

Using ColdFusion MX to Access CICS

A HostBridge™ White Paper



Toll-free: (866) 965-2427
Email: info@hostbridge.com

Copyright Notice

Copyright © 2002 by HostBridge Technology. All rights reserved. No part of this publication may be reproduced, transmitted, transcribed, stored in a retrieval system, or translated into any computer language, in any form or by any means, electronic, mechanical, magnetic, optical, chemical, manual, or otherwise, without prior written permission. You have limited permission to make hardcopy or other reproductions of any machine-readable documentation for your own use, provided that each such reproduction shall carry this copyright notice. No other rights under copyright are granted without prior written permission. The document is not intended for production and is furnished "as is" without warranty of any kind. All warranties on this document are hereby disclaimed including the warranties of merchantability and fitness for a particular purpose.

Revision date: 12/2/2002

Trademarks

HostBridge is a trademark of HostBridge Technology.
ColdFusion is a trademark of Macromedia, Inc.

Table of Contents

Overview: ColdFusion, CICS, and HostBridge	1
Types of CICS Applications	2
"Visual" vs. "Non-Visual" Transactions.....	2
Understanding CICS "Visual" Transactions.....	2
What is HostBridge?	3
Using the ColdFusion Trader Sample Application	4
Generating the Company Selection Menu.....	4
Building and Sending a URL to HostBridge	4
Handling the XML Response	5
Presenting a Real-Time Quote	6
Processing the HTTP POST	6
Handling the XML Response	7
Buying Shares	9
Invalid Input.....	11
Selling Shares.....	13
Production Considerations	13
Appendix A: CICS Trader Application.....	14
Appendix B: Complete Code for index.cfm	17

List of Figures and Screen Shots

Figure 1. CICS Application Access Taxonomy	2
Figure 3. Company Selection page	6
Figure 4. Real-Time Quote page	8
Figure 5. Buy Shares page	9
Figure 6. Transaction Aborted page	10
Figure 7. Error page shown for non-numeric data entered in numeric fields	11
Figure 8. Error page for numeric data out of range.....	12
Figure 9. Response page for successful transactions	12
Screen 1. Blank screen	14
Screen 2. Company Selection screen	15
Screen 3. Options screen	15
Screen 4. Real-time Quote screen.....	16
Screen 5. Session Over screen.....	16

Using ColdFusion MX to Access CICS

Using HostBridge, ColdFusion™ can easily create applications that include data from CICS. Together, HostBridge and ColdFusion allow organizations to transform their existing CICS applications into data sources that can be used by other eBusiness applications.

Overview: ColdFusion, CICS, and HostBridge

ColdFusion allows you to easily build Internet applications that integrate with databases, XML, web services, and more. ColdFusion MX empowers developers with a productive scripting environment and integrated search and charting capabilities. ColdFusion MX Server for J2EE Application Servers allow you to add the legendary ColdFusion ease of use to supported J2EE application servers.

However, while ColdFusion helps make it easier to build new applications, existing mainframe CICS applications have certainly not gone away. In fact, according to statistics from IBM and others, CICS has never been more successful:

- 30 years and \$1 trillion (per IDC) invested in CICS applications
- 14,000+ CICS customers worldwide
- 20,000+ CICS/390 licenses worldwide
- CICS is used by 490+ of IBM's top 500 customers
- 30 million end users of CICS applications
- 150,000+ concurrent users/system
- 5,000 CICS software packages from 2,000 ISVs
- 950,000 programmers earn their living from CICS
- CICS handles >30 billion transactions/day valued at >\$1 trillion/week

For companies with large investments in mainframe CICS applications, the ability to integrate these applications with products like ColdFusion MX is imperative.

HostBridge is a patent pending software product that XML-enables a broad class of existing CICS applications. HostBridge does this without requiring modification to the existing applications, and without screen-scraping. As a result, HostBridge is an ideal tool for integrating CICS applications with eXtend.

This White Paper presents a case study on how you can use HostBridge to integrate existing CICS applications with ColdFusion MX.

Types of CICS Applications

Not all CICS transactions operate the same way. As a result, the integration approach will depend on how the CICS transaction operates. The following diagram shows a high-level taxonomy of CICS transactions.

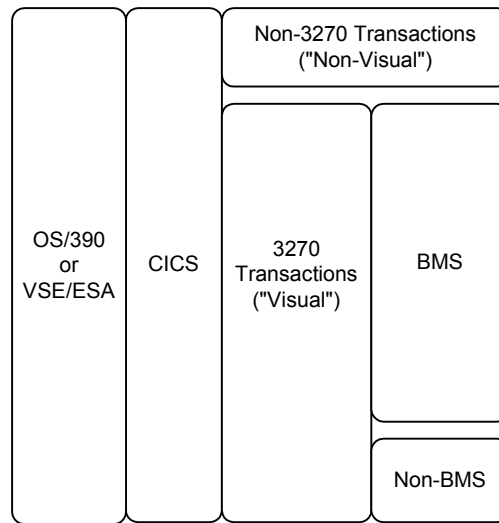


Figure 1. CICS Application Access Taxonomy

“Visual” vs. “Non-Visual” Transactions

CICS transactions fall into two broad categories: “visual” and “non-visual.” A “visual” transaction is one that expresses a presentation interface to an end-user at a terminal. You could also refer to a “visual” transaction as a “terminal-oriented” transaction. In contrast, “non-visual” transactions do **not** interact with an end-user. Instead, another program invokes these transactions. (This type of transaction is also referred to as “COMMAREA transaction” because the input/output parameters are passed to/from the transaction using an area of storage referred to as the “communication area,” or COMMAREA.)

The distinction between non-visual and visual transactions is important because integration possibilities exist for non-visual transactions that do not exist for visual transactions. As you might imagine, non-visual transactions are far easier to integrate with other programs than are visual transactions. However, visual transactions are far more common than non-visual transactions.

Understanding CICS “Visual” Transactions

In order to understand the integration possibilities for visual transactions, we need to further define this category.

CICS application developers have always had a number of choices in how to design their transaction to interact with an end-user at a terminal. The majority of applications use a component of CICS called Basic Mapping Support (BMS). BMS essentially handles the presentation logic of the transaction and relieves the application developer from having to encode and decode 3270 terminal data streams. The minority of applications that do not use BMS either include code to process 3270 data streams, or rely upon a non-IBM solution to handle presentation logic.

The distinction between BMS and non-BMS applications is important because integration possibilities exist for BMS applications that do not exist for non-BMS applications. For non-BMS applications, integration is based upon the terminal-oriented data stream generated by the application. As described elsewhere, this approach can have serious limitations.

What is HostBridge?

HostBridge is a patent pending software product that XML-enables existing CICS transactions. HostBridge runs under CICS on the mainframe. With HostBridge, any CICS transaction can be XML-enabled automatically. HostBridge does this without requiring modification to your existing applications, and without screen-scraping. As a result, HostBridge is the perfect tool for integrating CICS applications with ColdFusion.

Using the ColdFusion Trader Sample Application

Using ColdFusion and HostBridge, the sample application described in this section illustrates how to create an application that integrates with a CICS BMS application. (See Appendix A for a description of the CICS transaction used in this sample application.) The core of the sample application consists of one ColdFusion document named `index.cfm`, and two other files named `timeout.htm` and `hb_logo.gif` (both of which are static files).

At the top of the document, variables are that are used throughout the application are set, and HTML that is used throughout the application is displayed.

```
<cfset hbBase="http://192.168.10.232:3029/hostbridge" />
<cfset hbTimeout="600" />
<HTML>
<HEAD>
  <TITLE>Trader</TITLE>
  <meta http-equiv="refresh" content="#hbTimeout#; URL=timeout.htm">
</HEAD>
<BODY BGCOLOR="#####" TEXT="##000000">
<FONT FACE="Arial">
<IMG SRC="hb_logo.gif">
<P>
<H1>Share Trading Demonstration</H1>
```

Using the `hbBase` ColdFusion variable instead of the hostname throughout the document allows for an easy change of host and port number if needed. The `hbTimeout` variable sets the HostBridge session timeout value, as well as a refresh META tag in the document. This causes the page refresh to a static HTML page, `timeout.htm`, if the active HostBridge session times out from inactivity.

Generating the Company Selection Menu

Two form elements — “token” and “action” — control the flow inside of the ColdFusion document. When the ColdFusion application runs for the first time, these two elements are not yet set. The application displays the main menu and sets up a new HostBridge session because a token does not exist.

Building and Sending a URL to HostBridge

Our sample application builds an HTTP request and sends it to HostBridge, as shown in the code below.

```
<cfif len(trim(form.token)) is 0 or form.action is "Company Menu">
<h2>Company Selection</h2>
<!-- If a token is set, we'll use the session that is already available,
      otherwise, we'll start a new session to get a token and set the timeout --->
<cfif len(trim(form.token)) is 0>
  <cfhttp url="#hbBase#?hb_tranid=trad&hb_timeout=#hbTimeout#" method="get"
    resolveurl="no"/>
  <cfelse>
  <cfhttp url="#hbBase#?hb_token=#form.token#" method="get" resolveurl="no"/>
</cfif>
```

Below is the flow of the code segment above:

1. The token length after an initial transaction is 0, so the application recognizes this as the first time the application has been run.
2. Once the application enters the Company Selection block, it checks again to see if a token exists. If it does not exist, the application uses the `cfhttp` object to make an HTTP call to HostBridge. The `hbBase`

parameter will expand out to the base URL. The first invocation of HostBridge requires that a transaction be specified, which in this case is the “trad” transaction. The hbTimeout variable sets the timeout value for sessions.

3. If a token exists, such as in the case that the user returned to the main menu, the application would have simply performed a GET using the base URL and the token. This causes HostBridge to return the current XML and reset the timeout timer.

Handling the XML Response

After the cfhttp call, the cfhttp object receives the XML document returned by HostBridge. This document contains the field names and data for the Company Selection screen. (See Appendix A.)

Below is the flow for the code that handles the return document:

1. First, the token needs to be preserved for future HostBridge transactions.

```
<cfset mydoc = XmlParse( cfhttp.FileContent )/>
<cfset token = mydoc.XmlRoot.XmlChildren[1].XmlText/>
```

2. Next, a node is created to point to the various companies that exist in the application. The trader application only contains four companies. It would possible to make the application enumerate all elements that start with COMP, but for this example only COMP1 thru COMP4 are checked.

```
<cfset complnode = XmlSearch( mydoc,
    "/hostbridge/transaction/command/send_map/fields/field/value" )/>
<h3>Please select a company below</h3>
<form method="POST">
  <select name="companyid">
    <option value="1">#complnode[1].XmlText#</option>
    <option value="2">#complnode[2].XmlText#</option>
    <option value="3">#complnode[3].XmlText#</option>
    <option value="4">#complnode[4].XmlText#</option>
  </select>
  <p>
    <input type="hidden" name="token" value="#token#">
    <input type="submit" name="action" value="Real-Time Quote">
    <input type="submit" name="action" value="Buy Shares">
    <input type="submit" name="action" value="Sell Shares">
  </p>
</form>
```

3. The application generates and sends HTML output to the browser as shown in the figure below. The pull down box contains the names of the companies, which were obtained from the “value” attribute of each of the COMP1 thru COMP4 fields.

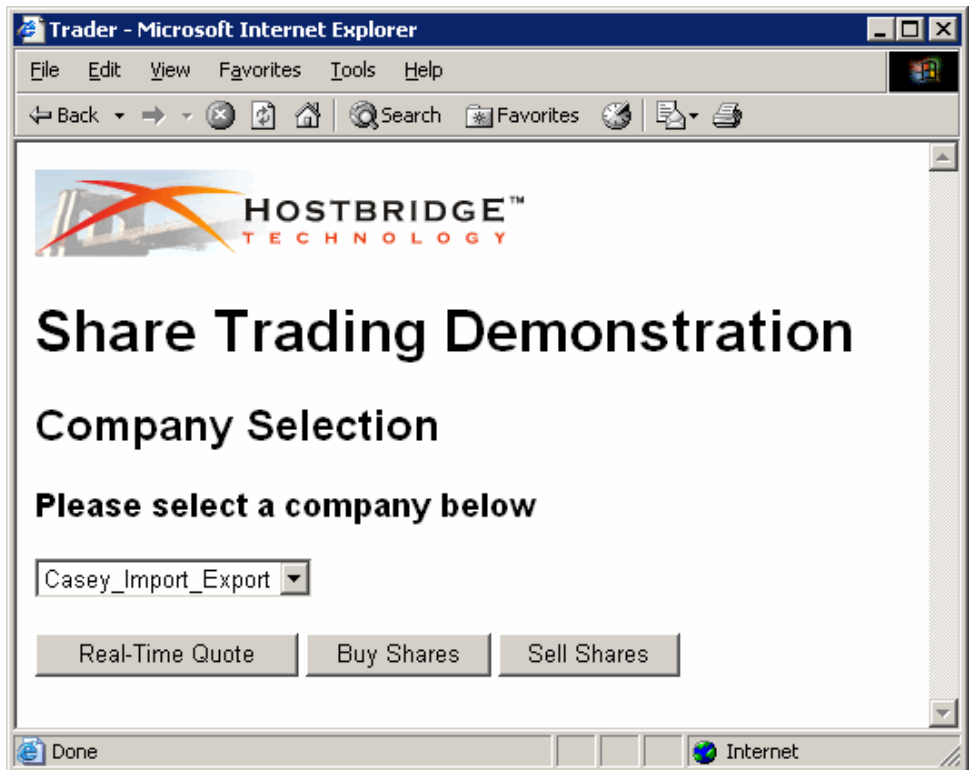


Figure 2. Company Selection page

Presenting a Real-Time Quote

After selecting a company and selecting “Real-Time Quote”, the browser performs an HTTP POST that passes the ID number of the company (1-4), the value of the action parameter (“Real-Time Quote”), and the HostBridge token, that the application passed to the browser as a hidden field in the form.

Processing the HTTP POST

The application receives the POST data and performs a series of actions based on the contents, as shown in the code below.

```
<cfelseif trim(form.action) is "Real-Time Quote">
<h2>Real-Time Quote</h2>

<!-- Have HostBridge set the company ID and run the transaction -->
<cfhttp url="#hbBase#?hb_token=#form.token#&option=#form.companyid#"
method="get" resolveurl="no"/>

<!-- Set OPT2 to 1, which gets us to the quote map -->
<cfhttp url="#hbBase#?hb_token=#form.token#&opt2=1" method="get" resolveurl="no"/>
```

Below is the flow of the code segment above:

1. The application checks to make sure the token exists and sets the action to “Real-Time Quote”.
2. To get to the Real-Time Quote information in the CICS application, the ColdFusion application traverses two screens. First, the CICS application requires that a company be selected. Once a company is selected, the CICS application presents options to request a quote, buy shares, or sell shares. The ColdFusion application presents all of these options to the user in the first menu, so it must walk through the two steps when branching out from the main menu.

3. The ColdFusion application sets the OPTION field in the first CICS screen to the company that was selected by performing an HTTP GET to HostBridge.
4. The CICS application presents the actions screen, and ColdFusion sets the OPT2 field to a value of "1", which requests a real-time quote.

After step 4, the ColdFusion application has executed the CICS trader application twice through HostBridge, without returning any HTML to the end user.

Handling the XML Response

The cfhttp object now contains the XML document returned by HostBridge that corresponds to the real-time quote screen. Below is a segment of code that shows how the ColdFusion application created a node for each field.

```
<cfset mydoc = XmlParse( cfhttp.FileContent )/>
<cfset companyNode = XmlSearch( mydoc,
"/hostbridge/transaction/command/send_map/fields/field[@name="COMP41"]/value" )/>
<cfset shrnowNode = XmlSearch( mydoc,
"/hostbridge/transaction/command/send_map/fields/field[@name="SHRNOW"]/value" )/>
<cfset share7Node = XmlSearch( mydoc,
"/hostbridge/transaction/command/send_map/fields/field[@name="SHARE7"]/value" )/>
<cfset share6Node = XmlSearch( mydoc,
"/hostbridge/transaction/command/send_map/fields/field[@name="SHARE6"]/value" )/>
<cfset share5Node = XmlSearch( mydoc,
"/hostbridge/transaction/command/send_map/fields/field[@name="SHARE5"]/value" )/>
```

Below is the flow of code segment above:

1. The ColdFusion application fills a table with the text from each node and creates a graph representing stock performance. The text values are referenced by the XmlText property of each node. For example, to reference the current stock value, #shrnowNode[1].XmlText# would be used. A screenshot of the browser output is shown below.

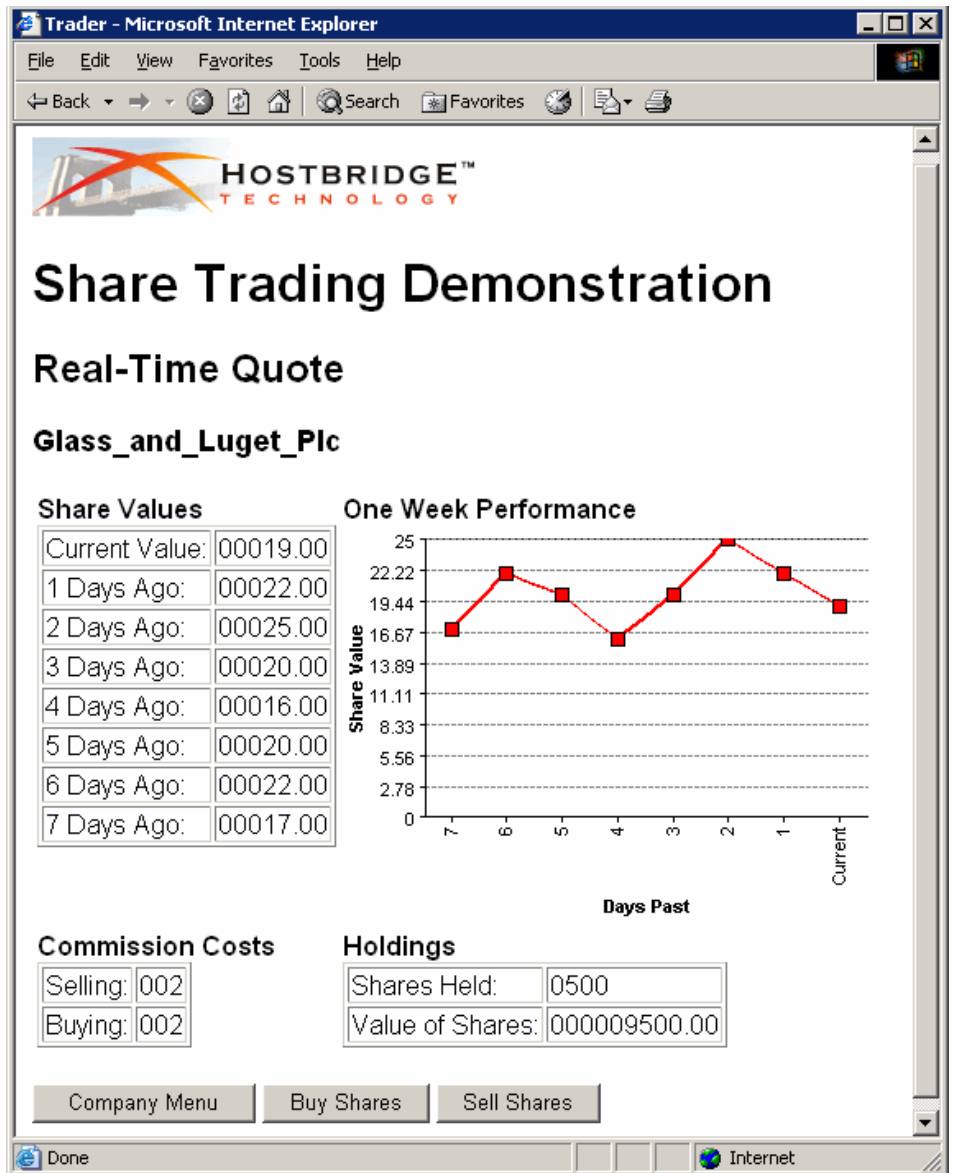


Figure 3. Real-Time Quote page

2. After the HTML is returned to the browser, the ColdFusion application drives the HostBridge session back to main company menu, as shown in the code below. This is done so that a buy, sell, or return to the main menu can be done from this screen. The form data for the company and token exist in the HTML, so a user could easily jump to any action from this screen. The actual CICS application requires the user to return up to the previous action menu to buy or sell shares.

```
<!-- Send PF3 twice in a row to back up to the main screen in the CICS application -->
<cfhttp url="#hbBase#?hb_token=#form.token#&hb_aid=PF3" method="get" resolveurl="no"/>
<cfhttp url="#hbBase#?hb_token=#form.token#&hb_aid=PF3" method="get" resolveurl="no"/>
```

Buying Shares

The buy and sell sections of the ColdFusion application are identical, differing only by labeling and the field names that differ in the CICS application.

```
<h2>Buy Shares</h2>
<!-- Set the company --->
<cfhttp url="#hbBase#?hb_token=#form.token#&option=#form.companyid#"
        method="get" resolveurl="no"/>

<!-- Select option 2, for buy --->
<cfhttp url="#hbBase#?hb_token=#form.token#&opt2=2" method="get" resolveurl="no"/>
<cfset mydoc = XmlParse( cfhttp.FileContent )/>

<!-- The company's name is stored in COMP51 in this map --->
<cfset companyNode = XmlSearch( mydoc,
    "/hostbridge/transaction/command/send_map/fields/field[@name=" "COMP51" "]/value" )/>
<h3>Buying shares of #companyNode[1].XmlText#</h3>
<form method="POST">
    Enter the number of shares that you wish to purchase:
    <input type="text" name="shares" value="">
    <p>
    <input type="hidden" name="companyid" value="#form.companyid#">
    <input type="hidden" name="token" value="#form.token#">
    <input type="submit" name="action" value="Complete Purchase">
    <input type="submit" name="action" value="Abort Purchase">
</form>
```

Below is the flow of the code segment above:

1. As in the Real-Time Quote code segment, the ColdFusion application selects the company by setting the OPTION field via HostBridge.
2. The application then sets the OPT2 field to 2, which corresponds to “buy shares”.
3. The application captures the data for field COMP51 in the XML document returned from HostBridge and uses the data to display the company name in the web browser as shown below.

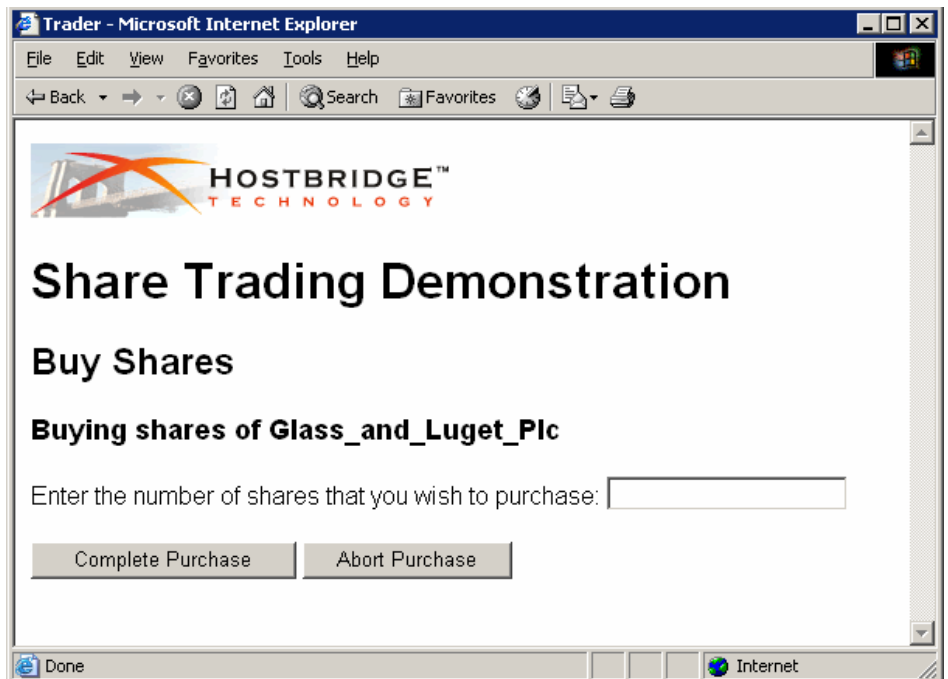


Figure 4. Buy Shares page

4. If the user selects Abort Purchase, the application informs the user that the purchase operation was cancelled and gives the user an opportunity to return to the main menu. Behind the scenes, ColdFusion drives the CICS application back to the main menu and returns the HTML shown in the code and figure below.

```
<!-- Send PF3 twice to get back to the main menu -->
<cfhttp url="#hbBase#?hb_token=#form.token#&hb_aid=PF3" method="get" resolveurl="no"/>
<cfhttp url="#hbBase#?hb_token=#form.token#&hb_aid=PF3" method="get" resolveurl="no"/>

<h2>Transaction Aborted</h2>

<form method="POST">
  <input type="hidden" name="token" value="#form.token#">
  <input type="submit" name="action" value="Company Menu">
</form>
```

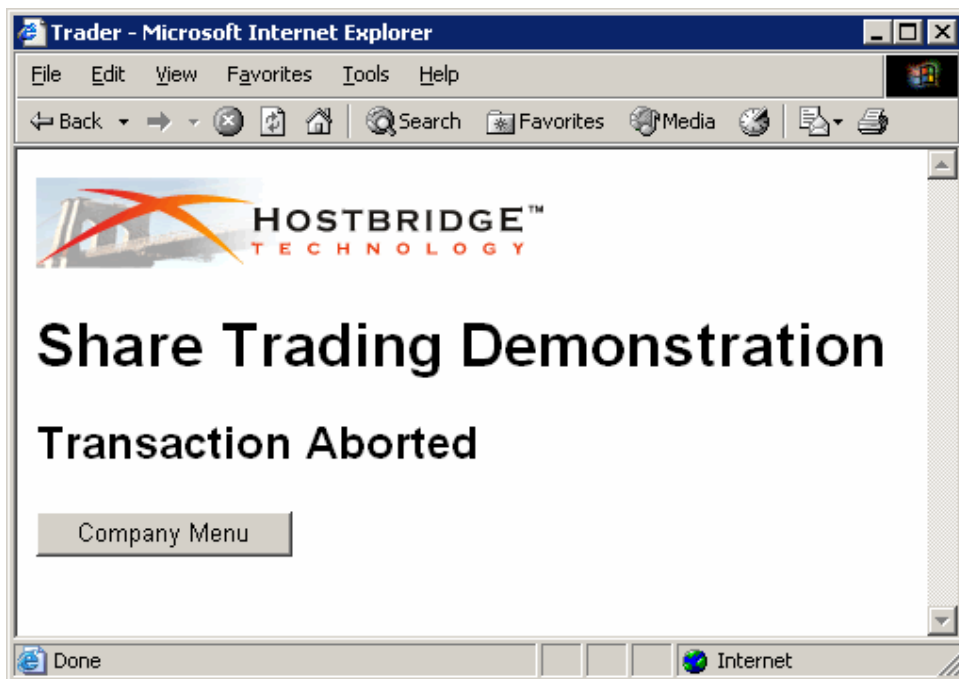


Figure 5. Transaction Aborted page

5. However, if the user selects "Complete Purchase", ColdFusion will set SHRBUY using HostBridge and complete the transaction.

```
<!-- Plug the shares parameter into the SHRBUY field in the CICS app -->
<cfhttp url="#hbBase#?hb_token=#form.token#&shrbuy=#form.shares#" method="get"
  resolveurl="no"/>
```

Invalid Input

When users submit invalid input, the ColdFusion application performs some checks to see what screen the CICS application is displaying before taking any action. The code for the checks appears below.

```
<cfset mydoc = XmlParse( cfhttp.FileContent )/>

<!-- If non-numeric data is entered, the app will stay on the same screen.
Otherwise if will go back to the options screen. We have to check to see
where we are. --->
<cfset mapNode = XmlSearch( mydoc, "/hostbridge/transaction/command/send_map/map" )/>

<!-- If we are still on T005, we have a problem --->
<cfif trim(#mapNode[1].XmlText#) is "T005">
<cfset mess5Node = XmlSearch( mydoc,
"/hostbridge/transaction/command/send_map/fields/field[@name="MESS5"]/value" )/>
<h2><font color="##FF0000">#mess5Node[1].XmlText#</font></h2>
<p>
The value that you entered, "#form.shares#", was invalid.

<!-- Back up one to get sync'ed up --->
<cfhttp url="#hbBase#?hb_token=#form.token#&hb_aid=PF3" method="get" resolveurl="no"/>
<cfelse>
<cfset mess3Node = XmlSearch( mydoc,
"/hostbridge/transaction/command/send_map/fields/field[@name="MESS3"]/value" )/>

<cfif #mess3Node[1].XmlText# is "Request Completed OK">
<h2>Your requested purchase of #form.shares# was completed successfully</h2>
<cfelse>
<h2><font color="##FF0000">Your requested purchase of #form.shares# was
NOT completed successfully.</font></h2>
<p>
Please check the requested amount to make sure that your entry was valid.
</cfif>
</cfif>

<!-- Back out to the main menu --->
<cfhttp url="#hbBase#?hb_token=#form.token#&hb_aid=PF3" method="get" resolveurl="no"/>
```

- If non-numeric data is submitted, the CICS application stays on the current map, "T005" and displays an error to the user, as shown below.

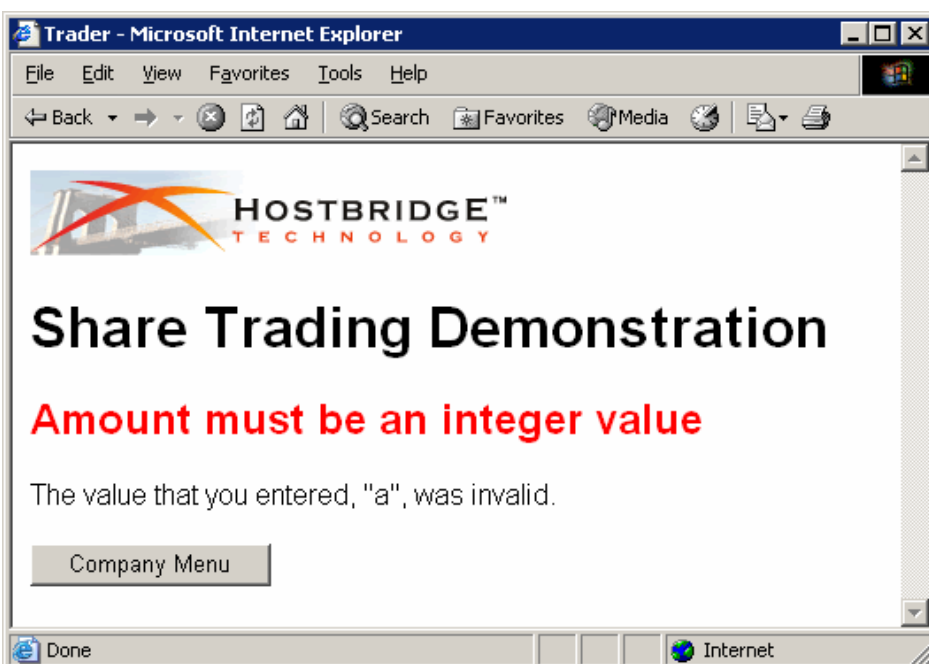


Figure 6. Error page shown for non-numeric data entered in numeric fields

- If a numeric value is submitted that is out of range (for example, if the user tried to buy more shares than is allowable), the application moves beyond “T005” but displays an error message, as shown below.

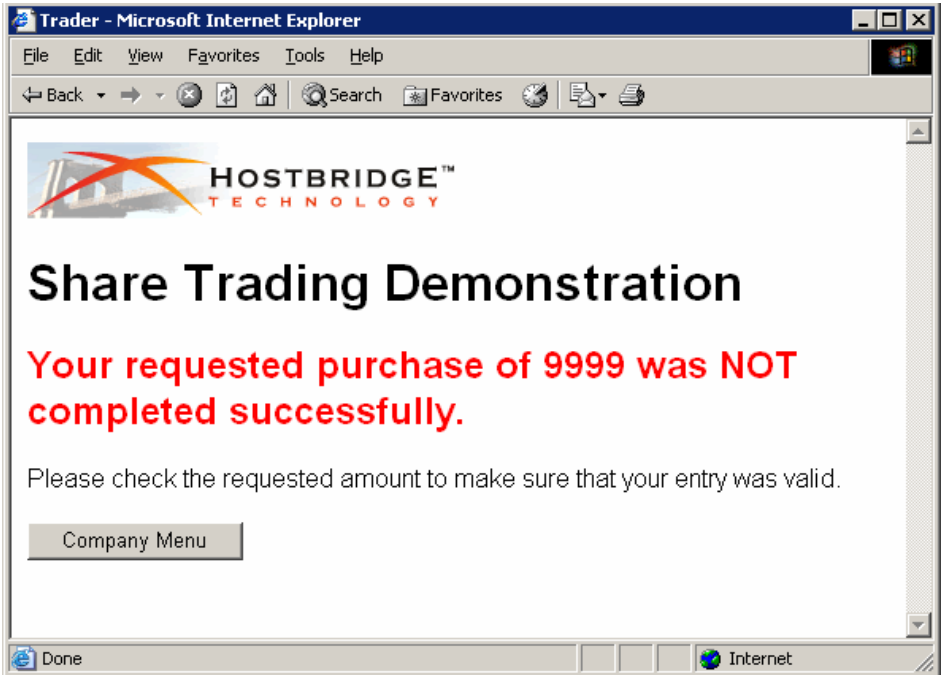


Figure 7. Error page for numeric data out of range

- If the application stays on T005, the sample application backs out one menu level using PF3 to stay in sync and notifies the user that the transaction was successful.

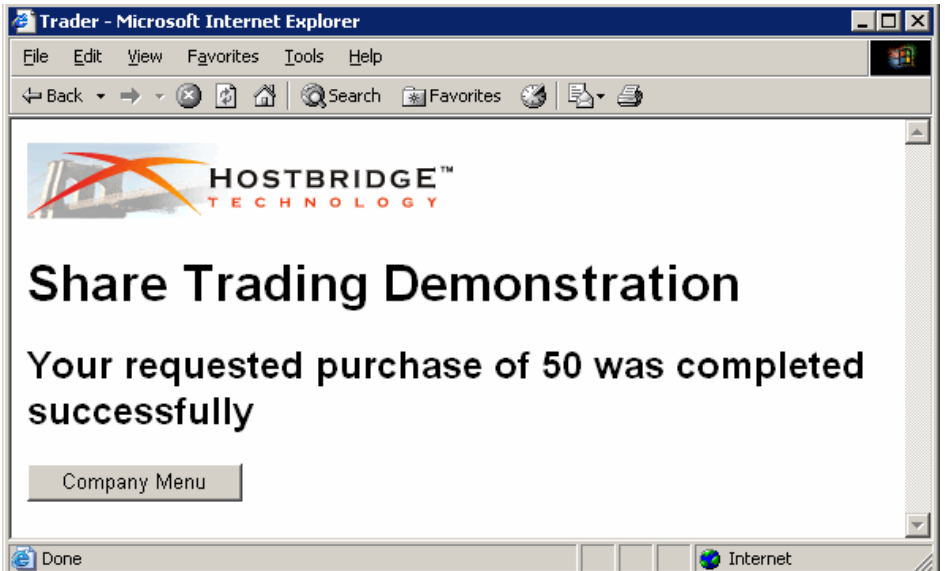


Figure 8. Response page for successful transactions

Selling Shares

The code that handles the selling of shares is identical to the code that handles purchases, with the exception of the change in labels and field names.

Production Considerations

To limit the complexity of this sample application, we did not include steps to verify that the CICS application is on expected screen. It is possible to disrupt the flow of this sample by using the Forward, Back, and Refresh buttons in the web browser.

Production environments should include routines that would check the current XML return from HostBridge for the correct MAP name or any ABENDs that may be received. If unexpected values are encountered, the application could take the required steps to get to the correct screen, or return a proper error message to the end user.

Appendix A: CICS Trader Application

IBM provides a sample CICS BMS application that simulates a stock trading application. The Share Trading Demonstration, or TRADER, consists of only a few screens that allow you to choose a company, get a stock quote, or buy/sell shares. The case study presented in this White Paper uses this application. The application transactions are pseudo-conversational and use the BMS commands SEND MAP and RECEIVE MAP to communicate with the end-user. The application is simple to use. The example below shows how to get a stock quote for Casey Import Export using the demonstration application.

1. Users begin at a blank screen. They type `trad` then press Enter to start the transaction and display the Company Selection screen.



Screen 1. Blank screen

2. The Company Selection screen allows users to select the company whose stock you want to act upon. The next transaction expects to receive a number with a value of 1, 2, 3, or 4. Enter 1 to select Casey Import Export.

```
Share Trading Demonstration                                TRADER.T002
Share Trading Manager: Company Selection

1. Casey_Import_Export
2. Glass_and_Luget_Plc
3. Headworth_Electrical
4. IBM

Please select a company (1,2,3 or 4) : 1_

-----
PF3=Exit                                                PF12=Exit
```

Screen 2. Company Selection screen

3. The Options screen appears. This screen allows users to get a stock quote, buy shares, or sell shares. The transaction expects to receive a number with a value of 1, 2, or 3. Enter 1 to retrieve the stock quote.

```
Share Trading Demonstration                                TRADER.T003
Share Trading Manager: Options

1. New Real-Time Quote
2. Buy Shares
3. Sell Shares

Please select an option (1,2 or 3): _

-----
PF3=Return                                              PF12=Exit
```

Screen 3. Options screen

- The Real-time Quote screen appears. This screen displays the stock quote for the selected company. The transaction now expects the user to press either the PF3 key to conduct a new set of transactions or the PF12 key to exit the application. Press PF12 to exit.

```
Share Trading Demonstration                                TRADER.T004
Share Trading Manager: Real-Time Quote

User Name:      RUSS
Company Name:   Casey_Import_Export

Share Values:
NOW:            00079.00
1 week ago:    00059.00
6 days ago:    00063.00
5 days ago:    00065.00
4 days ago:    00070.00
3 days ago:    00072.00
2 days ago:    00078.00
1 day ago:     00077.00

Commission Cost:
for Selling:    007
for Buying:     010

Number of Shares Held: 5693
Value of Shares Held: 000449747.00

Request Completed OK
-----
PF3=Return                                           PF12=Exit
```

Screen 4. Real-time Quote screen

- A message appears on the terminal screen to indicate the session is over.

```
Trader: Session Over
```

Screen 5. Session Over screen

Appendix B: Complete Code for index.cfm

The following is the complete code used to create the sample application documented in this white paper.

```
<!---
HostBridge ColdFusion Sample
(c)2002, HostBridge Technology

This sample code requires the IBM trader CICS application
--->

<cfoutput>

<!--- Try to prevent the browser from caching --->
<cfheader name = "Expires" value = "#Now()#">

<!--- Initialize parameters with defaults just in case --->
<cfparam name="form.action" default="">
<cfparam name="companyid" default="">
<cfparam name="form.token" default="">

<!--- Host and timeout settings can be adjusted here --->
<cfset hbBase="http://192.168.10.232:3029/hostbridge" />
<cfset hbTimeout="600" />

<HTML>
<HEAD>
<TITLE>Trader</TITLE>
<meta http-equiv="refresh" content="#hbTimeout#; URL=timeout.htm">
</HEAD>
<BODY BGCOLOR="##FFFFFF" TEXT="##000000">
<FONT FACE="Arial">
<IMG SRC="hb_logo.gif">
<P>
<H1>Share Trading Demonstration</H1>

<!--- If a token is not received as a parameter, we are being run for the first time.
Also, if the action is "Company Menu", we want to display the first screen just
as if we are starting a new session --->
<cfif len(trim(form.token)) is 0 or form.action is "Company Menu">

    <h2>Company Selection</h2>

    <!--- If a token is set, we'll use the session that is already available,
    otherwise, we'll start a new session to get a token and set the timeout --->
    <cfif len(trim(form.token)) is 0>
        <cfhttp url="#hbBase#?hb_tranid=trad&hb_timeout=#hbTimeout#" method="get"
resolveurl="no"/>
    <cfelse>
        <cfhttp url="#hbBase#?hb_token=#form.token#" method="get" resolveurl="no"/>
    </cfif>

    <cfset mydoc = XmlParse( cfhttp.FileContent )/>
    <cfset token = mydoc.XmlRoot.XmlChildren[1].XmlText/>

    <cfset complnode = XmlSearch( mydoc,
"/hostbridge/transaction/command/send_map/fields/field/value" )/>

    <!--- The Trader app's list of companies never change. It would be possible to enumerate
    through all of the companies one by one checking for COMP* instead of statically checking
    for four.
    Here we will just show the first four fields to show how to walk through them. --->

    <h3>Please select a company below</h3>
    <form method="POST">
        <select name="companyid">
```

```

        <option value="1">#complnode[1].XmlText#</option>
        <option value="2">#complnode[2].XmlText#</option>
        <option value="3">#complnode[3].XmlText#</option>
        <option value="4">#complnode[4].XmlText#</option>
    </select>
<p>

<input type="hidden" name="token" value="#token#">
<input type="submit" name="action" value="Real-Time Quote">
<input type="submit" name="action" value="Buy Shares">
<input type="submit" name="action" value="Sell Shares">

</form>

<!-- REAL-TIME QUOTE processing --->
<cfelseif trim(form.action) is "Real-Time Quote">

    <h2>Real-Time Quote</h2>

    <!-- Have HostBridge set the company ID and run the transaction --->
    <cfhttp url="#hbBase#?hb_token=#form.token#&option=#form.companyid#" method="get"
    resolveurl="no"/>

    <!-- Set OPT2 to 1, which gets us to the quote map --->
    <cfhttp url="#hbBase#?hb_token=#form.token#&opt2=1" method="get" resolveurl="no"/>

    <cfset mydoc = XmlParse( cfhttp.FileContent )/>

    <!-- Here we set a Node for each field we need, so that application changes won't break
        the Cold Fusion app. It is possible to use a higher level xpath to get all fields in
        an array, but a new field inserted into the map will throw off the ordering since each
        would be referenced via an array index --->
    <cfset companyNode = XmlSearch( mydoc,
"/hostbridge/transaction/command/send_map/fields/field[@name="COMP41"]/value" )/>

    <cfset shrnwNode = XmlSearch( mydoc,
"/hostbridge/transaction/command/send_map/fields/field[@name="SHRNW"]/value" )/>
    <cfset share7Node = XmlSearch( mydoc,
"/hostbridge/transaction/command/send_map/fields/field[@name="SHARE7"]/value" )/>
    <cfset share6Node = XmlSearch( mydoc,
"/hostbridge/transaction/command/send_map/fields/field[@name="SHARE6"]/value" )/>
    <cfset share5Node = XmlSearch( mydoc,
"/hostbridge/transaction/command/send_map/fields/field[@name="SHARE5"]/value" )/>
    <cfset share4Node = XmlSearch( mydoc,
"/hostbridge/transaction/command/send_map/fields/field[@name="SHARE4"]/value" )/>
    <cfset share3Node = XmlSearch( mydoc,
"/hostbridge/transaction/command/send_map/fields/field[@name="SHARE3"]/value" )/>
    <cfset share2Node = XmlSearch( mydoc,
"/hostbridge/transaction/command/send_map/fields/field[@name="SHARE2"]/value" )/>
    <cfset share1Node = XmlSearch( mydoc,
"/hostbridge/transaction/command/send_map/fields/field[@name="SHARE1"]/value" )/>

    <cfset sellNode = XmlSearch( mydoc,
"/hostbridge/transaction/command/send_map/fields/field[@name="SELL"]/value" )/>
    <cfset buyNode = XmlSearch( mydoc,
"/hostbridge/transaction/command/send_map/fields/field[@name="BUY"]/value" )/>

    <cfset heldNode = XmlSearch( mydoc,
"/hostbridge/transaction/command/send_map/fields/field[@name="HELD"]/value" )/>
    <cfset valueNode = XmlSearch( mydoc,
"/hostbridge/transaction/command/send_map/fields/field[@name="VALUE"]/value" )/>

    <cfset mess4Node = XmlSearch( mydoc,
"/hostbridge/transaction/command/send_map/fields/field[@name="MESS4"]/value" )/>

    <h3>#companyNode[1].XmlText#</h3>
    <p>
    <table border="0">
        <tr><td valign="top">
            <B>Share Values</B><br>
            <table border="1">
                <tr><td>Current Value: </td><td>#shrnwNode[1].XmlText#</td></tr>
                <tr><td>1 Days Ago: </td><td>#share1Node[1].XmlText#</td></tr>
                <tr><td>2 Days Ago: </td><td>#share2Node[1].XmlText#</td></tr>
                <tr><td>3 Days Ago: </td><td>#share3Node[1].XmlText#</td></tr>
            </table>
        </td></tr>
    </table>

```

```
|<td>4 Days Ago: </td><td>#share4Node[1].XmlText#</td></tr>
|<td>5 Days Ago: </td><td>#share5Node[1].XmlText#</td></tr>
|<td>6 Days Ago: </td><td>#share6Node[1].XmlText#</td></tr>
|<td>7 Days Ago: </td><td>#share7Node[1].XmlText#</td></tr>
</table>
</td>
<td>

<B>One Week Performance</B><br>
<cfchart
  xAxisTitle="Days Past"
  yAxisTitle="Share Value"
  format="png">
  <!-------
  Use cfchartdata to define the elements of the chart
  ----->
  <cfchartseries type="line">
    <cfchartdata item="7" value="#share7Node[1].XmlText#">
    <cfchartdata item="6" value="#share6Node[1].XmlText#">
    <cfchartdata item="5" value="#share5Node[1].XmlText#">
    <cfchartdata item="4" value="#share4Node[1].XmlText#">
    <cfchartdata item="3" value="#share3Node[1].XmlText#">
    <cfchartdata item="2" value="#share2Node[1].XmlText#">
    <cfchartdata item="1" value="#share1Node[1].XmlText#">
    <cfchartdata item="Current" value="#shrnowNode[1].XmlText#">

  </cfchartseries>
</cfchart>

</td>
</tr><tr>
<td valign="top">
<B>Commission Costs</B><br>
<table border="1">
  <tr><td>Selling: </td><td>#sellNode[1].XmlText#</td></tr>
  <tr><td>Buying: </td><td>#buyNode[1].XmlText#</td></tr>
</table>
</td>
<td valign="top">
<B>Holdings</B>
<table border="1">
  <tr><td>Shares Held: </td><td>#heldNode[1].XmlText#</td></tr>
  <tr><td>Value of Shares: </td><td>#valueNode[1].XmlText#</td></tr>
</table>
</td>
</tr>
</table>
<p>

<!-- Send PF3 twice in a row to back up to the main screen in the CICS application --->
<cfhttp url="#hbBase#?hb_token=#form.token#&hb_aid=PF3" method="get" resolveurl="no"/>
<cfhttp url="#hbBase#?hb_token=#form.token#&hb_aid=PF3" method="get" resolveurl="no"/>

<!-- Since we already have the company ID, we can easy jump into the buy or sell
screens easily --->

<form method="POST">
<input type="hidden" name="companyid" value="#form.companyid#">
<input type="hidden" name="token" value="#form.token#">
<input type="submit" name="action" value="Company Menu">
<input type="submit" name="action" value="Buy Shares">
<input type="submit" name="action" value="Sell Shares">
</form>

<!-- BUY SHARES --->
<cfelseif trim(form.action) is "Buy Shares">

  <h2>Buy Shares</h2>

  <!-- Set the company --->
  <cfhttp url="#hbBase#?hb_token=#form.token#&option=#form.companyid#" method="get"
  resolveurl="no"/>
  <!-- Select option 2, for buy --->
  <cfhttp url="#hbBase#?hb_token=#form.token#&opt2=2" method="get" resolveurl="no"/>

  <cfset mydoc = XmlParse( cfhttp.FileContent )/>

|  |

|  |

|  |

|  |

```

```

<!-- The company's name is stored in COMP51 in this map --->
<cfset companyNode = XmlSearch( mydoc,
"/hostbridge/transaction/command/send_map/fields/field[@name="COMP51"])/value" )/>

<h3>Buying shares of #companyNode[1].XmlText#</h3>

<form method="POST">
Enter the number of shares that you wish to purchase: <input type="text" name="shares" value="">
<p>
<input type="hidden" name="companyid" value="#form.companyid#">
<input type="hidden" name="token" value="#form.token#">
<input type="submit" name="action" value="Complete Purchase">
<input type="submit" name="action" value="Abort Purchase">
</form>

<!-- SELL SHARES --->
<cfelseif trim(form.action) is "Sell Shares">

<h2>Sell Shares</h2>

<!-- Set company --->
<cfhttp url="#hbBase#?hb_token=#form.token#&option=#form.companyid#" method="get"
resolveurl="no"/>
<!-- Select the sell option --->
<cfhttp url="#hbBase#?hb_token=#form.token#&opt2=3" method="get" resolveurl="no"/>

<cfset mydoc = XmlParse( cfhttp.FileContent )/>

<!-- Company is in COMP61 in this map --->
<cfset companyNode = XmlSearch( mydoc,
"/hostbridge/transaction/command/send_map/fields/field[@name="COMP61"])/value" )/>

<h3>Selling shares of #companyNode[1].XmlText#</h3>

<form method="POST">
Enter the number of shares that you wish to sell: <input type="text" name="shares" value="">
<p>
<input type="hidden" name="companyid" value="#form.companyid#">
<input type="hidden" name="token" value="#form.token#">
<input type="submit" name="action" value="Complete Sale">
<input type="submit" name="action" value="Abort Sale">
</form>

<!-- ABORT SALE and ABORT PURCHASE --->
<!-- Both functions are handled by the same code block --->
<cfelseif trim(form.action) is "Abort Sale" or trim(form.action) is "Abort Purchase">

<!-- Send PF3 twice to get back to the main menu --->
<cfhttp url="#hbBase#?hb_token=#form.token#&hb_aid=PF3" method="get" resolveurl="no"/>
<cfhttp url="#hbBase#?hb_token=#form.token#&hb_aid=PF3" method="get" resolveurl="no"/>

<h2>Transaction Aborted</h2>

<form method="POST">
<input type="hidden" name="token" value="#form.token#">
<input type="submit" name="action" value="Company Menu">
</form>

<!-- COMPLETE PURCHASE --->
<cfelseif trim(form.action) is "Complete Purchase">

<!-- Plug the shares parameter into the SHRBUY field in the CICS app --->
<cfhttp url="#hbBase#?hb_token=#form.token#&shrbuy=#form.shares#" method="get" resolveurl="no"/>

<cfset mydoc = XmlParse( cfhttp.FileContent )/>

<!-- If non-numeric data is entered, the app will stay on the same screen. Otherwise
if will go back to the options screen. We have to check to see where we are. --->
<cfset mapNode = XmlSearch( mydoc, "/hostbridge/transaction/command/send_map/map" )/>

<!-- If we are still on T005, we have a problem --->
<cfif trim(#mapNode[1].XmlText#) is "T005">
<cfset mess5Node = XmlSearch( mydoc,
"/hostbridge/transaction/command/send_map/fields/field[@name="MESS5"])/value" )/>
<h2><font color="##FF0000">#mess5Node[1].XmlText#</font></h2>

```

```

        <p>
        The value that you entered, "#form.shares#", was invalid.
        <!-- Back up one to get sync'ed up --->
        <cfhttp url="#hbBase#?hb_token=#form.token#&hb_aid=PF3" method="get" resolveurl="no"/>
    <cfelse>
        <cfset mess3Node = XmlSearch( mydoc,
"/hostbridge/transaction/command/send_map/fields/field[@name="MESS3"]/value" )/>

        <cfif #mess3Node[1].XmlText# is "Request Completed OK">
            <h2>Your requested purchase of #form.shares# was completed successfully</h2>
        <cfelse>
            <h2><font color="##FF0000">Your requested purchase of #form.shares# was NOT
completed successfully.</font></h2>
            <p>
                Please check the requested amount to make sure that your entry was valid.
            </p>
        </cfif>
    </cfelse>

    <!-- Back out to the main menu --->
    <cfhttp url="#hbBase#?hb_token=#form.token#&hb_aid=PF3" method="get" resolveurl="no"/>

    <form method="POST">
    <input type="hidden" name="token" value="#form.token#">
    <input type="submit" name="action" value="Company Menu">
    </form>

<!-- COMPLETE SALE --->
<!-- This code is identical to the code for COMPLETE PURCHASE, which slightly differing field names ---
>
<cfelseif trim(form.action) is "Complete Sale">

    <cfhttp url="#hbBase#?hb_token=#form.token#&shrsell=#form.shares#" method="get"
resolveurl="no"/>

    <cfset mydoc = XmlParse( cfhttp.FileContent )/>

    <cfset mapNode = XmlSearch( mydoc, "/hostbridge/transaction/command/send_map/map" )/>

    <cfif trim(#mapNode[1].XmlText#) is "T006">
        <cfset mess6Node = XmlSearch( mydoc,
"/hostbridge/transaction/command/send_map/fields/field[@name="MESS6"]/value" )/>
        <h2><font color="##FF0000">#mess6Node[1].XmlText#</font></h2>
        <p>
            The value that you entered, "#form.shares#", was invalid.
        <cfhttp url="#hbBase#?hb_token=#form.token#&hb_aid=PF3" method="get" resolveurl="no"/>
    <cfelse>
        <cfset mess3Node = XmlSearch( mydoc,
"/hostbridge/transaction/command/send_map/fields/field[@name="MESS3"]/value" )/>

        <cfif #mess3Node[1].XmlText# is "Request Completed OK">
            <h2>Your requested sale of #form.shares# was completed successfully</h2>
        <cfelse>
            <h2><font color="##FF0000">Your requested sale of #form.shares# was NOT
completed successfully.</font></h2>
            <p>
                Please check the requested amount to make sure that your entry was valid.
            </p>
        </cfif>
    </cfelse>

    <cfhttp url="#hbBase#?hb_token=#form.token#&hb_aid=PF3" method="get" resolveurl="no"/>

    <form method="POST">
    <input type="hidden" name="token" value="#form.token#">
    <input type="submit" name="action" value="Company Menu">
    </form>

</cfif>

</cfoutput>
</BODY>
</FONT>
</HTML>

```