pi network environment.

Q Systems was contracted to do a study of the existing systems and make recommendations for a complete overhaul of the existing environment. This included new systems to act as servers and a reconfiguration of the network.

THE NETWORK CHANGE
The prior environment consisted of two main servers. One server, a Macintosh, provided the main web interface. A second Intel-based system provided file and data storage. There were several workstations and printers all connected via Ethernet connections, which extended to the outside world through the main University of Tennessee (UT) network.

Email service was being provided by the university's computing services, and Headquarters desired to move this function to Tau Beta Pi equipment.

Several areas of enhancement were identified with this configuration. Since unique IP addresses were being assigned to each workstation, these workstations (and servers) could be vulnerable to attacks from outside sources. Reducing Tau Beta Pi to it's own sub network with firewall protection would enhance the security.

To accomplish this, Headquarters decided that three servers would be installed. One server would act as the file server, one would be the web server, and one would become the firewall, proxy server, and mail server. Since Tau Beta Pi was being charged for each unique IP address used, a way was devised to reduce the IP address assignment. This was made possible by using proxy software.

Network equipment was acquired to provide one connection to the UT network. This connection then was run through the firewall and proxy server, back to a small rack with connections used for Tau Beta Pi only. By doing the connections this way, all the existing wiring could be used and still provide access to the outside world and the UT network. Tau Beta Pi effectively became its own small network and still is able to access the world.

The computers chosen for the new servers were from Compaq Computer Corporation. Systems were to be rack mounted to conserve space. The disk drives were configured in a RAID-1 configuration with hot spares. Redundant power supplies were specified, along with UPS protection. All of this was done to provide a high level of availability, which would be essential because of the new web requirements.

There were some difficulties in getting all of this to work. Some difficulties were known; others were unknown, but expected. Some were surprises. Thanks to the outstanding work of our network specialists, Leslie Palmer and Mike Edmonds, the connections were made, and all was running smoothly while maintaining the existing functionality.

THE WEBSITE
When a website becomes interactive, it requires some sort of back-end operation that can process data and return responses. There must also be a database to support the storing and retrieval of data.

Tau Beta Pi required the membership database to remain in FoxPro because of its previous experience with the software. The Society was already in the process of working with another company to convert the database from a text file, stored on a UT mainframe, to FoxPro. Tau Beta Pi also had a requirement to create additional data stores for eligibility information provided by chapters, convention information, chapter officer information, and fellowship/scholarship applications.

The choice made for the web language was ColdFusion® (CF) produced by Allaire. CF has the capability of supporting multiple types of databases using native drivers or standard "open data base connectivity" (ODBC) connections. An ODBC driver was available for FoxPro as well as MSAccess, which would be used for the additional data stores.

ColdFusion® is also a rapid-development language that is well integrated into the demands of the web. CF code is never compiled, but is instead interpreted as the code is needed. This saves a tremendous amount of development time, and a step in the typical development process is eliminated. This interpreting of code, when required, also provides some fairly easy methods to debug the code. CF also integrates with HTML, which is used to produce the web pages on the client computer. There is also no need to provide any type of client software, because the result of the processing by CF is standard HTML.

There is a minor performance penalty when processing code that must be interpreted each time the code is used. ColdFusion® reduces this by building p-code when a section of code is first used. Subsequent requests for that same code result in CF using the already generated p-code. This saves the code-parsing step for each subsequent request for the script. The cache on the server is set to be large enough to hold the p-code contents of all the scripts in memory. Therefore, the performance penalty is small.

COLDFUSION® BASICS

ColdFusion® is an easy language for developers to use. People who have not seen it before, but are proficient in other languages, have no difficulty in transitioning to CF.

One of the big advantages is that CF does not require the prior declaration of variables. A variable is created when it is first assigned a value. CF variables are always "variant" type variables. The type of data stored in a variable is determined at the time the variable is used, and CF will do any necessary conversion.

Most HTML tags have a begin tag and an end tag with the begin tag something like <TAG> and the end tag like </TAG>. CF is no different. All CF tags begin with CF and are coded like HTML tags, with most having a begin tag <CFTAG> and an end tag </CFTAG>.

```
<CFSET V1=2>
 <CFSET V2="5">
 <CFSET V3=V1*V2>
```

Data that is to be output in CF must be identified to the CF server, and all regular HTML needs to be ignored. CF accomplishes this by requiring all CF variables to be surrounded with pound signs (#) and contained within CFOUTPUT tags. CF can control any output, including HTML. This allows truly dynamic page generation.

```
<CFOUTPUT>#V3#<CFOUTPUT>
```

CF has full conditional processing that allows the code to make intelligent decisions based on the data provided. Full nesting of conditional logic is allowed along with compound conditions. CF also supports short-circuit evaluation so that if a statement is encountered of the form IF A=B AND C=D and the A=B relationship is false, the C=D relationship is never evaluated.

CF supports multiple variable locations and scooping of these variables. Application variables are unique to the application and are shared by all users of the application. Server variables are shared by all applications on a server. Session variables are unique to each user of the application and are stored in the server memory. Client variables are unique for each user and are stored in the server registry. URL variables are stored in the address line of the browser and are unique per instance of the browser. Cookie variables are stored on the user's system in a special text file.

THE DEVELOPMENT FRAMEWORK

Since CF will work with many different databases, and it would take some time for Q Systems to acquire FoxPro, an alternative plan for development was devised. The structure of the database, as it would exist on the Tau Beta Pi system, would be replicated using Oracle. Care was used in the SQL (System Query Language) code to access the database to use only ODBC constructs that would be understood by the Oracle and FoxPro database environments. After all, ODBC is supposed to allow easy migration to other databases. With that out of the way (or so it was thought), the development started. As it turned out, this decision proved to be critical later in the development.

The overall layout was accomplished using two frames. The frame on the left would contain all the menu buttons and would be available at all times. The right frame would contain the critical data from each screen. This format would allow new functions to be selected without the necessity of working back through menus or providing menu functions on each screen. This simplifies the flow and the logic of the pages at a small sacrifice in screen real estate.

The code structure would be arranged in such a way that the individual pages would not require any intelligence as to the next step in the logic. All that an individual screen would require is to call a "dispatcher" that would look at the data submitted from a form and determine the next process required. These dispatchers contain all the logic that determines the flow of the data through the website. This was necessary as many of the pages contain multiple buttons that accomplish different functions. For the developers it becomes easy to determine the logic flow by looking at a single code file. The dispatcher logic would determine what button was clicked and what processing steps would be required next.

The dispatcher operates by calling various other scripts to continue the processing. Validation scripts are called to validate form data. If some database updating is required, then the dispatcher will call a database update routine. If all of this is successful, then the next page in the process is called.

Data validation is accomplished in scripts that verify the returned data in the context of what is being required.

```
<CFIF Form.FormID EQ "Form1">
  <CFIF Form.btnAction EQ "GO">
    <CFINCLUDE Template="Validate">
    <CFINCLUDE Template="UpdateDB">
    <CFINCLUDE Template="NextPage">
    <CFABORT>
  <CFELSEIF CheckNextButton>
       .
       .
<CFELSEIF CheckNextForm>
       .
       .
</CFIF>
```

Fields are validated for consistency, and lookups are performed against data in the database when required. If any of the conditions fail, then the code calls a standard error page and aborts. If all required tests for validity are satisfied, the code merely exits. The individual pages do not have to require error-display logic, because a standard template handles displaying of errors. If a change is needed in the error-display logic, only one file needs to be changed.

All the forms contain a variable that uniquely identifies each form. This allows the dispatcher to determine the form selected. The dispatcher then determines the button that was selected on the form and then begins the processing requested. Lack of any of this data will cause a default error page to display. There is a dispatcher for each logical process that is on the left menu frame. This type of logic provided for truly event-driven processing.

To identify uniquely each user of the system, a combination of cookie and session variables would be required. Cookies are nothing more than text data that websites use to store information. Cookies are benign and will not corrupt a computer. The only danger in using cookies is that malicious sites could track where you have been. In the new application for Tau Beta Pi there is nothing stored in a cookie that would compromise the users of the web application. Without knowledge of the application the cookies are meaningless, and no passwords are stored in cookies. Also, all the cookies used by the application are temporary and are removed when the browser is closed. No permanent information is stored on the client computer.

We also decided that the development would be open to the outside world so that interested parties could monitor and critique the process. The URL was given to select individuals, and these individuals were encouraged to examine the site with the understanding that it would sometimes crash. After all, it was still being developed, and coders do make mistakes. This outside access provided extremely valuable feedback in the development of the actual product and provided more end-user involvement.

THE CODING BEGINS

With the database designed, the connections made, the web virtual directories created, and the design framework satisfied, coding began.

Security was a paramount concern for users accessing the site. There would be multiple classes of users, and a method to securely identify those users was required. A standard logon page was created that would validate the user and continue if the user were valid. Unfortunately, this first attempt was not considered satisfactory by a board member, and significant changes were requested. As finally developed, all passwords are encrypted by the system and stored in the database encrypted. New users are assigned a somewhat random password that the user must change upon first login. The passwords follow strict requirements for content to force the user to create a more difficult to guess password. Passwords are aged and will expire after a finite time. When a new password is chosen, checks are made against prior passwords so prior used passwords are not allowed. The new password is validated against the old password and must exhibit a substantial change from the old one. The ability to email the user a password if the password is forgotten is also provided. The email is sent to the address on file so it is impossible for someone to have the password sent to a bogus address. Some of these changes were difficult and tedious to implement, but the ultimate result was a much more secure site.

The Security was a paramount concern for users accessing the site. There would be multiple classes of users, and a method to securely identify those users was required.

. . .

The ultimate result was a much more secure site.

Additional security is provided by the system so that after a certain amount of inactivity, the user will be forced to log back into the system. This is done with a combination of information stored on the client's computer and stored on the server. If certain conditions are not met when a screen is submitted, the login screen is forced.

The capability was provided to completely change the look and feel of the site by letting Tau Beta Pi change the logo, screen colors, and text colors. Screen colors and text colors are set by using CF variables to define the values in the HTML in the <BODY> tag. To allow this, all the scripts needed to conform to a standard form header. This information is maintained in a single file, so changes need only be made in one file to radically change the look of the entire site.

It was decided while in the development process that no Java applets would be used, because there is some concern from users about these applets. Instead JavaScript is used only where necessary on pages. These JavaScript functions allow client-side data manipulation, such as calculations and data entry. These are purely for the convenience of the user when entering data.

All data is validated on the server in order to provide a consistent error message. Consider a case with a combination of client-side and server validation when using a date. On the client side a date would be validated if a valid date were entered, but an invalid date would cause a message dialog to be presented to the user. Then the date would be corrected and sent to the server, but the server would recognize that the date is invalid in the context it is used. The user has now been presented with two different error messages. By doing only server-side validation, this dual error message syndrome is eliminated. This also provides more security, because it becomes impossible to save the contents of the web page to a file, remove the validation JavaScript, open the file, and then send the page to the server. Doing all server-side validation provides a higher level of data security.

A MINOR DISASTER

The development coding was proceeding smoothly using an Oracle database through ODBC connections. Since ODBC is supposed to be standard, no problems were foreseen when the change would be made to FoxPro. All that would have to be changed is the database connection in one file. As stated earlier, using existing resources allowed the development to proceed until FoxPro was acquired.

When FoxPro was installed, a significant surprise surfaced. After installing the tables and changing the DB connection, no one was able to log on. The code had not changed, so something was definitely wrong.

Jim Goetze eventually determined the source of the problem, and it was not pretty. It seems that FoxPro space fills all fields with blanks to the maximum size of the field. Oracle stores only as much data as told, while FoxPro stores the full size of the field. Fields declared as 20 characters in Oracle could contain from one to 20 characters of data. In FoxPro, it was different. If told to store five characters in a 20-character field, FoxPro stored the five characters plus 15 spaces to fill out the field.

The security system was failing because the encrypted password now contained trailing spaces when it should not. The decryption routine was failing. The trailing blanks could not be removed, because they might be valid. This required that the entire password algorithm be changed to eliminate any spaces in the encrypted password, and that solved the first problem.

The next problem was that data was being returned from the database containing these extra "padding" characters in all text fields. All of the code had to be changed to trim these extra spaces from the data returned from the database. Fortunately, CF provides a function that will efficiently remove these extra spaces.

Then another problem was discovered by Mr. Goetze. When records are deleted in Oracle (or any reasonable database), the records are immediately deleted. However, FoxPro only marks the records for deletion. That is not all bad, but FoxPro still returns those records when a query is run against the database. Therefore, deleted records were still being returned on queries when these records should not have been returned. Only when the database is reorganized are these "delete marked" records deleted. The solution was to preface all queries with a command to ignore delete records.

The development was off and running smoothly until another problem with FoxPro surfaced. FoxPro will not accept text fields that are longer than 256 characters when using SQL through ODBC. This was going to be a problem for which a solution would be difficult. Rather than expend the time and energy devising a non-trivial solution, it was decided to convert the tables that required these large text fields to an MSAccess database. Tau Beta Pi had a copy of this software and was agreeable to the change.

After all of this, the site was working properly again. We were surprised by the findings and learned a valuable lesson. The changes were not difficult and were easily implemented, although the difficulties just caught us all by surprise. It seems that MSAccess is a much smarter database than FoxPro when the database is being used as a data warehouse.

ODBC, which is supposed to be a common interface to databases, is not quite as open as one would think. Arbitrary limits imposed by the manufacturer of the products can limit the interchanging of database code. Many of these limits are not advertised and are discovered only by experiencing the problem.

The original configuration for the location of the FoxPro database needed to be modified when the site was installed. CF refuses to access any data source that is on a mapped network drive. The solution was to locate all the databases on the web server and map other servers to the location of the databases. CF also locks the database while it is using the database, but does not require exclusive access. This did require some small modifications to existing FoxPro scripts so that these scripts could access the database.

DOCUMENT GENERATION

There were several documents that needed to be generated from the system. Printing from the browser is never easy and is not consistent from one browser type to the next. Printing from AOL will yield different results than printing through Netscape or Internet Explorer. There is also no application control over fonts that cannot be overridden by the settings the user has made in the browser. It was clear that a different solution was needed for documents.

Using documents saved in a "rich text format" (RTF), which is highly compatible with many different applications, solved this document problem. These documents were created with dummy entries where data from the database was required. It was then a simple task to have ColdFusion® read the RTF file into memory, search for and replace the dummy data with real data, and then deliver the finished product to the user. This will cause the user's browser to launch the application associated with RTF data, typically Microsoft Word, and produce an attractive document. All fonts, spacing, lines, headers, footers, and page breaks are preserved using this method.

FINAL THOUGHTS

The process of putting together an interactive website was an adventure and learning experience. Except for the difficulties with FoxPro, the development went smoothly. Had we known at the beginning what we learned later, we would have approached the data handling differently.

Having the Tau Beta Pi website available to interested persons during development was also of great benefit. Many comments and suggestions were received during this process, which allowed difficulties in logic flow to be identified before they became difficult to change. Using the dispatcher logic allowed many of the logic-flow changes to be implemented easily.

Raymond H. Thompson has 30 years of experience in computer systems, systems design, and programming, ranging from the "big iron" mainframes to desktop systems. He spent 10 years in the U.S. Air Force developing fourth-generation language systems for personnel support systems and has written several custom compilers. He also has worked extensively in the banking industry. Ray has used ColdFusion® for two years, is a certified CF developer, and has authored several articles about the language and computers. A senior systems analyst, he developed the security and dispatching logic used within the Tau Beta Pi application.

Karen C. Robnett obtained a B.S. in business from the University of Tennessee and has 19 years of experience in automated data processing in both the private and governmental sectors. She has expertise in both the types of data applications developed (i.e. point-of-sale, accounting, marketing, waste tracking, etc.) and in the hardware/software platforms those applications use (i.e. IBM DOS/VSE, IBM MVS, IBM RISC 6000, DEC VAX Systems, Windows, Oracle, FoxPro, etc.). She has used front-end data-modeling tools to assist in design of efficient end-user relational databases and is familiar with the James Martin design methodologies.

Kasi Miller is a 1997 graduate of Maryville College, where she studied computer science/math. She has completed a variety of classes from Oracle Corporation and obtained several certifications. With three years of programming experience, she learned ColdFusion for the development of the new Tau Beta Pi system.

Stephen L. Cate received a B.S. in mathematics, an M.S.C.S., and an M.B.A. from the University of Tennessee. He has 25 years of experience in full-life-cycle development of information systems spanning a variety of hardware platforms, operating systems, and programming languages. His early career was in IBM mainframe installations, where he developed educational, manufacturing, and financial applications and served as a database administrator. He was the architect of Intersoft Systems' migration of its CPA client accounting software product from a minicomputer system to networked workstations. A senior systems analyst, he has developed environmental information systems and client/server and web-based accounting applications.

James E. Goetze is a senior systems analyst with 16 years of experience developing MIS. He obtained his B.S. in business administration from the University of Tennessee, Knoxville, and has experience in developing information systems for the DoD, DoE, and the commercial sector. He has been an employee of Q Systems for 2.5 years and was instrumental in locating problems and providing solutions to FoxPro database quirks.

Q Systems, Inc., a subsidiary of Advanced Resource Technologies, Inc. (*ARTI*), is a graduated 8(a) firm that provides information technology services and support, specializing in web-based development highlighted by corporate e-assessment, grants management, training management, and pre-employment testing applications. *ARTI* is a diversified information services provider and systems integrator, providing expertise in information systems and network engineering and program-management support.

2001-02 TBΠ SCHOLARSHIPS

The Tau Beta Pi Association Scholarship Program for senior-year study during the 2001-02 academic year will close January 15, 2001, when applications must be in the hands of Director of Fellowships D. Stephen Pierre Jr., P.E., Alabama Power Company, P.O. Box 2247, Mobile, AL 36652-2247.

Up to 12 cash awards, each worth $2,000, will be given in our third program. They will be made on the basis of high scholarship, strong recommendations, extracurricular activities, and financial need.

Winners will be selected by the Fellowship Board and notified before April 1. Junior members of Tau Beta Pi are eligible to apply. Applications are available from chapter presidents and at *www.tbp.org* .