U.S. Small Business Administration
Office of the Chief Information Officer
Office of Information Systems Support

SBA ColdFusion Programming Standards

Version: 3.2.2
Modified: 08/25/2008

TABLE OF CONTENTS:

1 Introduction 7
1.1 Revision History 8
2 Naming Conventions 10
2.1 File Names 10
2.1.1 Display Files (dsp_ prefix) 10
2.1.2 Action Files (act_ prefix) 10
2.1.3 Use CFLOCATION to Pass Off from an Action File to a Display File 10
2.1.4 Utility Files (various prefixes) 11
2.1.5 Always Match Case in File Names 11
2.1.6 Backup Files 11
2.2 Variable Names 12
2.2.1 Database Column Names 12
2.2.2 Datasource Names 12
2.2.3 Temporary Control Variables 12
2.2.4 Logic Variables 12
2.2.5 XML Variables 13
2.2.6 Standardized Variable Names Used by Shared Code 13
3 Coding Standards, Application-Specific Code 14
3.1 Application Model 14
3.1.1 “Thin Client” and Client-Side Data Validation 14
3.1.2 Server-Side Data Validation 14
3.1.3 Standardized Look-and-Feel 14
3.1.4 Our Goal Is 50% Shared Code 14
3.1.5 Externally Configurable Code 15
3.2 Application.cfm 17
3.2.1 When and Where Required 17
3.2.2 When Application.cfm Is Allowed in Subdirectories 17
3.2.3 Extending Application.cfm in a Subdirectory 18
3.2.4 Initialization 18
3.2.5 More on Initialization – Variables Scope versus Request Scope 19
3.2.6 Set Request.Version to Identify your Application’s Version Number 19
3.2.7 Never Use Client Scope – Requires a Waiver 19
3.2.8 No Longer Any Need to Encrypt Application.cfm 19
3.2.9 Session Control (CF 4.x and 5.x) 20
3.2.10 Session Control (CFMX) 21
3.2.11 Session Timeout 21
3.2.12 Session Conflicts in GLS 22
3.3 Security 24
3.3.1 Referrer Checks 24
3.3.2 Logins (Usernames and Passwords) 24
3.3.3 Data Validation for SQL 24
3.3.4 Shared (or “Generic”) Logins 25
3.3.5 Program Descriptions (Also Known As “Comment Headers”) 25
3.3.6 <form … method="post"> 25
3.3.7 Cookies 26
3.3.8 File Upload Restrictions 26
3.4 Database 27

3.4.1 Structured Query Language (SQL) versus Stored Procedure Calls 27
3.4.2 Use CFTRANSACTION, not CFLOCK, to Lock Database Changes 27
3.5 Miscellaneous 28
3.5.1 Browser Support (HTML, CSS and JavaScript) 28
3.5.2 Section 508 Support 28
4 Coding Standards, Shared Code 29
4.1 SBA Look-and-Feel 30
4.1.1 Screen Snapshot of SBA Look-and-Feel, Showing Page Regions 30
4.1.2 Regions of the Page and What They're Called 31
4.1.3 Which Regions are Optional 31
4.1.4 How to Call the SBA Look-and-Feel Custom Tag 32
4.1.5 Controlling the MainNav Buttons with the Show Attribute 33
4.1.6 How to Specify Inline HTML versus Frames 34
4.1.7 When to Use Inline HTML and When to Use Frames 35
4.1.8 What Happens When MainNav Is NOT a Frame 35
4.1.9 What Happens When MainNav IS a Frame 35
4.1.10 HOW to Use Inline HTML and HOW to Use Frames 36
4.1.11 What CSS Class Names to Use 40
4.1.12 The Screen Resizing Feature 41
4.1.13 The TextOnly Feature 41
4.1.14 The Automatic TextOnly Feature 42
4.1.15 Form Data Recovery 43
4.1.16 Features Requiring Some Knowledge of JavaScript 46
4.1.17 MainNav as a Frame 52
4.1.18 Using SBA Look-and-Feel on a Static HTML Page 53
4.1.19 Read the Custom Tags to Get More Information 54
4.2 Stored Procedure Call Files 55
4.2.1 Make Sure that the SPC Files Have Been Generated 55
4.2.2 Request Regeneration of SPC Files Whenever Parameter Lists Change 55
4.2.3 Load Only the Columns You Need into the Variables Scope 55
4.2.4 But Use Defaults Sensibly 55
4.2.5 Use LogAct to make error messages more user-friendly 56
4.2.6 Use Variables.TxnErr for Transaction Control 56
4.2.7 Retrieving Single Result Sets 56
4.2.8 Retrieving Multiple Result Sets 57
4.2.9 Calling a Stored Procedure in a Different Database 58
4.2.10 How to use it 59
4.3 Logging 60
4.3.1 Turning On Logging Support – The "Master Switch" 60
4.3.2 What to Use as the System Name – GLS Systems 61
4.3.3 What to Use as the System Name – Non-GLS Systems 61
4.3.4 All Developers Will Be Application Administrators in Development 62
4.3.5 The CF/Logging Admin Pages 62
4.3.6 Logging Levels – Debug, Info, Warn, Error and Fatal 63
4.3.7 Manual Logging Routines That You're Required To Add 63
4.3.8 Manual Logging Routines That Are Optional 65
4.3.9 Where the Log Files Reside 66
4.3.10 Cooperating With Other Developers in Development 66
4.4 Standard Callbacks 67
4.4.1 dsp_LookupZipToDropdown.cfm 67
4.4.2 dsp_LookupZipToDropdown.ajax.cfm 69
4.4.3 dsp_LookupNAICSDescTxt.ajax.cfm 70

4.4.4 get_ArrayUserRoles.cfm 71
4.4.5 get_GLSSession.cfm 72
4.4.6 Future Callbacks 73
4.5 Standard CFIncludes 74
4.5.1 bld_ServerCachedQueries.cfm 74
4.5.2 dsp_errmsg.cfm 76
4.5.3 dsp_options.cfm 76
4.5.4 dsp_sbalookandfeel_variables.cfm 78
4.5.5 get_actual_server_name.cfm 78
4.5.6 get_sbalookandfeel_variables.cfm 78
4.5.7 inc_starttickcount.cfm 78
4.5.8 inc_totaltickcount.cfm 78
4.5.9 OnRequestEnd.cfm 78
4.5.10 put_sbalookandfeel_messages.cfm 78
4.5.11 put_sbalookandfeel_variables.cfm 78
4.6 Standard JavaScripts 79
4.6.1 Use onChange, Not onBlur 79
4.6.2 Code for Reuse in the Form's onSubmit 79
4.6.3 EditDate 81
4.6.4 EditDateNonFuture 81
4.6.5 EditList 81
4.6.6 EditMask 81
4.6.7 EditPronetUserid 81
4.6.8 EditState 81
4.6.9 EditTin 81
4.6.10 ClearForm 81
4.6.11 DumpObject 81
4.6.12 FormSynopsis 81
4.6.13 GetXMLHttpRequest 81
4.6.14 LookupNAICSDescTxt 81
4.6.15 LookupZipToDropdown 82
4.6.16 NumToDollars 82
4.6.17 RoundTo2DecimalPlaces 82
4.6.18 RoundToNearest 82
4.6.19 RoundUpToNearest 82
4.6.20 SetFormEltValue 82
4.7 Standard UDFs and Other Utilities 83
4.7.1 bld_GetCFDirectoryActionList.cfm 83
4.7.2 bld_GetCFFileActionRead.cfm 83
4.7.3 bld_JaguarUDFs.cfm 83
4.7.4 bld_ListToArrayAllowingNulls.cfm 83
4.7.5 bld_ProcessDirectory.cfm 83
4.7.6 val_char.cfm 83
4.7.7 val_date.cfm 83
4.7.8 val_email.cfm 83
4.7.9 val_num.cfm 83
4.7.10 val_phone.cfm 83
4.7.11 val_state.cfm 83
4.7.12 val_taxid.cfm 83
4.7.13 val_url.cfm 83
4.7.14 val_zip.cfm 83
5 Best Practices 84

5.1 Improving Performance 84
5.1.1 Eliminate Redundancies, Share Code 84
5.1.2 Eliminate Redundancies, Share Code, part 2 85
5.1.3 Limit Record Set Size 85
5.1.4 Caching Result Sets 85
5.1.5 Explicitly Scope Variables 85
5.2 Code for Ease of Maintenance 86
5.2.1 Parameterize Directory Names and Paths 86
5.2.2 Indent Properly 87
5.2.3 Line Up Code to Make It Easier to Read and Spot Errors 87
5.2.4 Define Configuration Parameters at Top of Page 88
5.2.5 Make LOTS of Things Configurable 88
5.2.6 Document Your Code: Use “Hints” 89
5.2.7 Document Your Code: Use Comments for Actual Comments 89
5.2.8 Document Your Code: Use Descriptive Datanames 89
5.3 The “Right Way To Do It” 90
5.3.1 Use CFLOCK to Lock Server, Application, and Session Variables 90
5.3.2 Structured Query Language (SQL) in JDBC 90
5.3.3 Checking for Existence of CGI Variables 90
5.3.4 How to Break Out of Frames 91
5.3.5 How to Change the “RequestTimeout” of a Page 92
5.3.6 Dynamic HTML 93
5.3.7 How to Create an HTML Equivalent of a Graphic for TextOnly Mode 100
5.3.8 BLOBs, CLOBs and Text Datatypes, and CFQueryParam 103
5.3.9 SQL Injection, Data Validation and CFQueryParam 104
5.3.10 Cross-Browser HTML and JavaScript for Internet Sites 106
5.3.11 Suppressing Extraneous “White Space” 109
5.4 Debugging 112
5.4.1 Don’t Turn On CF Debugging Unless You Absolutely Have To 112
5.4.2 Use CFDUMP to Debug in ColdFusion MX 112
6 Application Deployment 113
7 Programming Cautions (“Gotchas” We’ve Discovered) 114
7.1 All Versions of ColdFusion 114
7.1.1 CFPROCRESULT 114
7.1.2 Calling a Java Method 114
7.1.3 Frequent Server Crashes 114
7.2 ColdFusion 4.5 115
7.2.1 The “Randomly Zeroed Out Money Fields” Problem 115
7.2.2 Sometimes You Get Errors on the Next Database Call 115
7.2.3 Sybase Error 3621 115
7.2.4 “Unknown Connect error!” 115
7.3 ColdFusion MX 6.x (and Conversion to MX in General) 116
7.3.1 JDBC: Like ODBC, Delimit Non-Numeric Literals with Single Quotes 116
7.3.2 JDBC: Parameters to Stored Procedures Must Be in Correct Order 116
7.3.3 JDBC: Designation of Input and Output Parameters Must Be Correct 116
7.3.4 JDBC: CFSQLTYPE=”CF_SQL_DATE” Is No Longer Supported 116
7.3.5 JDBC: Nullstring Passed in CFPROCPARAM Behaves Like Space 117
7.3.6 JDBC: the Syntax “= NULL” Is No Longer Allowed 117
7.3.7 JDBC: NULL Is Not a Value of a List, Either 117
7.3.8 JDBC: Stored Procedures Behave Differently Because of JDBC 117
7.3.9 StructKeyList 118
7.3.10 JSessionId 118

7.3.11 Periods in Variable Names 118
7.3.12 When Calling Java Methods, Datatype May Not Be String 118
7.3.13 The Data Validation for CFFORM Date Elements Is Incorrect 119
7.4 ColdFusion MX 7.x (and Conversion to 7.x) 120
7.4.1 CR and LF Can No Longer Appear in CFLOCATION URLs 120
7.4.2 Double Slash in a Path Is No Longer Treated the Same as One Slash 120
7.4.3 CFOUTPUT Mode Partially Propagates to Included Files 121
7.4.4 <cfset Variables.RequestTimeout = seconds> No Longer Works 121
7.4.5 Web Services: Arguments Scope Evaluated Ahead of Variables Scope 121
7.4.6 Web Services: Error Messages Have Gotten More Generic 121
7.4.7 Web Services: An Empty XML Namespace URL Crashes Axis 122
7.4.8 Web Services: Application.cfc OnRequest Messes Up Web Services 122
8 Example Files 123
8.1 Web Page User Interfaces 123
8.1.1 Example GLS Application.cfm 123
8.1.2 Example Non-GLS Application.cfm 123
8.1.3 Example Display Page (dsp_xxx.cfm) 123
8.1.4 Example Display Page in a Frame (dsp_xxx.cfm) 124
8.1.5 Example Action Page (act_xxx.cfm) 124
8.1.6 Example OnRequestEnd.cfm 124
8.2 Web Services 124
8.2.1 Example CFC File (xxx.cfc or wbs_xxx.cfc) 124
8.2.2 Example Included Function File (wbs_xxx.cfm) 124
9 Guidelines for Editing this Document 125
9.1 Headers and Footers 125
9.2 Use of Microsoft Word “Styles” Feature 125
9.3 Page Breaks 126
9.4 Default Font and Size 126

1 Introduction

This document describes a set of coding standards and recommendations for Agency ColdFusion applications. The goals of these standards are to:

- secure corporate data, web applications and servers from (1) hackers and/or (2) unintentional loss/damage of data;
- promote re-use of code;
- make code easy to read and understand; and
- ensure easier maintenance.

These standards are not intended to mandate functional organization of applications.

Request for waivers for any of these standards must:

- provide information on why the use of non-standard code is critical to the functionality of the application;
- offer reasons as to why other solutions are not viable;
- pose no security threats; and
- be requested in writing to the Office of the Chief Information Officer, Chief, Productivity Enhancement Staff.

Copies of waiver requests, approvals and/or denials should be kept with your application documentation.

1.1 Revision History

Starting with revision 3.0.4, this section will list all changes and modifications introduced to this document since its original release, in reverse chronological order. Hence, the most recent modifications and updates will be listed at the top of this section, respectively followed by less recent changes. These changes are as follows:

3.2.1 To complement ***4.1.7, When to Use Inline HTML and When to Use Frames***, added a 4-page new section ***4.1.10, How to use Inline HTML and How to Use Frames***. Added information about <cf_sbatree expandall="Yes"> to ***4.1.16.3, AppNav DHTML Tree Using <cf_sbatree> and <cf_sbatreeitem>*** (new feature). Expanded ***4.4, Standard Callbacks*** to include dsp_LookupZipToDropdown.ajax.cfm, dsp_LookupNAICSDescTxt.ajax.cfm and get_GLSSession.cfm. Started to flesh out ***4.6, Standard JavaScripts*** with more information about the JavaScripts themselves. Created a new section under Best Practices, ***5.3.6, Dynamic HTML*** and put 2 existing headings ("How to Show and Hide Page Elements Dynamically" and "How to Change Page Element Classes Dynamically") under it. Also under the new section, added Section 508 issues, the use of the DHTML navigation tree, putting data elsewhere on the page, etc. Throughout the document, replaced "old look-and-feel" screen snapshots with "new look-and-feel" equivalents.

3.2 Modified ***3.5.1, Browser Support*** (the list of required browser support for public-facing systems). Also modified entire section *Error! Reference source not found.*, ***SBA Look-and-Feel*** with "new look-and-feel" screen snapshots and explanations. Added a new section ***4.1.18, Using SBA Look-and-Feel on a Static HTML Page*** (because that's now possible).

3.1.2 Added a new section ***3.2.12, Session Conflicts in GLS***.

3.1.1 Revised and edited previous 3.1 version of this document for syntax and brevity.

3.1 Documented the standard cfinclude ***dsp_options.cfm***. Clarified policies on having multiple copies of ***Application.cfm***. Clarified the limited acceptable uses of JavaScript. Added a large new section under ***Application Model*** called ***Externally Configurable Code***, which includes the need for all GLS systems to be compatible with multiple roles. Expanded the explanation of ***bld_ServerCachedQueries*** to include 32 new queries and its new naming convention.

3.0.8 Added new section ***4.4, Standard Callbacks***. Documented the new server callbacks ***get_ArrayUserRoles*** and ***dsp_LookupZipToDropdown***. Documented the new JavaScripts ***SetFormEltValue*** and ***LookupZipToDropdown***. Documented new ***cfinclude bld_ServerCachedQueries***.

3.0.7 Clarified that ***AutoSubmit="Yes"*** is actually no longer required on ***Welcome*** pages in section ***4.1.14 Automatic Screen Resizing and TextOnly***. Documented new subdirectories of ***/library*** at the top of major chapter 4 Coding Standards, Shared Code. Added new subheading ***5.3.4 How to Break Out of Frames*** and ***5.3.10 Suppressing Extraneous "White Space"***, to ***Best Practices*** section. Augmented ***Best Practices*** section 5.3.5 with information about the new custom tag ***cf_SetRequestTimeout***. Added new major chapter 8 Example Files (for future expansion).

3.0.6 Added a new section ***5.3.6.6 How to Change Page Element Classes Dynamically.***

3.0.5 Added new section 3.5 for browser/levels and Section 508 support standards. This essentially was to elevate browser support to a standard (because it is a standard), while leaving the coding specifics in the ***Right Way to Do It*** section. Also added a new section *Error! Reference source not found.* ***How to Show and Hide Page Elements Dynamically.*** Updated 6 ***Application Deployment*** to reflect that EAR and WAR file deployment are now available, though not yet in use. Rebuilt ***Table of Contents*** section to reflect new pagination and section numbering.

3.0.4 Added this ***Revision History*** section. Added extensively more information about how to call <cf_sbalookandfeel>, its attributes, (especially the ***Show*** attribute, its relationship to button names and JavaScripts that get invoked when the user presses the buttons), the DHTML Tree custom tags (<cf_sbatree> and <cf_sbatreeitem>) for use in the ***AppNav*** region, how to write server callbacks that execute in the ***AppHidden*** frame and how to code ***MainNav*** as a frame. In the process, grouped together all <cf_sbalookandfeel> features that require knowledge of JavaScript. Added new section 4.3 on how to enable an application for logging, which is a new requirement for all applications.

2 Naming Conventions

Naming conventions are designed to quickly identify the purpose of each file, folder, directory, variable, etc. Standard naming conventions will provide ease of maintenance and updates and assist in code analysis.

2.1 File Names

In general, all file and folder/directory names visible to the public must be in lower case with no spaces, and no special characters except underscores. Exceptions – files not visible to the public, such as Application.cfm, OnRequestEnd.cfm (case sensitive under Unix), LocalMachineSettings.cfm and utility files (see below).

2.1.1 Display Files (dsp_ prefix)

Display files must begin with dsp_ and will be the only files that contain information that the user will see. These files can contain both CFML and HTML. Display files do not change anything on the server side. They only display information to the user. Queries can still be run within the display files but these queries can only obtain data, they cannot insert, update, or delete information on the server. Example: dsp_search.cfm

The initially requested file of a set of display files is also called the display page. In ColdFusion, a page can be composed of many files by the use of CFINCLUDE. Sometimes the terms page and file are used interchangeably, especially where only one file is involved.

2.1.2 Action Files (act_ prefix)

Action files must begin with act_ and do not display any information to the user. They can be used for many different purposes, but are most commonly used to change information on the server. Action files can be used to insert, update, and delete data within a database or could be used to change other data such as writing to a file. Example: act_insert_user.cfm

2.1.3 Use CFLOCATION to Pass Off from an Action File to a Display File

When they are done processing, action files pass off to display files via CFLOCATION. CFLOCATION performs a “302 Redirect”, which sends a command to the browser to treat the display file as the response of the action file. The browser responds by requesting the display file, which will be the most recently requested file. Later, if the user does a Refresh (MSIE) or Reload (Netscape), the browser will re-request the display file. It will not re-request the action file. This prevents multiple updates to the database. This is also the reason why action files are kept separate from display files.

Therefore, an action file should never pass off to a display page by using CFINCLUDE, as a Refresh or Reload would result in the re-execution of the action file, thereby causing multiple updates to the database.

2.1.4 Utility Files (various prefixes)

In order to promote code sharing, commonly performed routines may be isolated into their own utility files. Utility files are entered by CFINCLUDE, Custom Tag interface or CFINVOKE. Often, utilities perform functions that don't fall easily into act_ or dsp_ categories. In such cases, other prefixes that describe the file's function are allowed and encouraged:

bld_	builds a data structure and/or defines functions that manipulate that structure
fmt_	formats data, usually for display
get_	retrieves a value from a data structure
inc_	included file whose function is hard to characterize
qry_	performs a CFQUERY (rest of file name is generally the Query object's name. e.g. qry_GetNAICS.cfm builds query object GetNAICS)
put_	saves a value into a data structure
spc_	performs a CFSTOREDPROC ("spc" = "stored procedure call")
val_	validates data
wbs_	Web Service (cfc suffix alone does not necessarily suggest that it's a Web Service)

In general, you shouldn't make up your own 3-letter prefix. Chances are, there already exists a prefix that adequately describes your file's function. If not, the proper new prefix can be decided upon and documented in the list above.

2.1.5 Always Match Case in File Names

Certain file names receive special treatment by ColdFusion Server, namely, Application.cfm, OnRequestEnd.cfm and, starting with version 7.0, Application.cfc. On all ColdFusion Servers, these spellings must be exactly as shown. On Unix servers, where file names are case sensitive, they must be in the exact case shown as well. Because it would inhibit moving files between Unix and Windows servers to have case differences, the SBA naming standard is always to use the more restrictive case-sensitive spelling for Unix, even on Windows.

Similarly, whenever there are references to files (in CFINCLUDE, in CFLOCATION, in CFFILE, in A HREF, in IMG SRC, in SCRIPT SRC, etc), the path and file names' case must match those of the path and file exactly. For example, you may not CFINCLUDE a LocalMachineSettings.cfm file using localmachinesettings.cfm. Although localmachinesettings.cfm would work under under Windows, it will not work under Unix. Not matching case would interfere with moving the application to a Unix server. In order to assure that SBA ColdFusion applications are independent of the platform on which they run, you must always match case exactly, even on Windows servers.

Note that this restriction also applies to URLs in plain HTML. By Internet standard, the protocol (http, https) and server name parts of URLs are case-insensitive, and by convention are generally given in lower case. After the server name portion of the URL, however, the path and file parts of the URL are case sensitive on Unix, and, thus, must be treated as case sensitive in your code, even on Windows servers.

2.1.6 Backup Files

Until we have a Source Code Control System, cfm backup file name format is filename.yyyymmdd.cfm, if there is only one backup for a given date, or filename.yyyymmdd.1.cfm, filename.yyyymmdd.2.cfm, etc, if there is more than one. In either case, ***cfm*** must be used as the suffix. The yyyymmdd portion of this naming standard refers to the date of last modification, not the date you made the copy. The "Check-In/Check-Out Utility" uses this naming convention.

2.2 Variable Names

2.2.1 Database Column Names

Wherever variables correspond to database column names in the SBA's official databases (currently Sybase), the variable names must be identical to the database column name in both spelling and case. Where a rewrite is not immediately feasible, a "crosswalk document" must be written to identify which variables in ColdFusion correspond to which columns in the database. A crosswalk document is simply a tabular listing of ColdFusion names and their corresponding database names. Examples of ColdFusion variable names adhering to database column name include: Form.LoanAppNmb, URL.LoanAppNmb, Variables.LoanAppNmb, Session.LoanAppNmb, etc.

2.2.2 Datasource Names

When working on a shared development server, such as danube.sba.gov, the datasource names are decided by the system administrators. When developing your own PC and can define your own datasource, however, the current standard is to use the database name as the datasource (in the same case if Unix). For example, the DVLP1.pronet database's datasource would be pronet.

An exception is sbaref, which exists in the public tract (DVLP1, ADAPT1, WEBPROD1) and the financial tract (DVLP1, TEST1, PROD1). The datasource login for sbaref on public tract servers must be "sbaselect", while the datasource login for sbaref on financial tract servers must be "cfnonfinforms". (Since cfnonfinforms has no power in the financial databases, this is overridden at runtime with the login of the user, as resolved to a generic login by GLS.) So, sbaref presents a problem on DVLP1, the server that's on both the public and financial tracts. The solution is to define public_sbaref datasource with sbaselect for use by fastpublic, hubzone3, pronet, technet, etc, and loggedin_sbaref with cfnonfinforms for use by fast, loan, loanacct, loanapp, etc. If you need to define public_sbaref and/or loggedin_sbaref on your PC, see the database group for the passwords to the sbaselect and/or cfnonfinforms generic logins, respectively.

2.2.3 Temporary Control Variables

Because they are so frequently referenced, loop indexes and other control variables should generally be kept "short and sweet". Examples: i, j, Ctr, Idx, Pass, Temp, etc. Exception – in a shared utility file, local control variables should be made longer and identified with the utility, so as to avoid potential conflicts with control variables in the calling files. For example: get_MaxColWidth.cfm loops using MCWIdx as its loop index, not Idx, just in case the calling file is already using Idx.

2.2.4 Logic Variables

OCIO is working out a set of standardized logic variables to be used to reference paths, datasources, etc, so as to promote code sharing. The naming convention for standardized logic variables has not yet been determined.

2.2.5 XML Variables

Like OISS Project Database Names, variables containing the contents of XML elements must be identical to the element names, except during the transition between XML and database, where database names again apply. (The CFSET statements that move data between database names and XML names act as the "crosswalk".) This is particularly important where an organization outside the SBA is defining the XML element names according to their own naming conventions. Where the SBA defines the XML element names, they can be made the same as the database table and column names, so that a crosswalk isn't required.

2.2.6 Standardized Variable Names Used by Shared Code

Certain variable names are expected by some of our shared code (CFINCLUDEs, custom tags, etc). As you might expect, included files need standardized variable names more so than other types:

Variable	Used by	Contains
• Variables.db	SPC files	datasource name
• Variables.dbtype	SPC files	"Sybase11" (if CF 4.5), ignored in CFMX
• Variables.username	SPC files	login to override datasource login (*)
• Variables.password	SPC files	login to override datasource login (*)
• Variables.ErrMsg	SPC files	output variable, generally passed to dsp_errmsg
• Variables.TxnErr	SPC files	an error occurred, explained in section 4.2.6
• Variables.LogAct	SPC files	"logical action" of SPC file, explained in section 4.2.5
• Variables.cfprname	SPC files	<cfprocresult name>, explained in section 4.2.7
• Variables.cfpra	SPC files	<cfprocresult> array, explained in section 4.2.8
• Request. SpcUsingCFError	SPC files	"Yes" or "No" (whether SPC files should error) (***)
• Variables.ErrMsg	dsp_errmsg	intentionally named the same as for SPC filesf
• Variables.Commentary	dsp_errmsg	for good things that happened, as opposed to ErrMsg
• Request.SlafTextOnly	sbalookandfeel	"Yes" or "No" (whether the user wants test only) (**)
• Request.version	lastmodified	application version number to be displayed

(*) On public tract database servers (DVLP1, ADAPT1, WEBPROD1), the datasource login is used for selects, so the SPC files for stored procedures ending in "SelCSP" and "SelTSP" on public tract databases do not override the datasource login.

(**) Automatic Text Only and Screen Resizing maintains the "Slaf" ("sbalookandfeel") variables on your behalf, so if you use the automatic feature, you normally don't need to know any of the Slaf variable names. An exception is Request.SlafTextOnly, which you would use to decide whether or not to display graphics other than the ones normally displayed by <cf_sbalookandfeel>. For example, it's typical to display a graphic in a "welcome screen". SBA look-and-feel knows nothing about this graphic and will not suppress it for you. You have to code your own <cfif Request.SlafTextOnly> … <cfelse> … </cfif> logic to suppress it. If you're using the automatic feature, you can rely on Request.SlafTextOnly being defined.

(***) Some older systems, such as HUBZones, were written without error recovery. They expected to crash if an error occurred. On the public side of HUBZones, this was mitigated somewhat by coding an error page and calling <cferror> to tell CF to use that page. Until they can be rewritten to recover from errors, such systems can set Request.SpcUsingCFError to "Yes" to cause SPC file errors to crash.

3 Coding Standards, Application-Specific Code

3.1 Application Model

The SBA's programming model is to have a "thin client", business rules only in server-side validation (via stored procedures or application server), standardized look-and-feel and a goal of 50% shared code.

3.1.1 "Thin Client" and Client-Side Data Validation

Several things are implied by the term "thin client". One is that we don't use Java or plug-ins (such as Flash) where HTML will do the job. (<cf_sbatree> and <cf_sbatreeitem>, which use DHTML, are a good example.) Another is that JavaScript is used only in restricted ways, namely:

- Simple format restrictions in client-side data validation, such as disallowing alphabetic data in numeric fields, requiring EIN to be in 99-9999999 format, etc.
- Dynamic HTML, such as showing and hiding sections of a page, changing the class of a CSS-formatted display, etc.
- Providing running totals of numeric inputs as a service to the user, such as in balance sheets, income statements, calculation of debenture amounts, etc.
- SBA look-and-feel navigation.
- Alerts and confirmations of user actions that have significant consequences.

OCIO has developed standard JavaScripts for client-side data validation, which shall be used, except in the most application-specific circumstances. JavaScript will not be used for "business rules" logic, such as complex cross-edits among form elements.

3.1.2 Server-Side Data Validation

There are 2 kinds of server-side validation, so-called "presave validation", and business rules validation. Presave validation encompasses only whether or not the incoming data will "fit into its database column". As such, it is redundant to client-side data validation in JavaScript, but needs to be done on the server side, in case the user turns off JavaScript, and to prevent hacker attacks. Foreign key constraint checks, which can't be done on the client side, may also be necessary to prevent database errors at the time the database delete, insert or update is performed. Business rules validation occurs only server side via stored procedures or application server calls (Jaguar for security, for example). The goal is to perform business rules validation in only one place. Distributing business rules out to the application or the client defeats this goal and makes business rules harder to change, thereby rendering the SBA less responsive to change.

OCIO has developed standard includes and UDFs for server-side presave validation, which shall be used, except in the most application-specific circumstances.

3.1.3 Standardized Look-and-Feel

OCIO has developed standard custom tags for look-and-feel, which shall be used.

3.1.4 Our Goal Is 50% Shared Code

Shared code promotes consistency, stability and rapid application development. OCIO's goal is 50% shared code.

3.1.5 Externally Configurable Code

Initially, because there were no other mechanisms in place, configurations of SBA applications have been conducted via hard-coded configuration parameters. However, over time this expedient, but less flexible, approach is being replaced by techniques that allow configuration to be performed outside of the application, with no source code changes whatsoever. Where such mechanisms already exist, or are requested by management, they are mandatory.

3.1.5.1 Support for Multiple Roles, Privileges, Location Codes and Office Codes

The General Login System (GLS) allows a system to have multiple roles, privileges, location codes and/or office codes associated with any given user. This design allows any system under GLS to be configurable by IT Security regarding user rights and privileges Therefore:

> Systems that run under GLS must not crash or behave improperly if a user has more than one role, privilege, location code or office code.

If the behavior for one role is markedly different, it's completely acceptable to expect the user to make a choice. For example, in a time accounting application, a manager might have a role to see hours for all subordinates, but may choose to see and enter data for only his or her own hours.

The same applies to privileges, location codes and office codes. For example, if a lender may enter loan applications for 2 location codes (2 branches of a bank), you would have to require the user to choose one of those 2 location codes when entering a new loan application.

But it is NOT acceptable to pick the first role, privilege, location or office the user has (in the name of expediency), if doing so limits the user to only one role, privilege, etc, or causes the system to misbehave.

3.1.5.2 Reading External Parameter Tables

Some SBA databases (but not all) have tables for the specific purpose of external configuration. The names of such tables typically begin with "IMPrmtr" (internal management, parameter). New parameter types can be added to such tables as needed. Since database input/output is required to read such tables, it's a management decision as to what configuration parameters should be managed in this way. Check with your supervisor as to how he/she wants to handle a new configuration parameter.

Also, some systems have external tables in flat files. The Electronic Lending system called E-Tran has such files that translate XML element names to database names and provide database datatype information. Their names are SBA_ETran.vvv.columns.txt and SBA_ETran.vvv.tables.txt, where vvv is the version of the SBA_ETran XML specification. This design enables the definition of new versions of SBA_ETran without source code modifications.

Another example of external configuration in files is the popular "LocalMachineSettings" technique for varying configuration parameters on a server-by-server basis. These are normally not read as flat files, but rather cfincluded.

3.1.5.3 "New Style Logging"

"New Style Logging" is described at length in Section 4.3 of this document (as just "Logging"). It uses the highly efficient Java logging software log4j, which allows logging to be turned on or off externally.

Where systems have coded their own logging routines using <cffile action="append">, these must be converted to promote the use of the new logging routines.

3.1.5.4 Database-Driven Form Elements, with Cached Queries

Wherever codes are used to conserve space in the database, we have "definition tables", also known as "code tables" or "type tables", to translate codes to English text equivalents. When you have to display form elements (drop-down menus, checkboxes and/or radio buttons) to choose among these codes, you are required to use the database, not hard-coded HTML, to generate those form elements.

Because this imposes a database input/output penalty, small shared tables that don't change much are cached in the Server scope. This makes them available to all SBA systems, those that run under GLS and others that don't. See Section 4.5.1 for a more detailed explanation.

3.1.5.5 What Can't Be Externally Configured

There will always be a few things that cannot be externally configured.

For example, in systems that run under GLS, the GLS login process itself uses Jaguar to retrieve database login names. Therefore, the Jaguar host name and port number cannot be stored in a database parameter table. At the time you need it, you wouldn't have a database login with which to retrieve it. The same fact applies to datasource name.

Due to a syntax restriction on ColdFusion, another element that cannot be externally configured is the value clause of a <cfcase> tag. The value clause must contain a literal, not a variable. So although you may WANT to say the following…

```
<!--- Configuration Parameters: --->
<cfset Variables.ProdServers = "riogrande,wocs41">
<cfset Variables.TestServers = "rouge,yukon">

...

<cfswitch expression="#Request.SlafServerName#">
<cfcase value="#Variables.ProdServers#"> ... </cfcase>
<cfcase value="#Variables.TestServers#"> ... </cfcase>
...
</cfswitch>
```

… you will have to say the following instead:

```
<cfswitch expression="#Request.SlafServerName#">
<cfcase value="riogrande,wocs41"> ... </cfcase>
<cfcase value="rouge,yukon">      ... </cfcase>
...
</cfswitch>
```

3.2 Application.cfm

3.2.1 When and Where Required

Except for simple applications (that don't require logging in) and Web Services (which don't implicitly cfinclude Application.cfm), you must use Application.cfm or LocalMachineSettings.cfm to set global variables, for example, datasource name, timeouts for the application, and so on.

Even in simple applications, Application.cfm is a convenient place to set global configuration variables.

In an application that requires logging into GLS on the same server, you must have an Application.cfm that calls the <cfapplication> tag with name="GLS". This is what allows the user's GLS Session variables to be passed to your application. In addition, you must verify that Session.GLSAuthorized is defined and set to "Yes". If not, the <cflocation> to /gls/dsp_mustlogin.cfm or /gls/dsp_login.cfm command must be used.

In an application that requires logging into GLS on a different server, you needn't have name="GLS" in the <cfapplication> tag, because you can't share Session variables across servers. Typically, you would call the standard library routine get_ArrayUserRoles (see 4.4.4) to retrieve the user's rights from the server containing GLS.

3.2.2 When Application.cfm Is Allowed in Subdirectories

Normally, you must define one and only one Application.cfm file at the root directory of the application. There are a couple of situations, however, where a subdirectory of your application may define its own Application.cfm.

3.2.2.1 Turning Off GLS Login Requirement for Scheduled Tasks, Etc.

A Scheduled Task is normally done outside of the context of a login. (The login of the datasource is typically used instead.) So, if the root directory of a GLS system is obeying standards (kicking the user out of the directory if they're not logged in), a Scheduled Task could never work. For this reason, we typically create a subdirectory called /scheduled, as in the following example:

```
/myapp                  (root directory of GLS system)
/myapp/Application.cfm (kicks non-logged-in users out of directory)
/myapp/scheduled        (Scheduled Tasks reside here)
/myapp/scheduled/Application.cfm    (used in Scheduled Tasks)
/myapp/scheduled/act_scheduled1.cfm (not a standard name, made-up)
/myapp/scheduled/act_scheduled2.cfm (not a standard name, made-up)
```

When act_scheduled1.cfm or act_scheduled2.cfm executes, ColdFusion locates and uses Application.cfm in /myapp/scheduled. ColdFusion does not progress up the file system tree any further, so it doesn't see the Application.cfm in /myapp. The instance of Application.cfm in /myapp/scheduled contains only configuration parameters and such to be used in the context of a Scheduled Task. It does NOT kick the user out of the directory for not being logged into GLS.

You might also need to turn off GLS login in a /experiments subdirectory, for example, that exists only in development.

3.2.3 Extending Application.cfm in a Subdirectory

The reason why we don't want to have Application.cfm files in subdirectories is that they would all have to be edited/maintained in sync with each other. The objective is to avoid producing multiple copies in multiple location, but if a subdirectory contains a subsystem that needs to perform an action differently from the rest of the system, it may have an Application.cfm THAT INCLUDES THE MAIN APPLICATION.CFM and extends it. For example, this Application.cfm file extends its parent by kicking out users who have not yet logged in:

```
<cfinclude template="../Application.cfm">
<cfif NOT Variables.GLSAuthorized>
    <cflocation url="/gls/dsp_login.cfm">
</cfif>
```

3.2.4 Initialization

To minimize locking, it's advisable to move all Session variables into the Variables scope at this same time. To minimize EXCLUSIVE locking, which slows down your entire application, never code CFPARAMs or CFSETs of Session variables in a single CFLOCK. Instead, have 2 CFLOCKs, the first one being READONLY, determining whether the second one is necessary, and a second one being EXCLUSIVE to set the Session variables. For example, instead of coding:

```
<cflock scope="Session" type="Exclusive" timeout="30">
    <cfparam name="Session.xxx" default="123">
    <cfparam name="Session.yyy" default="abc">
    <!--- etc --->
    <cfset Variables.xxx                            = Session.xxx>
    <cfset Variables.yyy                            = Session.yyy>
    <!--- etc --->
</cflock>
```

(which always requires an exclusive lock), instead code:

```
<cfset Variables.SessionScopeWasInitialized         = "No">
<cflock scope="Session" type="ReadOnly" timeout="30">
    <cfif IsDefined("Session.xxx")>
        <cfset Variables.xxx                        = Session.xxx>
        <cfset Variables.yyy                        = Session.yyy>
        <!--- etc --->
        <cfset Variables.SessionScopeWasInitialized = "Yes">
    </cfif>
</cflock>
<cfif NOT Variables.SessionScopeWasInitialized>
    <cfset Variables.xxx                            = "123">
    <cfset Variables.yyy                            = "abc">
    <!--- etc --->
    <cflock scope="Session" type="Exclusive" timeout="30">
        <cfset Session.xxx                          = Variables.xxx>
        <cfset Session.yyy                          = Variables.yyy>
        <!--- etc --->
        <cfset Variables.SessionScopeWasInitialized = "Yes">
    </cflock>
</cfif>
```

3.2.5 More on Initialization – Variables Scope versus Request Scope

The difference between the Variables scope and the Request scope is that Request scope variables are directly available in the Request scope to custom tags. As a general rule, this defeats one of the benefits of custom tags, which is that their environment is largely separate from that of the caller.

In the wholesale copying of Session scope data into a scope that doesn't require locking, keep in mind that passing data to a custom tag should be a decision, not something that happens automatically. Therefore, in general, the Variables scope should be used. When only a few data items are in the Request scope, it makes it clear that their purpose is to be passed to a custom tag. In other words, in general, only shared code defines Request scope variables.

3.2.6 Set Request.Version to Identify your Application's Version Number

One variable that must be in the Request scope is Request.Version. That's because a page within a frame would call <cf_lastmodified> directly, but a page that doesn't use frames would call <cf_sbalookandfeel> in a way that would cause <cf_sbalookandfeel> to call <cf_lastmodified>. Therefore, if Version were in the Variables scope, <cf_lastmodified> wouldn't know whether to reference Caller.Version or Caller.Caller.Version (if that's even allowed). To keep the code simple, <cf_lastmodified> was programmed to look for Request.Version, so that it wouldn't matter how deeply the calls were nested.

Furthermore, setting Request.Version is a standard. In order to receive positive feedback from users, managers must know which version the users are utilizing, so the version must be continuously updated.. You are to use "variable-length dotted decimal" format (number.number, number.number.number or (rarely) number.number.number.number), which is a de facto industry standard for versions:

- Major versions (the first number) are for major changes in capabilities or how the application is used. A change of major version is intended to attract attention and caution as to its release. Examples: conversion of PRO-Net to ColdFusion, addition of Loan Servicing to ETran.
- First-level minor versions (the second number) are for feature enhancements that are significant. Examples: addition of GSAAdvantage or NAICS code searches to PRO-Net, conversion of GLS to SBA look-and-feel. There is always at least a first-level minor version. That is, the third major version is "3.0", not "3".
- Second-level minor versions (the third number) are for bug fixes, not feature enhancements. It's okay to go from "3.0" to "3.0.1".
- Third-level minor versions (the fourth number), if used, is for very minor bug fixes. It is generally NOT okay to go from "3.0" to "3.0.0.1", as that skips a level without explanation. Application changes at this level should generally attract little or no attention from users.

3.2.7 Never Use Client Scope – Requires a Waiver

You may not use Client variables without a waiver to do so. If you are granted a waiver to use Client variables, you must store them in a database, not in the registry. Since, as a general rule, you will NOT be using Client variables, do not set ClientManagement to "Yes" in the CFAPPLICATION tag.

3.2.8 No Longer Any Need to Encrypt Application.cfm

See section 3.3.4 Shared (or "Generic") Logins for information about how to retrieve logins from the database. Because we now have more secure alternatives to hard coding logins in Application.cfm, the previous SBA ColdFusion Standard requiring that Application.cfm always be encrypted is rescinded.

3.2.9 Session Control (CF 4.x and 5.x)

OMB Web standards require that Federal Web sites never write cookies to the user's hard drive, but the CFAPPLICATION tag will do this if SETCLIENTCOOKIES="Yes". Also, CF's handling of CFID and CFToken can allow a "session swap", either accidentally or maliciously, if another session's CFID and CFToken are coded on the URL. Finally, if not carefully thought out, the movement of Session variables into the Variables scope can cause unnecessary Exclusive locking. The following code example shows how to eliminate all 3 of these problems:

```
<cfapplication
   name                              = "applicationname"
   sessionmanagement                 = "Yes"
   sessiontimeout                    = #CreateTimeSpan(0,1,0,0)#
   setclientcookies                  = "No">

<cfset Variables.Initialized         = "No">
<cflock scope="Session" timeout="30" type="ReadOnly">
   <cfif IsDefined("Session.Initialized")>
      <cfif   (NOT IsDefined("Cookie.CFID")
           or (NOT IsDefined("Cookie.CFToken")
           or (Cookie.CFID    IS NOT Session.CFID)
           or (Cookie.CFToken IS NOT Session.CFToken)>
         <cflocation template="dsp_newsession.html" addtoken="No">
      </cfif>
      <cfset Variables.xxxx          = Session.xxxx>
      <cfset Variables.yyyy          = Session.yyyy>
      <!--- etc --->
      <cfset Variables.Initialized  = "Yes">
   <cfelse>
      <cfcookie name="CFID"    value= "#Session.CFID#">
      <cfcookie name="CFToken" value= "#Session.CFToken#">
   </cfif>
</cflock>

<cfif NOT Variables.Initialized>
   <cflock scope="Session" timeout="30" type="Exclusive">
   <cfset Session.xxxx              = "whatever">
   <cfset Session.yyyy              = "whatever">
   <!--- etc --->
   <cfset Session.Initialized       = "Yes">
   <cfset Variables.Initialized     = "Yes">
   </cflock>
</cfif>
```

Explanation of Session Control Code Example:

(1) The older practice of doing an Exclusive lock on the Session scope, just in case you have to initialize Session variables, is wasteful and has adverse effects on performance, because every page has to do an Exclusive lock. In this example, Variables.Initialized is used in a ReadOnly lock to determine whether the Session scope has been initialized. The result is, in all cases except the very first time, when the session is established, the Session scope locks will be ReadOnly. This is important because multiple ReadOnly locks of the same scope are not queued. (A ReadOnly lock doesn't have to wait for another ReadOnly lock to be released. It can continue on as if there were no other lock. A ReadOnly lock will only wait on an Exclusive lock.) The use of Variables.Initialized above assures that only on the very first time will there be an Exclusive Session scope lock. (This was also described in section 3.2.4 Initialization.)

(2) Only on the very first time will the CFID and CFToken cookies be sent to the browser. Because the CFCOOKIE commands don't have the EXPIRES attribute, they will be "session cookies" (held in memory, not saved to the user's hard drive), as per OMB and Federal CIO Council mandate.

(4) The SBA has pure HTML pages that establish a new session in the event that a session swap would have occurred (in this case, "dsp_newsession.html"). An example is the one used by PRO-Net. The user is nevertheless prevented from entering a CFML page with another user's session.

(5) If your application runs under GLS, you don't need to check for the existence of cookies and call CFCOOKIE yourself. GLS will already have assured that the cookies exist. In fact, you can treat the non-existence of Cookie.CFID or Cookie.CFToken as an error condition.

3.2.10 Session Control (CFMX)

Standards for the use of CFLOGIN and CFLOGINUSER have not yet been established. In the meantime, the standards for CF 4.x and 5.x, above, will still work under ColdFusion MX, if you test for Java Session Control:

```
<!--- Underscore appears in pre-MX SessionId --->
<cfif Find("_", Session.SessionId) GT 0>
   <cfset Variables.SessionControl = "ColdFusion">
<cfelse>
   <cfset Variables.SessionControl = "Java">
   <!--- Flag to use JSessionId instead of CFID and CFToken. --->
</cfif>
```

3.2.11 Session Timeout

If the maximum Session timeout value for the server you are on has been exceeded, the default timeout, not the maximum timeout, will be generated. Therefore, if your users need as much time as possible, ask the administrator(s) of the server what the maximum timeout value is for that server, and specify that as your CFAPPLICATION SESSIONTIMEOUT value. At present, all of our ColdFusion Servers are set for 1 hour for both maximum and default timeouts.

3.2.12 Session Conflicts in GLS

As more and more systems are brought under GLS for their login, roles and permission controls, the potential for one subsystem of GLS to conflict with others increases enormously. A subsystem of GLS cannot treat the entire Session scope as its own property, because all subsystems of GLS share the Session scope with all other subsystems of GLS.

3.2.12.1 Keep All Subsystem-Related Data in a Session Object

This has not been a standard in the past, but in the future, wherever any subsystem of GLS must keep 5 or more data items in the Session scope, those items must be kept in an object in the Session scope (usually a ColdFusion structure, or "struct"), not as individual variables. The name of the object will generally be the same as the System name in Security (= the subsystem of GLS). If 2 subsystems have need to share their Session variables, the name of the object must at least be descriptive of their shared function. (For example, LoanOrig and LoanServ might choose to share their Session variables in a structure called Session.ELend.)

If a subsystem of GLS has been following section 3.2.4, Initialization, and copying all Session variables into the Variables scope to minimize locking, this new standard will not present much of a change: For example, when PRO-Net's admin functions were brought under GLS, many of its Session variable names, such as "Session.AdminUser", could have conflicted with other subsystems. But because it adhered to section 3.2.4, the switchover to a structure was relatively easy. PRO-Net's System name in Security was "ProNet", so the new structure to hold PRO-Net-related Session variables was called Session.ProNet.

Then, instead of saying:

```
<cflock scope="Session" type="ReadOnly" ...>
    <cfset Variables.AdminUser = Session.AdminUser>
    <cfset Variables.Admin8a   = Session.Admin8a>
    <cfset Variables.AdminSDB  = Session.AdminSDB>
    ((etc))
</cflock>
```

it was sufficient to say:

```
<cflock scope="Session" type="ReadOnly"...>
    <cfset Variables.AdminUser = Session.ProNet.AdminUser>
    <cfset Variables.Admin8a   = Session.ProNet.Admin8a>
    <cfset Variables.AdminSDB  = Session.ProNet.AdminSDB>
    ((etc))
</cflock>
```

The remaining PRO-Net code could simply reference Variables.AdminUser (or whatever) without being concerned as to how it was being saved and restored in Application.cfm, and without having to relock the Session scope.

As your subsystem of GLS is renovated from time to time to bring it up to newer standards (such as 4.3, Logging), you must isolate your subsystem's Session data from the rest of the Session scope using this technique (if you're keeping 5 or more data items in the Session scope).

3.2.12.2 Don't Alter Session Variables Set by Other Subsystems or by GLS Itself

Simply put, if you didn't create it, then, generally speaking, you're not allowed to alter it.

That goes for Session data created by GLS itself, such as the Session.IMUserTbl data about the user or the default LocId associated with the user. So if you need to allow the user to change their default Session.LocId or Session.PrtId, you need to send the user back to GLS, so that GLS can perform that function.

And that goes for Session data created by other subsystems of GLS, such as the Session.ProNet structure described in the previous section. Generally speaking, only GLS gets to alter Session data created by GLS, only PRO-Net gets to modify Session.ProNet, only ELend gets to modify Session.ELend, etc.

Of course, "in general" means that there may be exceptions. But inasmuchas they would be deviations from the standard, those exceptions must be approved by the Director of OISS.

3.3 Security

3.3.1 Referrer Checks

CGI.HTTP_REFERER (misspelled per the HTTP standard with 3 R's) can easily be spoofed and has no meaning if the user goes to the page by e-mail hotlink, Favorites (Internet Explorer) or Bookmarks (Netscape). So the earlier standard of checking CGI.HTTP_REFERER for ".sba.gov" is rescinded.

3.3.2 Logins (Usernames and Passwords)

- End users must not be permitted to bypass the login page and access any file in a protected directory or protected resources.
- System-generated passwords must not contain 1's or 0's or lower case L's or upper case O's.
- User passwords must contain both letters and digits and include a minimum of 8 characters. (GLS password formation is controlled by GLS, so rely on errors returned by its Java methods.)
- Passwords stored in database tables must be one-way-encrypted. Use the Hash() function to one-way-encrypt passwords to a 32-character hexadecimal number. To validate a login with a one-way-encrypted password, Hash() the password the user entered and compare it to the hashed value on the database.

3.3.3 Data Validation for SQL

Numeric data entered by the user must be validated as numeric, or else parsed using Val(), CFQUERYPARAM or CFPROCPARAM. In any text field that will be referenced in PreserveSingleQuotes(), any quoted data entered by the user must have apostrophes doubled. These measures prevent hackers from submitting commands into a SQL statement. In case it's hard to read below, the Replace() functions in this example are changing single apostrophes to double apostrophes:

```
<CFSET AndClause    = "">
<CFIF Len(Form.bus_nm) GT 0>
   <CFSET Temp      = Replace(Form.bus_nm, "'", "''", "ALL")>
   <CFSET AndClause = "#AndClause# AND (bus_nm like '#Temp#%')">
</CFIF>
<CFIF Len(Form.bus_st) GT 0>
   <CFSET Temp      = Replace(Form.bus_st, "'", "''", "ALL")>
   <CFSET AndClause = "#AndClause# AND (bus_st like '#Temp#%')">
</CFIF>
<CFQUERY NAME="buscard_qry" DATASOURCE="#Variables.db#">
SELECT * from address
WHERE (bus_id = #Val(Form.bus_id)#)
#PreserveSingleQuotes(AndClause)#
</CFQUERY>
```

3.3.4 Shared (or "Generic") Logins

The datasource's login is a shared login that represents the permissions of any user who has not yet logged in. In any application that requires a login, such as those controlled by GLS, the datasource's login is a shared login that has no permissions, generally "cfnonfinforms".

In any application visible outside the firewall, all other shared logins, particularly those with update permissions, must come from behind the firewall. They specifically cannot come from a file on the Web server, even if that file is encrypted. Current mechanisms for getting a shared login through the firewall are via stored procedure (PRO-Net and related applications) or via Jaguar application server (GLS-protected applications).

If shared logins are moved to database access Web Services behind the firewall (to keep them from ever residing outside the firewall), access to those Web Services must be controlled, either by ColdFusion MX roles or by providing the user login on every call.

3.3.5 Program Descriptions (Also Known As "Comment Headers")

Programs must contain descriptive documentation of program functions and features. All updates and modifications must be recorded in the description. They must contain the name of the programmer, office or company, and version history. In addition, all accounting applications, such as ELend or Funds Control, must also have a revision history (audit trail of all released versions), so that the authorship of every line of code can be determined by the Unix utility "diff".

```
<!---
AUTHOR:           Nicolle Gurule
DATE:             02/16/2000
DESCRIPTION:      This file displays search results.
NOTES:            None.
INPUT:            Form variables from dsp_search.cfm.
OUTPUT:           Tabular display of search results.
REVISION HISTORY: 09/15/2003, DYL: Added address.
                  04/30/2003, DYL: Fixed state code.
                  02/16/2000, NG:  Original implementation.
--->
```

3.3.6 <form … method="post">

You must use the POST method versus the GET method in forms that gather data, particularly where passwords are involved, because using GET causes the password to appear in clear text on the screen in the URL.

This would normally not be visible, since the action page would normally cflocation to a display page on normal completion. But if an error occurred on the action page, the form element values become visible on the URL.

3.3.7 Cookies

The Government, in general, discourages the use of cookies. However, some applications may require cookies associated with session management. Prior to ColdFusion MX, ColdFusion used cookies CFID and CFToken. ColdFusion MX allows a new way (cookie JSessionID), but this feature must be enabled on a server-wide basis by the server's system administrator.

The following policies pertain to the use of cookies at the SBA:

- All cookies must be "session cookies", also known as "temporary cookies", or "memory cookies". Session cookies expire when the user quits their browser. This restriction implies that you cannot use the EXPIRES attribute of CFCOOKIE. See section ***3.2.9 Session Control (CF 4.x and 5.x)*** for an example of how set CFID and CFTOKEN.
- In ColdFusion applications, session cookies are preferable to maintaining a session with CFID and CFToken on the URL, because allowing CFID and CFToken on the URL can result in an accidental (or malicious) "session swap".
- Cookies and the content they collect must be described fully in SBA's Web Privacy Statement on the home page. (CFID, CFToken and JSessionID don't collect any data.)
- Cookies that collect sensitive data content must have the SECURE option enabled. (Again, CFID, CFToken and JSessionID don't collect any data.)

Federal mandates prohibit the use of "persistent cookies" - that is, cookies that are stored on the hard drive of user's PCs. The use of persistent cookies requires the personal approval of the Administrator of the SBA.

Requests for a cookie waiver must provide the following detail:

- The name of the proposed cookie
- Full explanation of why the cookie is critical to the application
- Reasons why alternatives are not viable
- Detail on the cookie content
- Description of how the information is used
- The type of cookie (persistent or session)

3.3.8 File Upload Restrictions

CFFILE ACTION="UPLOAD" must ACCEPT only text MIME types, and you must first compare CGI.CONTENT_LENGTH to the maximum file upload size currently allowed by the security group. Furthermore, no file uploaded by the public is allowed to remain on any Web server's hard drive. You must immediately read it (CFFILE ACTION="READ") and delete it (CFFILE ACTION="DELETE"). Failure to follow these rules can allow (1) uploading a virus or (2) a "Denial of Service" attack wherein a user repeatedly uploads a huge file and eventually fills the Web server's hard drive.

3.4 Database

3.4.1 Structured Query Language (SQL) versus Stored Procedure Calls

Wherever a stored procedure exists to do so, all deletes, inserts, selects and updates must be via CFSTOREDPROC calls, rather than your own SQL. (See 4.1.17 Stored Procedure Call Files.)

SQL may be used where no corresponding select stored procedure exists, as in the case of dynamically generated SQL. Certain SQL select techniques cannot effectively be done in stored procedures, such as the use of IN or NOT IN with a list of values. Similarly, where the user can choose from many possible search criteria, stored procedures cannot effectively handle them all due to a mathematical problem known as “combinatorial explosion”. In such circumstances, SQL is the only solution that works.

Also, where a stored procedure does not yet exist, but database permissions allow deletes, inserts, selects and/or updates without going through a stored procedure, you may have to code SQL to test your code before the stored procedure is written. This may have the added benefit of detecting special problems and refining the requirements specification for the stored procedure itself. However, whenever this is done, the application must be rewritten to use the stored procedure, once it becomes available.

Last but not least, there is a 255 character limit on parameters to a stored procedure in Sybase. (Microsoft SQL Server does not have this limit, but Sybase does.) So if a Sybase column to be inserted, selected or updated is of datatype “text” or a Binary Large Object (BLOB), it’s understood that it cannot be passed as a parameter to a stored procedure and must be inserted, selected or updated via SQL.

3.4.2 Use CFTRANSACTION, not CFLOCK, to Lock Database Changes

Limit concurrent updates to the database using CFTRANSACTION, not CFLOCK. CFTRANSACTION will affect all accesses to the data being updated, not just those occurring in your ColdFusion code. Example code:

```
<cftransaction action="BEGIN" isolation="Serializable">
   <!--- database update with spc file --->
   <!--- database update with spc file --->
   <!--- database update with spc file --->
   <cfif Variables.TxnErr>
      <cftransaction action="ROLLBACK" />
   <cfelse>
      <cftransaction action="COMMIT" />
   </cfif>
</cftransaction>
```

Note that the tags with ROLLBACK and COMMIT contain a slash just before end-of-tag. This is a convention from XML that means “this tag, which normally has a closing tag, doesn’t have a closing tag in this case”. ColdFusion uses this convention to nest ROLLBACK and COMMIT actions inside a <cftransaction> … </cftransaction> block.

3.5 Miscellaneous

3.5.1 Browser Support (HTML, CSS and JavaScript)

In "Intranet" applications (those that will be seen only by SBA employees and others behind the SBA firewall, it is acceptable to support only the current version of Microsoft Internet Explorer ("MSIE") for Windows.

But in "Internet" applications (those that will be seen by the public), you must support up to 2 levels back of the following browsers:

1. MSIE for Windows (2 levels back = 7.0, 6.0 and 5.5)
2. Netscape for Windows (2 levels back = 8.1, 7.2 and 7.1)
3. Firefox for Windows (2 levels back = 2.0, 1.5.0.8 and 1.5)
4. Firefox for Macintosh (2 levels back = 2.0, 1.5.0.8 and 1.5)
5. Safari for Macintosh (2 levels back = 2.0.4, 2.0 and 1.3)

Note that MSIE for Macintosh is no longer on this list. Its versions were capped more than 3 years ago, and is no longer on the Microsoft site for download. Also, Firefox has been added to the list. And there have been recent new releases of all of these browsers which are now on this list.

Also, although not required, it is advisable to be compatible with strictly-standards-compliant browsers, so that your code will continue to run correctly as all browsers become more standards-compliant:

6. Opera for Windows (current version)

NOTE: *Although you're not required to support anything more than the current MSIE for Windows in intranet applications, it is generally recommended to code for cross-browser compatibility anyway, as this allows the SBA to change its standard browser without major code conversions, and prepares you for coding for the wider Internet user community.*

Since HTML, CSS and JavaScript versions vary according to browser versions, supporting the browser implies writing HTML, CSS and JavaScript that behave correctly in those browser versions. See section 5.3.10, below, for specific help and guidance on writing cross-browser compatible code.

3.5.2 Section 508 Support

Section 508 support is the subject of other SBA standards documents, so those standards will not be discussed here. It's sufficient to say that your ColdFusion application must obey those standards as well.

4 Coding Standards, Shared Code

Where Shared Code exists to do what you need to do, you must use it. On all SBA ColdFusion Servers, shared code always resides in the following directories mostly at the Web server's document root level:

/cfincludes/datasourcename	Stored Procedure Call files (see 4.2)
/library/callbacks	Callbacks to server in AppHidden frame (see 4.1.16.4)
/library/cfincludes	CFML code invoked by <cfinclude>
/library/classes	Client-side Java (applets), contains /source and /examples
/library/css	Cascading Style Sheets
/library/customtags	CFML custom tags (example, SBA look-and-feel)
/library/html	Sharable HTML (blank.html, etc)
/library/images	Images (gif, jpeg, jpg, png, etc)
/library/javascripts	client-side data validation, form control, etc
/library/pdf	Sharable PDF files
/library/swf	Sharable Flash files
/library/udf	User Defined Functions
/library/video	Movies (typically very large, mov, wmv, etc)
/library/xml	Sharable XML, notably XSD components
/opt/coldfusion[mx[7]]/Java/classes	Java custom tags (file system address, not under doc root)

To add new features to existing Shared Code, there's also a "prelibrary" directory with the same subdirectories. Developing new library features out of prelibrary avoids the problems of impacting other developers, who are generally using library, not prelibrary, as their top level directory for Shared Code. If you wish to create some new code to be shared across all applications, you will have to ask to be added to the prelibrary Unix group, which allows you to edit files in prelibrary and its subdirectories.

Generally speaking, you should adhere to a similar pattern in your application when you have application-specific code that you want to share within your app:

/myapp/cfincludes/datasourcename	Application-specific (edited) SPCs
/myapp/cfincludes	Files included from multiple subdirectories of myapp
/myapp/customtags	(must be invoked by cfmodule)
/myapp/java	Server-side Java (servlets and CFXs)
/myapp/javascripts	Scripts not related to SBA Look-and-Feel
/myapp/javascripts/sbalookandfeel	Scripts invoked by "MainNav" buttons
/myapp/udf	(etc)
/myapp/xml	(etc)

Adhering to this convention allows developers on one application to be assigned to another application and know where to find things. For now, this naming convention is recommended and not mandatory. However, application-specific Shared Code directories must adhere to this naming convention.

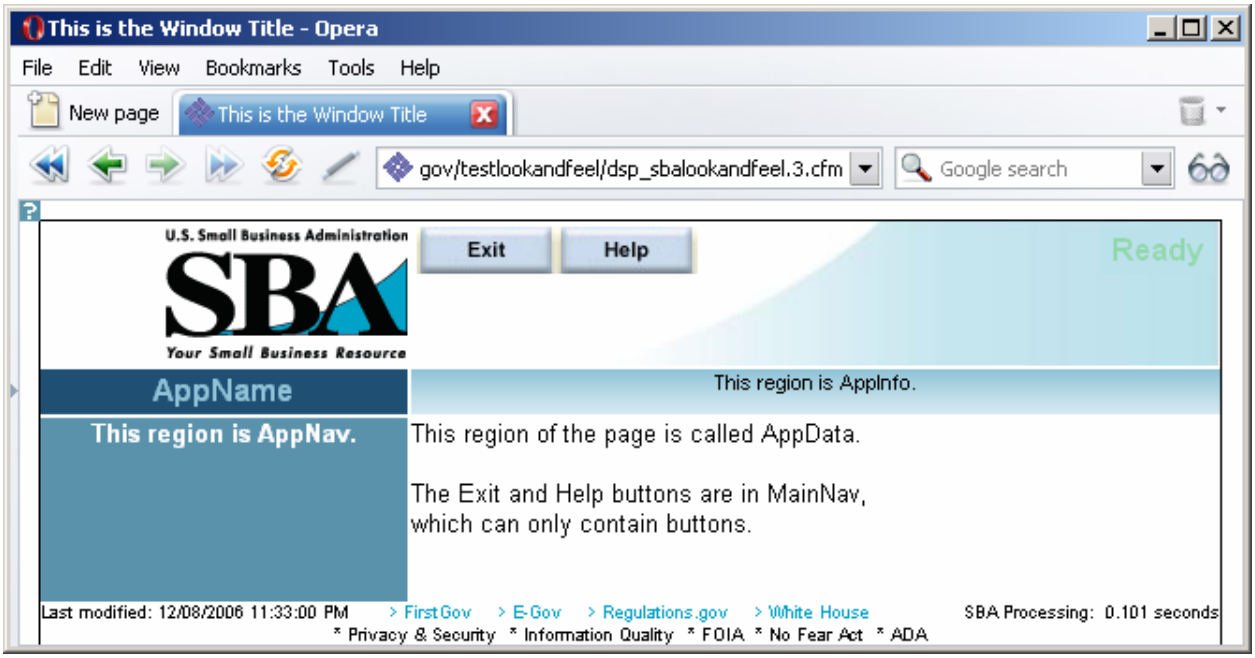
Only application-specific Shared Code may be used in application specific Shared Code directories. You may NOT copy system-wide Shared Code from /library into your own app's directories and use them from there. Examples of truly application-specific code might be JavaScripts that reference specific form element names (EditBS in ELend) or codes unique to the application's database (PronetMincFormat). If code could be shared across multiple applications, it belongs in /library, and you must use the /library version of it, not your own local copy.

4.1 SBA Look-and-Feel

To present a consistent look-and-feel across all applications, OCIO/OISS has developed 2 ColdFusion custom tags, called cf_sbalookandfeel and cf_mainnav. If your application has no need of frames, you probably won't ever call cf_mainnav.

Perhaps the hardest part of learning how to use the SBA look-and-feel custom tags is the wide range of features, which quickly induces "information overload". This portion of the SBA ColdFusion Standards document is meant to introduce you to the features gradually, in a top-down approach, so that you'll be better able to decide which features to use and which to ignore.

4.1.1 Screen Snapshot of SBA Look-and-Feel, Showing Page Regions



Opera is being used in this snapshot to show that SBA look-and-feel is cross-browser compatible. Regardless of browser, the page looks pretty much like this. SBA look-and-feel has been tested with Microsoft Internet Explorer for PC and Mac, Netscape for PC and Mac, Firefox for PC and Mac, Opera for PC and Safari for Mac. The reason why we're able to support such a wide range of browsers is that cf_sbalookandfeel adheres to the Cascading Stylesheets – Positioning (CSS-P) standard.

As you can see from the included text, the page is divided up into "regions". Not all of the regions are required. Four of them can be inline (markup residing in the current file) or frames (markup residing in separate files, usually for code sharing). The choice is up to you. The appearance will be the same.

Note: <u>All regions have 0 pixels of padding.</u> This was necessary for consistent cross-browser positioning. A side benefit is that it gives you edge-to-edge control over what appears in each region. If you don't want 0 pixels (if you would prefer, say, 2 pixels of padding), you can simply wrap everything in ((region's markup)) , or use a <table> with cellpadding and/or cellspacing. It's much easier to add the padding you want than it is to get rid of padding you don't want.

4.1.2 Regions of the Page and What They're Called

The regions of the page (listing them top-down and then left to right) are as follows:

- **WindowTitle** is at the very top of the window. In tabbed browsers, it also appears in the tab.
- The **SBA Look-and-Feel Menu** control is a tiny medium blue square in the upper left corner.
- The **10-Pixel White Margin** is the outer white region which, when clicked, maximizes the AppData region or reverts it back to original size. If you produce end user documentation for your system, be sure to tell your end users about this maximize/minimize feature. It's not terribly obvious or well-known, but it greatly enhances ease-of-use.
- **SBALogo** is a hotlink to the SBA Home Page (http://www.sba.gov).
- **MainNav** is situated to the right of SBALogo. It can be a frame imaged by a separate ColdFusion page. It contains an optional "Ready Light" and the main navigation buttons, as follows:
  - The top row of buttons are standard, and appear in a standard order, regardless of the order you specify them. These buttons come from /library/images/sbalookandfeel.
  - The bottom row of buttons are application-specific, appear in the order you specify and come from /library/images/applookandfeel. (This gives you a convenient place to look for buttons that already say what you need one to say.)
  - Both rows invoke JavaScripts that are unique to your application. Therefore, although the Admin button's appearance and placement are standard, pressing it performs a different action in each application.
- **AppName** is under SBALogo. The name of your application is put there. One line only, please. If you need to display more, consider using a portion of AppInfo or a title at the top of AppNav.
- **AppInfo** is to the right of AppName. It can be inline HTML or a frame imaged by a separate ColdFusion page. It's used to identify "what you're looking at" (what loan, what company, etc).
- **AppNav** is under AppName. It can be inline HTML or a frame imaged by a separate ColdFusion page. It's where you put situational navigational elements, such as hotlinks or a navigation tree. For example, if the user presses the Reports button in MainNav, it might take the user to a /reports subdirectory of your app, and in that subdirectory, AppNav might contain a list of reports that the user has permissions to display. .
- **AppData** is to the right of AppNav. It can be inline HTML or a frame imaged by a separate ColdFusion page. It usually contains data entry forms, search results, welcome screens, etc. It's the largest and most important region of the screen, where users conduct most of their work.
- **BotMost** is the region where the standard, end-of-page hotlinks appear.
- **AppHidden** is invisible. It's always a frame containing a separate page. If you don't specify which page in the SBA look-and-feel call, it will initially be /library/html/blank.html. It's used for "server callbacks" (see 4.1.16.4).

4.1.3 Which Regions are Optional

If AppInfo isn't given, MainNav expands downward to fill the space normally filled by AppInfo. This feature could someday be used to make room for more MainNav buttons, but so far that hasn't been necessary.

If AppNav isn't given, AppData expands leftward to fill the space normally filled by AppNav. Typically, you would do this to display a big search form or wide tabular data, such as the results of that search. Although it could be done, you would NOT do this to display data for printing, such as all data associated with a loan application, or a report. For printable reports, you would typically leave SBA look-and-feel entirely and display the report against a plain white background, because SBA look-and-feel elements would be inappropriate in the context of a printed report.

4.1.4 How to Call the SBA Look-and-Feel Custom Tag

You would generally never use <cfmodule> unless you were developing a new version of the custom tag itself. Generally speaking, you should use <cf_customtagname> syntax instead, as follows:

```
<cf_sbalookandfeel
    attribute=value
    attribute=value
    ...>
    ((possible AppData HTML here, appended to AppDataInline))
</cf_sbalookandfeel>
```

Attributes (defaults of multiple-choice features shown in **bold**):

```
AppDataInline – HTML of AppData (see below)
AppDataURL    – URL of AppData if you want it to be a frame
AppHiddenURL  – URL of AppHidden frame (not usually used)
AppInfoInline – HTML of AppInfo if AppInfoURL not given
AppInfoURL    - URL  of AppInfo if you want it to be a frame
AppName       - Name to appear in AppName region of page
AppNameHeight – Height, in pixels, of AppName and AppInfo regions
AppNavInline  - HTML of AppNav if AppNavURL not given
AppNavURL     - URL  of AppNav if you want it to be a frame
AutoResize    - "Yes" or "No" – whether to turn on feature
Configs       - Used in development of new versions of custom tag *
Debug         - Used in development of new versions of custom tag *
JSInline      - HTML of JavaScripts, usable by all inline regions
LibURL        - Used in development of new versions of custom tag *
MainNavHiddens- Used to pass CF data to MainNav JavaScripts *
MainNavJSURL  - Directory of MainNav JavaScripts *
MainNavURL    - URL of MainNav if you want it to be a frame
ReadyLight    - "Yes" or "No" – whether to turn on feature *
Show          - List of buttons to appear in MainNav *
WindowTitle   - Name to appear in title bar of window
TextOnly      - Old, "Yes" or "No", nowadays use AutoResize *

* = passed to <cf_mainnav> if you don't give MainNavURL
```

There are 2 ways to give inline HTML for the AppData region. One way is to put it between <cf_sbalookandfeel> and </cf_sbalookandfeel>. The other way is to pass it as the AppDataInline attribute. Either way is fine, and you can do both. If you do both, the contents of AppDataInline will be put into AppData first, followed by whatever's between <cf_sbalookandfeel> and </cf_sbalookandfeel>.

The handling of frames versus inline HTML is not nearly so flexible, because the custom tag won't know which to do. For example, if you give both AppInfoInline and AppInfoURL, <cf_sbalookandfeel> will abort the call, saying that you must use one or the other, but not both.

4.1.5 Controlling the MainNav Buttons with the Show Attribute

In order to make SBA look-and-feel extensible without reprogramming every time a new button is introduced, the Show list of buttons is the list of ALT attributes for the button images. From this list of ALT attributes, it generates the button graphic image file names, JavaScript names, JavaScript file names, etc. Because of restrictions of file names and JavaScript expressions, their names must have spaces and hyphens stripped out first. It's easier to show everything that happens by providing an example such as the following: Suppose the ALT attribute of your button is "Go To E-Gov":

ALT attribute:	Go To E-Gov	
JavaScript function name:	DoGoToEGov	(spaces and hyphens removed)
JavaScript file name:	DoGoToEGov.js	(js suffix added)
JavaScript name of graphic:	document.gotoegov	(lower case to match JPEGs)
"Lowlighted" (mouse not over) graphic:	/libray/images/applookandfeel/gotoegov_lo.jpg	
"Highlighted" (mouse over) graphic:	/libray/images/applookandfeel/gotoegov_hi.jpg	

It's not SBA look-and-feel's responsibility to make sure that the JavaScript file or /applookandfeel images exist. Its responsibility is simply to generate HTML based on your inputs. It's your responsibility to make sure that the files exist. But you don't have to do it all yourself. OISS has specialists in graphics and JavaScript to create them for you. All you have to do is request the creation of a new button. Whoever creates the button graphics for you will also put them into the /applookandfeel directory for you.

You can also shortcut the process by searching /library/images/applookandfeel for an already existing button that may be close enough to what you want it to say. Hence, if there were already files there called "egov_lo.gif" and "egov_hi.gif", and you felt that these file names were descriptive enough, you could go ahead and use them, even though they were not specifically created for your application. All you would need to do in that case is create (or request creation of) a DoEGov.js file that defines a JavaScript function called DoEGov.

Assume this file didn't exist, and your button has ALT text "Go To E-Gov". Suppose further that you have already built "Yes"/"No" variables with User in the name, indicating the privileges of the current user. You would normally build the Show attribute according to those privileges, as in the following example ("Variables." omitted to keep the lines from wrapping in this example):

```
<cfset Show      = "Exit,Help"><!--- All users --->
<cfif AdminUser>
    <cfset Show  = ListAppend(Show, "Admin")>
</cfif>
<cfif PublicUser>
    <cfset Show  = ListAppend(Show, "Go To E-Gov")>
</cfif>
<cfset Show      = ListAppend(Show, "Reports")><!--- All users --->
...
<cf_sbalookandfeel
    ...
    Show         = "#Show#">
```

The actual contents of the Show attribute might be "Exit,Help,Go To E-Gov,Reports". Although it appears to be wrong to have spaces in the list, the contents of the ALT attributes of the graphics is what's expected. (If the user is in TextOnly mode, it will instead be the text hotlink. Since the user had to request TextOnly mode, that's just what the user wanted, only the text that says what the hotlink does.)

4.1.6 How to Specify Inline HTML versus Frames

To specify inline HTML:

- **MainNav** – Don't give the MainNavURL attribute to cf_sbalookandfeel. cf_sbalookandfeel will call cf_mainnav for you, causing MainNav to be HTML in a cell of a <table> on the same page.
- **AppInfo** – Build the HTML into a variable (by convention and force-of-habit, it's typically called Variables.AppInfo). Don't give the AppInfoURL attribute to cf_sbalookandfeel, but instead give AppInfoInline and pass the contents of that variable (for example, AppInfoInline="#Variables.AppInfo#").
- **AppNav** – Build the HTML into a variable (by convention and force-of-habit, it's typically called Variables.AppNav). Don't give the AppNavURL attribute to cf_sbalookandfeel, but instead give AppNavInline and pass the contents of that variable (for example, AppNavInline="#Variables.AppNav#").
- **AppData** – Don't give the AppDataURL attribute to cf_sbalookandfeel, but instead give the HTML between <cf_sbalookandfeel> and </cf_sbalookandfeel>, or in AppDataInline.

Recommendation for inline HTML: A very convenient way to build inline HTML in a variable is using <cfsavecontent>. It's very much like outputting HTML to a Web page, and much easier than using string concatenation. For example, if you need to have JavaScripts available to any of the above regions (usually AppData), you would use the JSInline attribute. The following is an example of how to build it:

```
<cfsavecontent variable="Variables.JSInline"><cfoutput>
<script src="/library/javascripts/EditMask.js"></script>
<script src="/library/javascripts/EditDate.js"></script>
<script>
<!—
function   DoThisOnSubmit(pForm)
{
...
}
// -->
</script>
</cfoutput></cfsavecontent>
<cf_sbalookandfeel JSInline="#Variables.JSInline#" ... >
```

To specify frames: To have a region imaged by a separate ColdFusion page in a frame, pass the following attributes to cf_sbalookandfeel with the URL of the separate ColdFusion:

- **MainNavURL** – Example MainNavURL="dsp_mainnav.cfm". (In the page pointed to by MainNavURL, you would call <cf_mainnav>.)
- **AppInfoURL** – Overrides AppInfoInline. Example AppInfoURL="dsp_appinfo.cfm".
- **AppNavURL** – Overrides AppNavInline. Example AppNavURL="dsp_navtree.cfm".
- **AppDataURL** – Overrides HTML between <cf_sbalookandfeel> and </cf_sbalookandfeel>. (If there's also HTML between <cf_sbalookandfeel> and </cf_sbalookandfeel>, it will mess up the display of the page. So don't give both AppDataURL and HTML between <cf_sbalookandfeel> and </cf_sbalookandfeel>.) Example AppDataURL="dsp_enterapp.cfm".
- **AppHiddenURL** – If given, preloads the invisible frame in which you can do server callbacks. If not given, "/library/html/blank.html" will be used. (Note that, no matter what, the AppHidden frame will always exist, so that it can be used later by library routines when/if needed).

4.1.7 When to Use Inline HTML and When to Use Frames

As mentioned above, MainNav, AppInfo, AppNav and AppData can be either inline HTML or frames imaged by separate ColdFusion pages. (That's why we refer to them generically as "regions".)

The decision of whether to use inline HTML or frames to image a region is normally based on page load times and trying to get the pages to load as fast as possible. If a region is shared across many pages and doesn't change much, using a frame would be the suggested alternative, so that other segments of the page that do not change much can change without having to reload the frame. If, however, virtually all regions of the page change, inline HTML would be the recommended alternative.

Scrolling convenience is no longer a major consideration. With the original version of cf_sbalookandfeel, if both AppNav and AppData were potentially long (and hence, liable to result in scrolling), you might have been forced make them both frames so that they could scroll independently of one another. But with the new CSS-P version of cf_sbalookandfeel, inline regions can scroll independently too, just like frames. It also has the added advantage of guaranteeing that the BotMost hotlinks will remain on the screen.

Frames offer the greatest potential to significantly optimize page load times, but at the cost of making the application more complex. You might very likely have to code JavaScript commands for one frame to tell another frame to reload itself. If the other frame is a server callback in AppHidden, it will likely pass data back to the current frame using JavaScript when it's done. And you have to learn how to target other frames in hotlinks, so HTML also gets a little more complex.

4.1.8 What Happens When MainNav Is NOT a Frame

When the MainNavURL attribute is not specified, cf_sbalookandfeel calls cf_mainnav for you and passes all attributes that are allowed by both tags. Currently, that list is ActionURL, Configs, Debug, LibURL, MainNavHiddens, MainNavJSURL, ReadyLight, Show and TextOnly.

In addition, it passes 2 attributes of cf_mainnav which are NOT attributes of cf_sbalookandfeel: InFrame (always "No" in this situation) and Width. Simply put, because cf_mainnav's output will be inline HTML, cf_mainnav needs to to be told that it's not in a frame and must constrain the width of the HTML.

Therefore, even if you're not coding a separate page to display a MainNav frame, you still need to know the cf_mainnav attribute names and what they do, because you have to give them in the call to cf_sbalookandfeel. (The most important ones are ReadyLight and Show, by the way.)

4.1.9 What Happens When MainNav IS a Frame

When the MainNavURL attribute is specified, it contains the URL of another page that will fill the MainNav frame and will call cf_mainnav. It's very important that certain attributes of both custom tags be the same.

In particular, if you say <cf_sbalookandfeel ReadyLight="Yes" MainNavURL="dsp_mainnav.cfm">, and dsp_mainnav.cfm says <cf_mainnav ReadyLight="No">, a JavaScript error will result. The reason is, the frames document will generate JavaScript to control the ReadyLight, but the MainNav frame won't define it. When the JavaScript executes, it generates an error when attempting to reference an undefined object.

4.1.10 HOW to Use Inline HTML and HOW to Use Frames

Simply put:

- When a region is INLINE, you must NEVER provide <DOCTYPE>, <html>, <head>, </head>, <body>, <cf_lastmodified>, </body> or </html>.
- When a region is in a FRAME, you must ALWAYS provide <DOCTYPE>, <html>, <head>, </head>, <body>, </body> and </html>. In addition, if the region is AppData, you must also call <cf_lastmodified> just before </body>.

4.1.10.1 Special Problem – Executing JavaScript onLoad with an Inline Region

When a region is in a frame, you can just code <body onLoad=" … "> and do anything you want when the frame is fully loaded. But what if the region is inline? The answer is to put all of your onLoad code into a function called DoSomethingDifferentOnLoad (with exactly that spelling and capitalization), and include it in the JSInline attribute of the <cf_sbalookandfeel> call. In addition, your DoSomethingDifferentOnLoad function must do everything that's done in the "else" condition of the <cf_sbalookandfeel>-generated body tag's onLoad. Here's why:

When <cf_sbalookandfeel> generates the body tag, the onLoad JavaScript checks for the existence of DoSomethingDifferentOnLoad. If it sees a function by that exact name, it calls the function INSTEAD OF its usual initializations, not in addition to them. Therefore, it's the responsibility of your DoSomethingDifferentOnLoad function to do the exact same initializations that would have been done by the generated body tag, which is the code in the "else" condition.

Another way to do it would have been for the generated body tag's onLoad to do the initializations, then call DoSomethingDifferentOnLoad if it's defined. That would have been less work for those who want to use the DoSomethingDifferentOnLoad mechanism. But, in the process, it would have also denied you the ability to execute your own code AHEAD OF the initialization code, in case you needed to do that for some reason. (It's highly recommended that you perform the SBA-look-and-feel initializations FIRST, so that they will be done even if there are errors in your own JavaScript.)

Example 1 (AppData OnLoad when AppData and MainNav Are Both Inline):

Because MainNav is inline, <cf_sbalookandfeel> generates the body tag as follows:

```
<body onload="
if  (top.DoSomethingDifferentOnLoad)
    top.DoSomethingDifferentOnLoad();
else
    {
    MainNavDoThisOnLoad();
    SlafDoThisOnLoad();
    }
"
onresize="SlafDoThisOnResize();">
```

Because MainNav is inline, <cf_sbalookandfeel> must initialize it by calling MainNavDoThisOnLoad. Because you're OVERRIDING <cf_sbalookandfeel>'s initializations, you must call it too.

Therefore, if you want to put the focus into your form called FormRight, form element BusCntctFirstNm, you would do the following:

```
<cfsavecontent variable="Variables.JSInline">
    <cfoutput>
<script>
function DoSomethingDifferentOnLoad ()
{
MainNavDoThisOnLoad();
SlafDoThisOnLoad();
document.FormRight.BusCntctFirstNm.focus();
}
</script>
</cfoutput>
```

Then later:

```
<cf_sbalookandfeel
    ...
    JSInline             = "#Variables.JSInline#"
    ...>
```

Example 2 (AppData OnLoad when AppData Is Inline But MainNav Is In a Frame):

Because MainNav is in a frame, <cf_sbalookandfeel> genertates the body tag differently, as follows:

```
<body onload="
If  (top.DoSomethingDifferentOnLoad)
    top.DoSomethingDifferentOnLoad();
else
    SlafDoThisOnLoad();
"
onresize="SlafDoThisOnResize();">
```

Note that the else clause doesn't bother to call MainNavDoThisOnLoad, because that's the responsibility of the MainNav frame. So you shouldn't call MainNavDoThisOnLoad either. In fact, you must NOT call MainNavDoThisOnLoad in your DoSomethingDifferentOnLoad function, because it won't be defined:

```
<cfsavecontent variable="Variables.JSInline">
    <cfoutput>
<script>
function DoSomethingDifferentOnLoad ()
{
SlafDoThisOnLoad();
document.FormRight.BusCntctFirstNm.focus();
}
</script>
</cfoutput>
```

The call to <cf_sbalookandfeel> is the same as in Example 1.

Example 3 (Appending to JSInline):

Suppose you have a standard header file that you cfinclude at the start of all your pages. So Variables.JSInline is already defined with stuff you need. In a situation like that, how would you append your DoSomethingDifferentOnLoad function into the Variables.JSInline that was defined in the header file? It's actually quite simple:

```
<cfsavecontent variable="Variables.JSInline">
    <cfoutput>#Variables.JSInline# <!--- Defined in dsp_header --->
<script>
function DoSomethingDifferentOnLoad ()
{
((whatever))
}
</script>
</cfoutput>
```

This is exactly analogous to appending something with <cfset>, except that you're doing it within a <cfsavecontent>. This <cfsavecontent> technique has been tested, and it works just fine.

(Note the courtesy to other developers in commenting where the existing JSInline data comes from.)

Example 4 (JSInline Already Contains a DoSomethingDifferentOnLoad Function):

If you have a special case file that needs to do something different from your header file's DoSomethingDifferentOnLoad, it's best to modify the header file's DoSomethingDifferentOnLoad. The crude way to do it (not recommended) is to check the current page name:

In dsp_header.cfm:

```
function DoSomethingDifferentOnLoad()
{
((whatever would have been done by cf_lookandfeel's body "else"))
((whatever is normally done for all pages that use dsp_header.cfm))
...
var sPathArray = document.location.pathname.split("/");
var sFileName  = sPathArray[sPathArray.length – 1];
if  (sFileName == "dsp_contacts.cfm")
    document.FormRight.BusCntctFirstNm.focus();
}
```

The split() method is defined for all JavaScript strings since JavaScript 1.1, over 10 years ago. Its ColdFusion equivalent is ListToArray, so this is essentially like ListToArray(CGI.Script_Name). The main difference is that JavaScript arrays are 0-based, so you have to subtract 1 to get the last element.

This technique will work, but it's NOT RECOMMENDED, because it isn't extensible. In other words, if you wanted to put the focus on different form elements on different pages, you would have to keep modifying and modifying and modifying dsp_header's DoSomethingDifferentOnLoad function to test for all the different page names. It would also hinder sharing that header file in a different directory, because the DoSomethingDifferentOnLoad function would be essentially tied to the directory it was created for.

The more elegant and extensible way to do it is to check for the presence of a special function specific to your application:

In dsp_header.cfm:

```
function DoSomethingDifferentOnLoad()
{
((whatever would have been done by cf_lookandfeel's body "else"))
((whatever is normally done for all pages that use dsp_header.cfm))
if  (top.DoSomethingExtraOnLoad)
    top.DoSomethingExtraOnLoad();
}
```

In dsp_contacts.cfm:

```
<cfsavecontent variable="Variables.JSInline">
    <cfoutput>#Variables.JSInline# <!--- Defined in dsp_header --->
<script>
function DoSomethingExtraOnLoad ()
{
document.FormRight.BusCntctFirstNm.focus();
}
</script>
</cfoutput>
```

Do you see the improvement? It's extensible to other pages. Now any page that wants to do extend dsp_header's DoSomethingDifferentOnLoad function simply has to define its own "DoSomethingExtraOnLoad" function. And dsp_header.cfm's DoSomethingDifferentOnLoad function doesn't have to be reprogrammed to recognize several file names.

Note that this new function name has no meaning to SBA look-and-feel. It is known only to your own application's DoSomethingDifferentOnLoad function.

4.1.10.2 Special Problem – Referencing a Frame

If you're used to referencing the Document Object Model of a page without frames, it's a little extra work to reference the same sorts of things when frames are involved, but not too much extra. Frames are window objects in JavaScript. Therefore, you can reference all of the properties of the window object, such as document, location, etc, and all of a window's methods, such as reload(), writeln(), etc, using a frame reference.

The topmost window/frame reference is "top", so the SBA look-and-feel regions (IF THEY ARE FRAMES) can be referenced as top.AppData, top.AppHidden, top.AppInfo, top.AppNav and/or top.MainNav. Of these, only AppHidden is always a frame.

Other than the need to reference the frame to get to an object in a different frame, you can do pretty much everything you could do if the object were in the same frame. The major exception is populating options into a dropdown menu. Microsoft Internet Explorer 5.0 and higher will not let you do that. Instead, you have to define a function in the same frame as the dropdown, then call that function from other frames. (An example of how to do this is in the non-AJAX version of LookupZipToDropdown.)

4.1.11 What CSS Class Names to Use

The call to cf_sbalookandfeel will link in the cascading stylesheet file /library/css/sba.css, which defines the standard classes and associated colors for SBA look-and-feel. It will also specify the CSS class names to be used in each of the regions of the page. But when those regions are frames (imaged by other pages), you'll need to link in /library/css/sba.css yourself on those pages, and use the following class names:

MainNav background	– class="headernav"
AppInfo background	– class="inthead"
AppInfo labels	– class="infolabel"
AppInfo data	– class="infodata"
AppNav background	– class="leftnav" (or leftnavCopy)
AppNav subheader of AppName	– class="leftnavtitle"
AppNav hotlinks	– class="menuitem"
AppNav color of highlighted link	– class="menuitem_hi" (colors text like highlighted hotlink)
AppData	– class="normal" or no attributes (<body>)

Normally, you'll get these classes and colors automatically in inline regions of the page. Occasionally, while defining a <table> for example, you may accidentally override the normal SBA look-and-feel defaults. If that happens, you can restore them by using the class names or colors above.

Also, the defaults provided by cf_sbalookandfeel are just the backgrounds. Hotlinks, headers, data, etc, are standardized class names, but they aren't applied automatically. Rather, they must be applied manually to the specific items you want to colorize. Examples:

AppInfo:

```
<td class="infolabel">Loc ID:</td><td class="infodata">#Variables.LocId#</td>
```

AppNav (suppose "Monthly Report" is currently being displayed in AppData):

```
<a target="AppData" href="whatever" class="menuitem">Weekly Report</a><br>
<span class="menuitem_hi">Monthly Report</span><br>
<a target="AppData" href="whatever" class="menuitem">Quarterly Report</a><br>
<a target="AppData" href="whatever" class="menuitem">Annual Report</a><br>
```

Because Monthly Report is currently being displayed in AppData, you have generated AppNav such that the Monthly Report is not a hotlink, but instead is the same color that hotlinks get when the mouse hovers over them. Presumably, if the user clicks the Quarterly Report hotlink, you would engineer the frames interaction such that AppNav would get regenerated with Quarterly Report in the , and Monthly Report in a hotlink. This technique shows the user which report is currently selected, with consistent look-and-feel.

Previously, this document gave an example of hardcoding the hightlighted color:

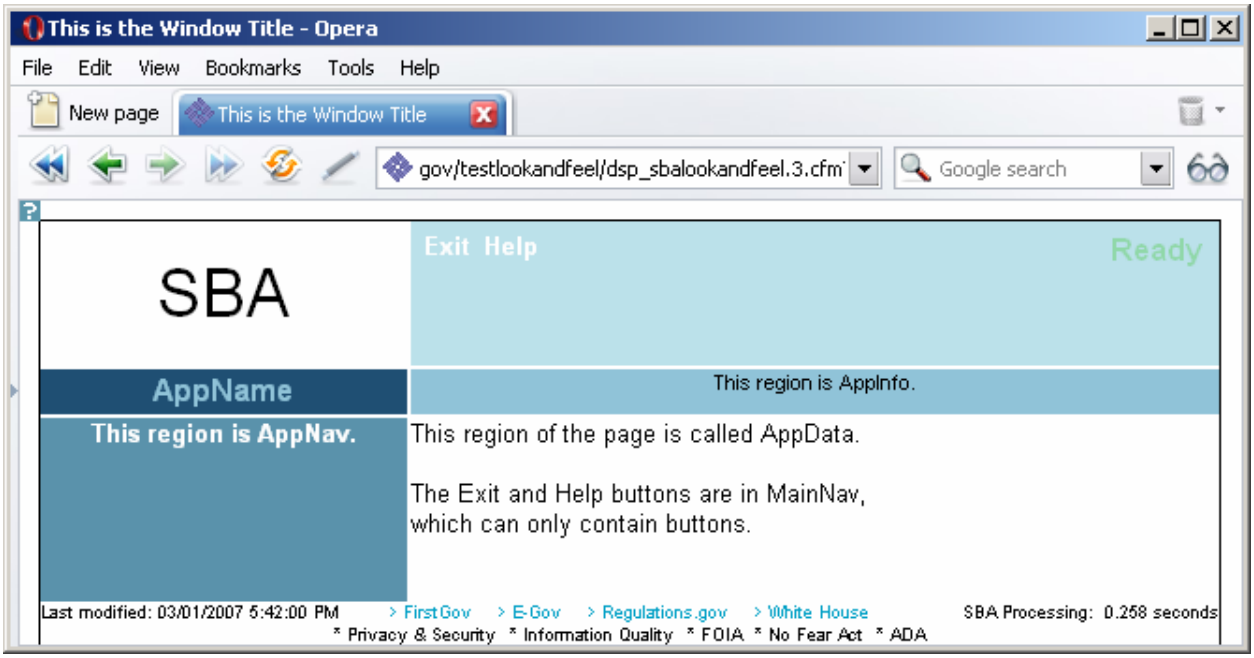
```
<span style="color:#ffffcc;">Monthly Report</span><br><!-- Don't do this! -->
```

Currently, the color of a highlighted hotlink with class="menuitem" is yellow (#ffffcc), so that would work. But it would misbehave if we changed the color scheme of menuitem in the future. So use class="menuitem_hi" instead. (Note the similarity to MainNav's highlighted button file names.)

4.1.12 The Screen Resizing Feature

Every page is initially displayed at the default size for MSIE for Windows at 1024x768 resolution. If JavaScript is turned on in the user's browser, SBA look-and-feel will then automatically resize the page to the height and width of the current window. In addition, if the user resizes the window, SBA look-and-feel will continue to automatically resize the page accordingly. But if JavaScript is not turned on, and the page will remain the default size.

4.1.13 The TextOnly Feature



SBA look-and-feel supports a TextOnly attribute which affects the SBALogo, the buttons in MainNav and the graphic backgrounds in MainNav and AppInfo. When the user requests the TextOnly version of your pages, you can simply say <cf_sbalookandfeel TextOnly="Yes"> (the old, one-page-at-a-time way to do it), and all of those graphics are converted to text hotlinks and other HTML equivalents.

TextOnly mode affects only the graphics that SBA look-and-feel generates. It doesn't affect the graphics in the HTML that you yourself display. It also doesn't affect the tiny folder and document icons of the DHTML tree generated by <cf_sbatree> and <cf_sbatreeitem>, although someday the automatic TextOnly feature might be able to tell those custom tags to use text equivalents as well. For your own, application-specific graphics, see 5.3.6.6 for an example of how to create an HTML equivalent.

As mentioned above, coding TextOnly="Yes" is the old, one-page-at-a-time way to set TextOnly mode. A new way exists to automatically convert your entire application. So before coding the TextOnly attribute, read 4.1.14 Automatic Screen Resizing and TextOnly. It could save you a lot of coding.

4.1.14 The Automatic TextOnly Feature

To allow automatic support for TextOnly, include the following after <cfapplication> in your Application.cfm file:

```
<!--- IncDir is an example name. See section 5.2.1, below. --->
<cfset IncDir           = "/library/cfincludes">
<cfapplication
    name                = "GLS"
    SessionManagement   = "Yes"
    SessionTimeout      = #CreateTimeSpan(0,1,0,0)#
    SetClientCookies    = "No">
<cfinclude template="#IncDir#/get_sbalookandfeel_variables.cfm">
```

Automatic TextOnly used to be tied to screen resizing, which explains why the attribute is still called AutoResize. Including get_sbalookandfeel_variables allows you to use the <cf_sbalookandfeel AutoResize="Yes" … > attribute. If you attempt to use this attribute without using the cfinclude, your page will crash because a variable won't be defined.

AutoResize defines a hidden form and a JavaScript function. To make it easy to remember, the JavaScript function is called top.AutoResize. It takes one parameter, "pToggleTextOnly". If it's true, SBA look-and-feel will switch automatic TextOnly from "Yes" to "No", or "No" to "Yes". If it's false, that switch won't be made.

The newest versions of SBA look-and-feel use cascading style sheet positioning (CSS-P) and JavaScript to adapt to new users' browsers, so the previous technique of using AutoSubmit="Yes" on "welcome" pages is no longer necessary. (It requires 2 hits on the server and messes up the user's Back button.) But a welcome page is still a good place to allow the user to toggle between Text Only and Graphics, like so:

```
<a href="javascript:top.AutoResize(true);">
    <cfif Request.SlafTextOnly>Graphics</cfelse>Text Only</cfif>
</a>
```

YOU DO NOT NEED TO ADD AUTORESIZE="YES" TO EVERY SBALOOKANDFEEL CALL IN YOUR APPLICATION. Once the automatic feature has been set or by a call to top.AutoResize, all pages controlled by the same Application.cfm will automatically adopt that TextOnly mode default. One welcome page is usually enough to convert an entire application!

4.1.15 Form Data Recovery

Part of SBA look-and-feel is to recover form data when a user returns to the last form page they were on. Examples of this concept are as follows:

- If a user enters search criteria and does a search, but then returns to the search criteria page to refine the search, all of the previous contents of that search criteria form should still be there, so that the user can tweak the criteria slightly without having to reenter all the previous criteria manually.
- If server-side data validation in an action page determines that the data entered cannot be saved to the database (alphabetic data in a numeric field, for example), the action page must return to the associated display page with error message(s) and all of the data entered by the user must be restored to the form as well.

To this purpose, SBA look-and-feel implements standard routines for saving and restoring form data in the Session scope. These routines are generically referred to as the "Form Data Recovery" facility. There is only one place to store form data, so it can be done only for the most recent form page the user was on. If the user goes to another page that uses Form Data Recovery, that page becomes the most recent and the previous form page's data is lost. We don't keep a history of all forms the user visited, just the last one.

There are 2 kinds of Form Data Recovery: manual and automatic. In manual Form Data Recovery, you call the "get" and "put" routines directly yourself, when you want them done and only at that time. In automatic Form Data Recovery, SBA look-and-feel calls the "get" and "put" routines for you.

<u>Manual Form Data Recovery:</u>

To **save** the Form scope to the Session scope, do the following:

```
<cfinclude template="put_sbalookandfeel_saveformdata.cfm">
```

To **recover** the saved form part of the Session scope to the Variables scope, do the following:

```
<cfinclude template="get_sbalookandfeel_saveformdata.cfm">
```

Variable names remain the same. If you need to set other variables containing "checked" or "selected" to restore checkboxes, radio buttons and drop-down menus, do so after calling the get routine.

As a programming convenience, Variables.Commentary and Variables.ErrMsg are also saved and restored along with the Form variables. (Use Commentary for normal completion messages such as "New contact saved to database." Use ErrMsg for abnormal messages, such as "Zip Code not numeric".) It's not a coincidence that Variables.ErrMsg is the same variable generated by Stored Procedure Call files. Rather, this is intentionally done so that if an error occurs, you can simply save form data (call the "put" routine) and return to the display page. Also, there's a standard routine to display server messages, Commentary and/or ErrMsg, which you should call at the top of your form display region:

If the form page is NOT in a frame:

```
<cf_sbalookandfeel ...>
<cfinclude template="/library/cfincludes/dsp_errmsg.cfm">
<cfoutput>
...
```

If the form page IS in a frame:

```
<body ...>
<cfinclude template="/library/cfincludes/dsp_errmsg.cfm">
```

Automatic Form Data Recovery:

Like Automatic Screen Resizing and TextOnly, Automatic Form Data Recovery requires the following line immediately after cfapplication tag that establishes the Session scope:

```
<cfinclude template="get_sbalookandfeel_variables.cfm">
```

If this line is already included in your Application.cfm file, the only step remaining to get Automatic Form Data Recovery is to define a hidden form element called PageNames (plural), containing the names of pages that share the same form variables.

Suppose the file containing the form is dsp_lend.cfm. You would simply add the following:

```
<input type="Hidden" name="PageNames" value="dsp_lend.cfm">
```

Also, get_sbalookandfeel_variables will define Variable.PageName (singular) for you to be the current file name, so you could also have said:

```
<input type="Hidden" name="PageNames" value="#Variables.PageName#">
```

That technique has the advantage of being easy to copy and paste into any form. It will pick up the page name of whatever page it happens to be in.

The reason why it's called PageNames (plural), is that it's actually a comma-delimited list. In general, it will be only the current page name (a one element list, hence no commas). But if a set of pages all share the same form variables, you can list all the pages in the set. For example, in TECH-Net, where dsp_search.cfm contains a search criteria form and dsp_awardlist.cfm uses the same search criteria to page through the results 25 awards at a time, you would say the following in the form in dsp_search.cfm:

```
... value="dsp_search.cfm,dsp_awardlist.cfm">
```

Automatic Form Data Recovery - Saving Just Commentary and ErrMsg (not automatic):

Automatic Form Data Recovery is done at the time of get_sbalookandfeel_variables, just after <cfapplication>, in Application.cfm. Therefore, the saving of the Form scope occurs then (before you've done anything). Hence, if you build a Commentary or ErrMsg and want it passed back to the display page, it must be saved separately. Suppose Inc contains "/library/cfincludes". The example below demonstrates how this may be done:

```
<cfif Variables.TxnErr>
    <cfinclude template="#Inc#/put_sbalookandfeel_savemessages.cfm">
    <cflocation ((back to display page))>
</cfif>
```

Or, if you want to pass back ErrMsg in the case of errors and Commentary in the case of success:

```
<cfinclude template="#Inc#/put_sbalookandfeel_savemessages.cfm">
<cfif Variables.TxnErr>
    <cflocation ((back to display page))>
<cfelse>
    <cflocation ((forward to success page))>
</cfif>
```

Automatic Form Data Recovery - Form Page Initialization:

You may be wondering, “if I use automatic Form Data Recovery, how do I know whether it’s the first time the user has entered the page (so as to display defaults), and how do I know whether it’s a return to the page (so as to display recovered form data)?” The answer is Variables.FormDataRecovered. If you include get_sbalookandfeel_variables.cfm immediately following your <cfapplication> tag, it will always be defined and contain “Yes” or “No”.

Also, the only safe time to apply formatting is immediately after a record first comes off of the database. Suppose that the database contains 50000 as someone’s salary, and you format it as $50,000.00 using the DollarFormat ColdFusion function. If form data is recovered, and you tried to format it with DollarFormat a second time, it would crash. You would be doing, in effect DollarFormat(“$50,000.00”), which is not a numeric input. Therefore (it bears repeating), the only safe time to apply formatting is right after a record first comes off of the database.

That said, suppose a page had only two form fields, called Gender and Salary, that could come from the database (existing record), or from a new record. An example initialization, partially in English and partially in ColdFusion, without Section 508 or other standards applied, would be as follows:

```
<cfif NOT Variables.FormDataRecovered>
    <cfif ((creating a new record))>
        <cfset Variables.Gender = "">
        <cfset Variables.Salary = "">
    <cfelse>
        ((read from database))
        <cfset Variables.Gender = GetRecord.Gender>
        <cfset Variables.Salary = DollarFormat(GetRecord.Salary)>
    </cfif>
</cfif>
<cfswitch expression="#Variables.Gender#">
<cfcase value="M"><cfset GM = "checked"><cfset GF = ""></cfcase>
<cfcase value="F"><cfset GM = ""><cfset GF = "checked"></cfcase>
<cfdefaultcase>   <cfset GM = ""><cfset GF = ""></cfdefaultcase>
</cfswitch>
<form ...>
<input type="Hidden" name="PageNames" value="#Variables.PageName#">
Gender:
<input type="Radio" name="Gender" #GM# value="M">Male
<input type="Radio" name="Gender" #GF# value="F">Female
Salary:
<input type="Text"  name="Salary" value="#Variables.Salary#">
</form>
```

Simpler case: In the case of a search criteria form, where there isn’t any data coming off of the database, you would simply initialize variables to their defaults:

```
<cfif NOT Variables.FormDataRecovered>
    <cfset Variables.Gender = "">
    <cfset Variables.Salary = "">
</cfif>
```

4.1.16 Features Requiring Some Knowledge of JavaScript

4.1.16.1 Controlling the "ReadyLight"

The ReadyLight attribute (values "Yes" or "No") controls whether or not to show the so-called "ready light". It says "Loading" against a white background when any frame isn't fully loaded yet. It goes dark (same shade of dark blue as the background) and says "Ready" when the page is fully loaded and ready for data input. Its primary usefulness is in a frames environment, but it can be used by any page.

At run time (on the browser), if it becomes necessary for one frame to tell another frame to reload, you can control the ready light by calling the JavaScripts generated for just that purpose.

If MainNavURL was NOT given (MainNav is inline), call:
top.SetReadyLightToLoading() or
top.SetReadyLightToReady().

If MainNavURL WAS given (MainNav is in a frame), call:
top.MainNav.SetReadyLightToLoading() or
top.MainNav.SetReadyLightToReady().

Normally, you would simply call SetReadyLightToLoading() and then tell the other frame to reload. The file in the other frame would set its own top.gFrameIsFullyLoaded[FrameName] variable to true and call top.SetReadyIfAllFullyLoaded(), which in turn would call SetReadyLightToReady() if all the frames are fully loaded.

SetReadyLightToLoading() and SetReadyLightToReady() will be generated differently depending on whether TextOnly is "Yes" or "No", so it's important not to bypass calling those functions and do what they do in your own code. If your code references the ready light graphic directly, and the user switches to TextOnly mode, your code will fail. So you should always go through the SetReadyLightToLoading() and SetReadyLightToReady() functions, because they will always be correct for whatever mode the user is in (TextOnly = "Yes" or "No").

<u>How the ReadyLight is typically used:</u>

Although it can be used in a no-frames page, the ReadyLight is typically used in a frames environment. Its purpose is to keep impatient users from entering data before all of the components of the page (frames) are fully loaded. Sometimes it can be tricky as to how to do this, and when to reload frames.

ELend provides a good example of how the timing problems can be resolved:

1. If the page in AppData modifies the structure of a loan application (adding a new borrower, for example), AppNav must be refreshed to show the new borrower in the navigation tree.
2. If the page in AppData modifies identifying information about the loan (the name of the loan or the loan amount, for example), AppInfo must be refreshed to reflect that new information.
3. If the page in AppData modifies the status of the loan, MainNav may have to be refreshed to show different buttons.

When should these various frame refreshes be done? If they're all done at the same time (at the time the user hits the Save/Next button in AppData), it's very likely that AppNav, AppInfo and MainNav will win the race to the database (because they don't do much), and so will refresh before the update implied by the AppData submission. Therefore, the next display page in AppData tells the other frames to refresh. So it's very important that the ReadyLight continues to say "Loading" until all the refreshes have completed.

4.1.16.2 Creating MainNav JavaScripts

For every button name you give in the <cf_sbalookandfeel> Show attribute, you have to create a Do((ButtonName)).js JavaScript that tells the browser what to do when the user clicks on that button.

As previously described in 4.1.5, the name of the JavaScript (both the file name and the function name it contains) are generated from the Show name as follows:

- Strip spaces and hyphens
- Add Do before the stripped name
- Add the ".js" suffix to form the file name

So <cf_sbalookandfeel Show="New Application,Help,Copy Application,E-Tran"> would require creating 4 JavaScript files:

- DoNewApplication.js containing function DoNewApplication
- DoHelp.js containing function DoHelp
- DoCopyApplication.js containing function DoCopyApplication
- DoETran.js containing function DoETran

You place them all in one directory, usually /applicationname/javascripts/sbalookandfeel. You then tell <cf_sbalookandfeel> where they are using the MainNavJSURL attribute. So in the case of TECH-Net, where the application directory is /technet (no hyphen) on danube, but /tech-net (with hyphen) on all other servers, you would say:

```
<cfswitch expression="#CGI.Server_Name#">
<cfcase value="danube.sba.gov">
    <cfset MainNavJSURL  = "/technet/javascripts/sbalookandfeel">
</cfcase>
<cfdefaultcase>
    <cfset MainNavJSURL  = "/tech-net/javascripts/sbalookandfeel">
</cfdefaultcase>
</cfswitch>
<cf_sbalookandfeel ...
    MainNavJSURL   = "#Variables.MainNavJSURL#"
    ... >
```

MainNav will try to invoke these JavaScripts automatically based on what it sees in Show. If the files don't exist, the browser will get a JavaScript error. It's no different than what occurs if the button graphics are not present in /library/images/sbalookandfeel or /library/images/applookandfeel. It tries to reference them wherever they're supposed to be. If they aren't there, the browser gets an error.

In the case of button graphics, if you need a new one, you can request that a new one be created for you. The same is true of MainNav JavaScripts. If you're not familiar with JavaScript, you can request that a MainNav JavaScript be created for your new button. The request will be routed to someone who knows JavaScript. Usually it's a matter of going to a particular URL when the user clicks on the button, so usually all you have to do is say what the URL should be along with your request.

To expedite page load rates, browsers cache JavaScripts from files with the js suffix. Caching is one of the reasons why we're allowed to use JavaScripts in the "thin client" of our application model. The downside is that those JavaScripts are inherently static. They can't contain ColdFusion variable references. But this limitation may be circumvented as explained in the section below.

Overcoming the limitations of static JavaScript:

Use MainNavHiddens to save <input type="Hidden"> tags in the form that contains the MainNav buttons. This allows you to access server data in the MainNav JavaScripts (which execute on the browser). For example, in TECH-Net, the index pages are in /technet/docrootpages (no hyphen) on danube, but they're in /tech-net/docrootpages (with a hyphen) on enile, and in / (the document root) in production. You could therefore do the following:

```
<cfswitch expression="#CGI.Server_Name#">
<cfcase value="danube.sba.gov">
    <cfset IndexPagesURL = "/technet/docrootpages">
</cfcase>
<cfcase value="enile.sba.gov">
    <cfset IndexPagesURL = "/tech-net/docrootpages">
</cfcase>
<cfdefaultcase>
    <!--- Have to omit trailing slash, hence: --->
    <cfset IndexPagesURL = ""><!--- document root --->
</cfdefaultcase>
</cfswitch>
<cfsavecontent variable="Variables.MainNavHiddens"><cfoutput>
<input type="Hidden" name="IndexPath" value="#IndexPagesURL#">
</cfoutput></cfsavecontent>
<cf_sbalookandfeel ...
    MainNavHiddens = "#Variables.MainNavHiddens#"
    ... >
```

Then, in DoHome.js, which gets invoked when the user hits the Home button in MainNav, you can do the following:

```
Function DoHome (pForm)
{
top.location.href = pForm.IndexPath.value + "/index.cfm";
}
```

Hence, although the JavaScripts invoked by MainNav are static (js suffix, not cfm suffix), they can nevertheless be made to perform dynamic functions (and know things that the server knows), as directed by hidden form tags in MainNavHiddens. That's the reason why all of the Do((ButtonName)) JavaScripts are passed a reference to the MainNav form, to give static JavaScripts access to ColdFusion data. It's a very powerful capability that assures that we will never be limited in any way by the static nature of scripts.

You could even look up a GLS logged-in user's username, find out the user's gender and family name, pass them in hidden fields, and address the user by name in alerts. ("Sorry, Mr. Williams, but the Help button is not yet implemented.") Of course, that would not be a professional thing to do on a government Website. It would establish an overly familiar tone and might even raise privacy concerns that the Website was somehow spying on the user. So it's just a facetious example, but one that illustrates the power and flexibility of MainNav.

4.1.16.3 AppNav DHTML Tree Using <cf_sbatree> and <cf_sbatreeitem>

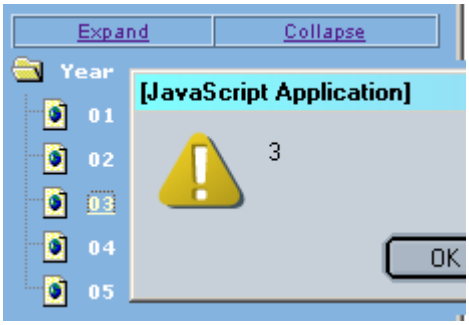
The SBA's <cf_sbatree> and <cf_sbatreeitem> custom tags were designed to exactly mimic <cftree> and <cftreeitem>. If you don't already know how to use <cftree> and <cftreeitem>, your learning curve for these custom tags is the same as learning how to use the ColdFusion versions. But if you already know how to use the ColdFusion versions, your learning curve is minimal. Here are the differences:

- In keeping with the "thin client" aspect of the SBA Application Model (see 3.1.1), our custom tags generate the tree using DHTML and cached JavaScript. The ColdFusion tags generate Java or Flash. It has been estimated that DHTML trees are so lightweight, they can handle upwards of 25,000 nodes, whereas Java bogs down after only a few hundred nodes and Flash at a few thousand. With our custom tags, the user doesn't have to install any plug-ins or wait on the initialization of the Java or Flash execution environment.
- <cf_sbatree> does not have to exist within a <cfform>. In fact, it doesn't even need to exist within an HTML <form>. That's because pressing a node triggers the execution of a JavaScript, not the submission of a form. You are responsible for defining the JavaScript, or you may request that one be created for you by one of the SBA's JavaScript specialists. It will be passed the value attribute of the node that the user clicked. Another, optional attribute you can provide is ExpandAll, which will initialize all folders to being expanded (<cf_sbatree ExpandAll="Yes">).
- <cf_sbatreeitem> has only 5 attributes, not the full set of attributes supported by <cftreeitem>:
  - **display** Display name of node, the text that appears on the screen, mandatory
  - **expand** "Yes" or "No" (currently not supported), optional
  - **img** "Document" or "Folder", mandatory
  - **parent** value attribute of Folder node that contains this node, optional if at top level
  - **value** value passed to your JavaScript if the user clicks the node, mandatory
- Our custom tags don't maintain any separate "Path" value that you can test on the server. In fact, you don't even necessarily invoke the server at all. The only thing you have is the value of the clicked node's value attribute in your JavaScript on the browser. If you want to track Path, you have to do so yourself with the value attributes you generate. If you want to pass this information to the server, you have to do so yourself.

Example (typically the top level folder has the value "root", but that didn't fit in this page, so using "Y"):

```
<cf_sbatree function="window.alert"><!--- Simply display value --->
   <cf_sbatreeitem display="Year" img="Folder" value="Y">
   <cf_sbatreeitem display="01" img="Document" parent="Y" value="1">
   <cf_sbatreeitem display="02" img="Document" parent="Y" value="2">
   <cf_sbatreeitem display="03" img="Document" parent="Y" value="3">
   <cf_sbatreeitem display="04" img="Document" parent="Y" value="4">
   <cf_sbatreeitem display="05" img="Document" parent="Y" value="5">
</cf_sbatree>
```

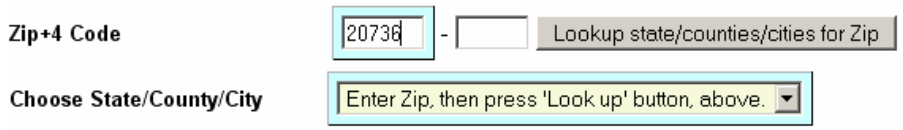
Result (after clicking 03):



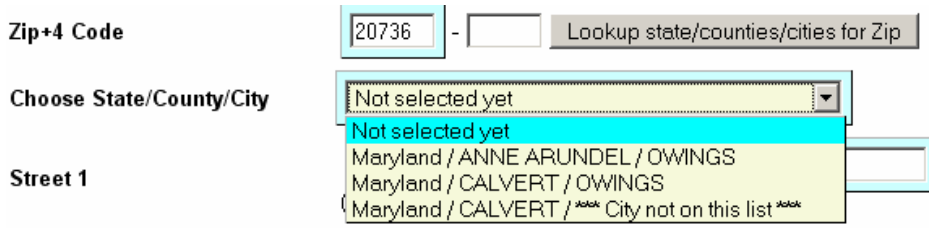
4.1.16.4 Server Callbacks in the AppHidden Frame

Sometimes, while the user is filling out data in one of your forms, you want to get information that exists only on the server. You don't want to submit the current form to do this, but you have to make some sort of server request. In this case, you might want to code a server callback in the AppHidden frame.

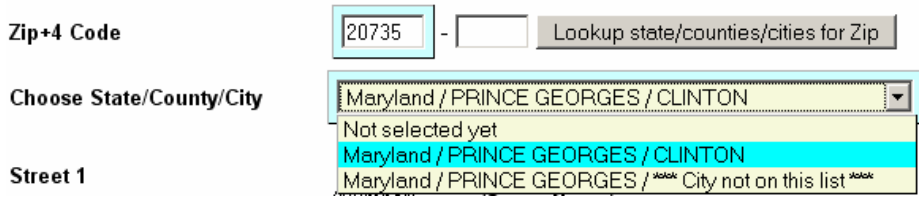
For example, suppose the user is filling out an address, and you'd like to supply the state, county code and city, based on the Zip code the user entered (screen snapshots from the ELend Project Info screen):



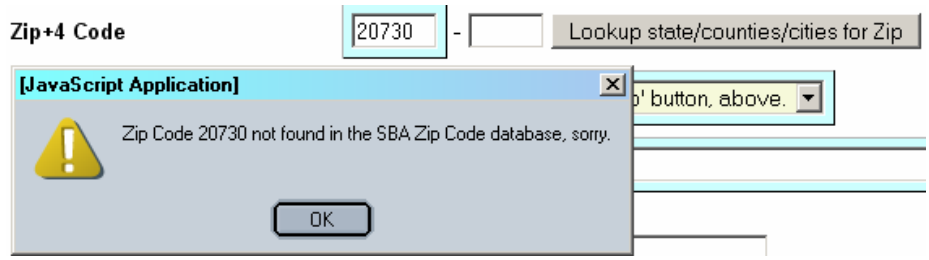
If more than one item is found, you want to alter the drop-down menu to show all State/County/City combinations, to let the user choose which one, and preselect "Not selected yet":



If there's exactly one State/County/City for the Zip code, you would still want to populate the drop-down, but in this case, you would want to preselect the only one found:



If nothing was found for the Zip code, you would want to pop up a JavaScript alert:



"How is this magic performed?" you may be wondering. Well, there are several steps:

1. In the <input type="button"> that initiates the lookup, define an onClick JavaScript to do a callback to the server in the hidden frame, AppHidden. There are 2 ways to do this:

```
<input type="Button" value="Lookup state/... (etc)" onClick="
top.AppHidden.location.href = 'dsp_callbackpage.cfm?Zip5Cd='
                            + this.form.Zip5Cd.value;
">
```

or

```
<input type="Button" value="Lookup state/... (etc)" onClick="
window.open('dsp_callbackpage.cfm?Zip5Cd='
            + this.form.Zip5Cd.value, 'AppHidden');
">
```

2. Write the server callback ColdFusion file to accept Zip5Cd on the URL and return a page with JavaScript to populate the drop-down menu (or display an alert that the Zip code wasn't found).

If you want to see an example of the server callback page used from the ELend Project Info screen example above, it's in **/elend/applications/dataentry/dsp_lookup_by_zip.cfm**. It's a rather complex example, because it can accept Zip with or without Zip+4. Furthermore, it accepts the form element names of the drop-down menu and other form elements to populate them as well. That's because it has to be used on pages that have 2 lookups by Zip, one for physical address and one for mailing address. So to read it, you should also look at **/elend/applications/dataentry/dsp_proj.cfm** (the Project Info screen used in the examples above), so that you can see its form element names.

In the future, AppHidden frame callbacks will be generalized for use in any application and reside in /library/callbacks. Then, if you want to add a Zip code lookup, NAICS code lookup, franchise code lookup, etc, you have a standard place to look for existing code to do so.

<u>Warning about Microsoft Internet Explorer 5 and above:</u>

As of MSIE version 5, you can no longer populate a drop-down in another frame directly.

In order to write JavaScript that's cross-browser compatible and works in MSIE 5+, you must define a JavaScript function in the frame that defines the drop-down menu. (In SBA look-and-feel, the frame that defines the drop-down would generally be AppData.) In that function, you write general-purpose code to populate the drop-down menu's options array (clear the array, add a new option to the end of the array, etc). If you find it easier to do so, write several general-purpose functions.

Then, when you write the server callback in AppHidden, the JavaScript it generates must administer calls to the general-purpose function(s). In other words, in the usual case, the callback in AppHidden will do JavaScript calls of the form **top.AppData.FunctionName(parameters)** to manipulate the drop-down in AppData.

Perhaps in the future, the ELend server callbacks in AppHidden will be made into general purpose utilities you can call out of /library. In fact, with current emphasis on shared code, they probably will be. At the moment, however,, they are not in /library, so you would have to use the routines in /elend as examples.

4.1.17 MainNav as a Frame

Generally speaking, the contents of MainNav don't change much. Usually, the buttons the users see there are usually based on the user's permissions and roles in the application. Consequently, it's a good candidate for turning into a frame.

In the past, when we couldn't mandate that users have JavaScript turned on, there was no way to make the MainNav buttons do anything without defining a <form> and an action page that the form went to. Typically, the page that images the MainNav frame would be called dsp_mainnav.cfm, and its action page would be called act_mainnav.cfm. The idea was that the action page would do whatever the JavaScript would have done, if the user had JavaScript turned on.

In time however, it became apparent that there were some things that this design just simply could not do. In particular, it was impossible to vary the "target frame" for the action to take place. MainNav could not force the resubmission of AppData (standard look-and-feel button "Save") unless ALL the buttons caused the resubmission of AppData. Similarly, it would be impossible for some buttons to pop up a new window without all buttons popping up a new window.

Therefore, currently, we use JavaScript in MainNav (see 4.1.16.2) and the only real consideration is how to generate the buttons. The answer is, you call a different custom tag, <cf_mainnav>, as follows:

```
<cf_mainnav
    attribute=value
    attribute=value
    ...>
```

Note that there isn't any </cf_mainnav>. Attributes (defaults of multiple-choice features shown in **bold**):

```
Configs       - Used in development of new versions of custom tag
Debug         - Used in development of new versions of custom tag
InFrame       - "Yes" or "No", because you're in a frame, use "Yes"
LibURL        - Used in development of new versions of custom tag
MainNavHiddens- Used to pass CF data to MainNav JavaScripts
MainNavJSURL  - Directory of MainNav JavaScripts
ReadyLight    - "Yes" or "No" – whether to turn on feature
Show          - List of buttons to appear in MainNav
ActionURL     – Old, URL to handle submission of MainNav form.
TextOnly      - Old, "Yes" or "No",   nowadays use AutoResize
```

Note that, except for InFrame and Width, these are the same features that got an asterisk in 4.1.4, indicating that they would be passed to an internally generated call to <cf_mainnav> when you aren't putting MainNav into a frame. So all of your knowledge of <cf_sbalookandfeel> carries over.

It's important that features match up to the frames document that defines MainNav. (If the frames page says ReadyLight="No" but the call to <cf_mainnav> says ReadyLight="Yes", there will be a JavaScript error because the ReadyLight won't be defined.) Also, you have to <link> /library/css/sba.css and pick up the proper MainNav background color with <body class="headernav">. Other than that, it's easy.

4.1.18 Using SBA Look-and-Feel on a Static HTML Page

Now that the automatic resizing feature is taking place in HTML, CSS and external JavaScript (not in ColdFusion), its quite possible to use the same tags on a static HTML page. If you do this, the page will resize itself every time the user resizes its window, and the look-and-feel will be compatible with all of our ColdFusion applications that use SBA look-and-feel.

In "new SBA look-and-feel", ColdFusion is used just to build the and <iframe> tags properly. So, generally speaking, the easy way to build the tags properly for a static page is to use a ColdFusion page and capture the source with View Source. The resulting static page will display properly on a server that doesn't have ColdFusion.

Prerequisites:

- /library/css/sba.css
- /library/css/sba.textonly.css (if text only) - <link> tag AFTER sba.css (to override its classes)
- /library/images (if NOT text only)
- /library/javascripts/sbalookandfeel/sbalookandfeel.js
- your own JavaScripts, associated with the MainNav buttons you use

Containment hierarchy for tags (give these values as the id attribute):

```
(body)
        DivMarginT
        DivMarginR
        DivMarginB
        DivMarginL
        DivEnvelope
                DivSBALogo
                DivMainNav
                DivAppName
                DivAppInfo
                DivAppNav
                DivAppData
                DivBotMost, in its entirety (<ul> and <li> tags too)
         (DivAppHidden is not needed in a static environment, because callbacks require ColdFusion.)
```

Comments:

- DivSBALogo must have style.display already defined for the AppData maximize/minimize feature to work. Therefore, even if you don't give the background style because the page is text only, you still need style="display:;" or style="display:block;", to define style.display.
- For the same reason, AppData's style.left and style.width must already be defined. So always specify style="left:___px; width:___px;" (according to the size of AppData).
- To keep sbalookandfeel.js from erroring, all of the tags listed above have to be defined. So if you want to not have AppInfo and expand MainNav downward into AppInfo's space, specify AppInfo's style="display:none;" and override the height of MainNav with a style attribute.
- If you define any of the tags as containing frames, give the frame the same name as the Div that contains it, minus the "Div" part. And give the frame the same id attribute, changing Div to Frm. Example <iframe name="AppData" id="FrmAppData" … >.
- If you define frames for MainNav or AppInfo, specify the scrolling="No" attribute.
- Again, it's easiest to let cf_sbalookandfeel generate this stuff correctly for you and just copy the HTML from View Source. The custom tag knows all the rules.

4.1.19 Read the Custom Tags to Get More Information

The most recent and current information about cf_sbalookandfeel and cf_mainnav can be found in the comment headers of the custom tags themselves. At most ColdFusion installations, custom tags are typically in the /opt/coldfusion[mx[7]]/CustomTags directory. But at the SBA, they're under /library:

/library/customtags/sbalookandfeel.cfm
/library/customtags/mainnav.cfm

Every attribute is documented in the comment headers, even the more obscure ones that few developers ever use. Furthermore, by reading the custom tags, you can see how they do what they do. It makes them less mysterious.

4.2 Stored Procedure Call Files

The stored procedure call files (prefix “spc_”) greatly sped up our migration to ColdFusion MX. They have been programmed to allow lots of tricks to get the most out of each call.

4.2.1 Make Sure that the SPC Files Have Been Generated

SPC files must be generated by a utility called the “Stored Procedure Call File Generator”, or SPC file generator for short.

The SPC file generator

- asks for a login that’s a valid user in the database being generated for
- reads the database scripts that generated the stored procedures
- parses them to find the parameter list
- uses the parameter list to generate CFPROCPARAMs in the right order, datatype, etc
- generates the SPC files into /cfincludes/dbname, where dbname is the name of the database

Not all developers are permitted to generate SPC files because only a few can create the directory into which the SPC files are stored. Rather, you must request that the SPC files be generated if they don’t already exist.

4.2.2 Request Regeneration of SPC Files Whenever Parameter Lists Change

Stored procedures can be recompiled over and over again without having to regenerate the SPC files, provided that their parameter lists have not changed. If their parameter lists change, however, they must be regenerated.

4.2.3 Load Only the Columns You Need into the Variables Scope

The SPC file uses CFPARAM to default every input parameter to the nullstring. Anything you load into the Variables scope before the SPC file call will override the CFPARAM. Therefore, if a stored procedure has an Identifier (action code) that requires only 2 of its 20 input parameters, simply set Variables.Identifier and the 2 parameter names in the Variables scope. The SPC file will do the rest for you.

4.2.4 But Use Defaults Sensibly

If you’re conducting only one SPC file call in a ColdFusion page, go ahead and use the defaults all you want. (The defaults are all the CFPARAM tags at the start of the SPC file.) But if you’re making multiple calls, remember that once a variable is defined, it will be used on subsequent calls and the CFPARAM tags will have no effect. This is particularly true of parameter names commonly used by the database group, such as @Identifier. Once Variables.Identifier is defined, it will continue to be used in subsequent calls with the last value it was given. Therefore, if lots of stored procedure calls are involved, it’s better to Variables.Identifier explicitly on each call.

Always set parameters of type bit to 0 or 1. Parameters of type bit cannot be null.

4.2.5 Use LogAct to make error messages more user-friendly

The SPC files contain standardized error handling. Not knowing what you were intending to do with the call, the SPC files simply say "While trying to #Variables.LogAct#, …" and default LogAct to "call ((name))". So if you don't set your own LogAct ("logical action"), your error messages will look like this: "While trying to call FrnchsSelTSP, ..." Jargony names like that can frighten non-computer-savvy users who may worry that they broke something important. So if the reason for the call was to validate a franchise code the user gave you, you can load Variables.LogAct with "validate the franchise code" before calling the SPC file. That way, if a database error occurs, the message will be much friendlier: "While trying to validate the franchise code, ..."

4.2.6 Use Variables.TxnErr for Transaction Control

TxnErr carries on a major role in transaction control. Per a rule established by the database group, once TxnErr has been set to "Yes" by any of the SPC files in a transaction, none of the remaining calls to SPC files will try to call their associated stored procedures. Furthermore, you can use it to decide whether to roll back the transaction.

Example:

```
<cftransaction action="BEGIN">
   <!--- Numerous calls to SPC files occur here. --->
   <cfif Variables.TxnErr>
      <cftransaction action="ROLLBACK" />
   <cfelse>
      <cftransaction action="COMMIT"  />
   </cfif>
</cftransaction>
```

The two inner CFTRANSACTION tags have a trailing slash inside them to indicate that they have no closing tag. This is a convention from XML that was adopted by ColdFusion for this purpose, so as not to prematurely terminate the outer CFTRANSACTION tags.

4.2.7 Retrieving Single Result Sets

If all you want to retrieve is one result set, you can use Variables.cfprname to set the query variable name for the result set. If you want to retrieve a result set other than the first one, set Variables.cfprset (which is otherwise defaulted to 1). When the CFPROCRESULT tag is generated ("cfpr") it will use these for the name and set attributes. The default value for Variables.cfprname is "Ignored", because you're probably going to ignore the result set if you didn't set Variables.cfprname to something more meaningful.
Example:

```
<cfset Variables.Identifier   = 11>
<cfset Variables.LoanAppNmb   = Form.LoanAppNmb>
<cfset Variables.LogAct       = "retrieve collateral">
<cfset Variables.cfprname     = "getCollat">
<cfinclude template="/cfincludes/loanapp/spc_LoanCollatSelTSP.cfm">
<cfif NOT Variables.TxnErr>
   <cfif getCollat.RecordCount GT 0>
   <!--- etc --->
   </cfif>
</cfif>
```

4.2.8 Retrieving Multiple Result Sets

The SPC files allow you to use Variables.cfpra ("CFPROCRESULT array") to retrieve multiple result sets,. If it's defined and is a ColdFusion array, the SPC files expect the name to be in column 1 and the set to be in column 2 of the array. The following is a technique to allow easily adding, deleting or re-ordering the result sets in response to stored procedure changes:

```
<cfset Variables.cfpra         = ArrayNew(2)>
<cfset i=0>
<cfset i=i+1><cfset cfpra[i][1]="getNAICS"><cfset cfpra[i][2]=i>
<cfset i=i+1><cfset cfpra[i][1]="getStats"><cfset cfpra[i][2]=i>
<cfset i=i+1><cfset cfpra[i][1]="getPrins"><cfset cfpra[i][2]=i>
<cfset i=i+1><cfset cfpra[i][1]="getBorrs"><cfset cfpra[i][2]=i>
<cfset i=i+1><cfset cfpra[i][1]="getGuars"><cfset cfpra[i][2]=i>
<cfset Variables.LogAct        ="retrieve loan data">
<cfinclude template="/cfincludes/loanapp/spc_LoanPrintSelCSP.cfm">
<cfif Variables.TxnErr>
   <cfoutput>#Variables.ErrMsg#</cfoutput>
   <cfabort>
</cfif>
<cfset ArrayClear(Variables.cfpra)>
<cfset Variables.cfpra         = "">
```

Note the last 2 lines. If you leave Variables.cfpra as a ColdFusion, it will continue to be used in future SPC file calls. This is a safe way to make sure that that doesn't occur.

4.2.9 Calling a Stored Procedure in a Different Database

Calling a stored procedure in a different database requires that one first manually edit an SPC file. SPC files are usually generated. Therefore, you generally don't want it to reside in the cfincludes subdirectory associated with its database. Otherwise, the next time the entire database's SPC files are regenerated (perhaps to add a new feature to all of them), it will get overlaid and you'll have to re-edit it manually again. If it's application-specific code, you might copy it to a subdirectory of your application's top directory.

Suppose you need to access sbaref's TechAreaCdSelTSP from within the fast datasource. There are 2 stored procedures with that name, one in the fast database and one in the sbaref database. The standard SPC files would be in:

```
/cfincludes/fast/spc_TechAreaCdSelTSP.cfm
/cfincludes/sbaref/spc_TechAreaCdSelTSP.cfm
```

Though rather complicated, there are 2 ways to call the sbgaref stored procedure from the fast datasource:

(1) You could copy it out of the standard /cfincludes directory entirely, and put it into your application directory, clearly identifying it as application-specific code:

```
/fast/cfincludes/sbaref/spc_TechAreaCdSelTSP.cfm
```

(2) Alternatively, if you think someone else in some other application might have need to do the same thing, you could keep it in the standard /cfincludes directory hierarchy but change its name. That is, copy it from the sbaref subdirectory to fast subdirectory, while changing its name to:

```
/cfincludes/fast/spc_sbaref_TechAreaCdSelTSP.cfm
```

The choice of where to create the new file is based on whether another application needs to do the same thing (call the sbaref stored procedure using the fast datasource).

Either way, your new custom SPC file will not conflict with either of the standard, generated SPC files. On the next regeneration of standard fast or sbaref SPC files, the new file you've just created won't be overlaid.

Now you have to modify the new file. The trick is to make sure that every output CFPROCPARAM has a value attribute, even though it won't be used. Here's what you have to do:

(1) In the new file, edit <cfstoredproc procedure="sbaref..TechAreaCdSelTSP" …
(2) In the new file, add value="" to every <cfprocparam type="Out" …

It's not clear why this works, nor whether or not it's still necessary add the value clauses under CFMX 7. (It hasn't been tested yet under CFMX 7. Regardless, the file would still have to be copied, so that the cfstoredproc procedure attribute can be manually changed without risk of being overlaid.)

4.2.10 How to use it

- Login > Choose Function > Developer Utilities > Generate SPC Files.
- Select datasource. New options will become available (dynamic HTML).
- Answer new fields on the screen:
    - If the login associated with the datasource can read stored procedure scripts (and most can), leave username and password blank. Else enter a Sybase or Oracle database login that can, according to whether the datasource is Sybase or Oracle.
    - Some databases have "internal management" stored procedures (beginning with "IM") for everything. In that case, select "Include IMs".
    - If the application has public functions, such that "select stored procedures" (ending in SelTSP or SelCSP) can be done withotut logging into GLS, select "Datasource" on the next line.
    - And if you only want to generate for a stored procedure that was recently changed, sort by "Create Date", so that it'll be near the top of the list that'll be in reverse chronological order.
- Press "Get Stored Procedure Names".
- Check the stored procedures you want to regenerate. There are also checkboxes at the top and bottom of the list to Check/Uncheck All. Press "Generate Checked".
- Last but not least, this process is state-sensitive. So if you get in trouble (selected the wrong thing perhaps), start over from the beginning.

4.3 Logging

There are now logging routines that can be used (some automatically, some manually) to log events in applications for performance monitoring purposes, not for audit trails of security violations or anything else which may violate privacy regulations. The logs are accessible with any Web browser, so specifically may NOT store anything security or privacy related in the logs. We may, at some time in the future, choose to extend the logging feature with other types of logs, but for now, our logging routines are for performance monitoring only.

The standard SBA logging routines use "log4j" an open source Java-based logging interface that imposes almost no performance penalty at all (nanoseconds) when logging is turned off and minimal overhead (microseconds) when it's turned on. Furthermore, logging can be turned on or off from outside the application. For this reason, all SBA ColdFusion applications will be required to support logging unless granted a specific exemption.

To prevent a worst-case scenario where logging is crashing CF pages, and we need to get into the logging administration CF pages to resolve this matter, the logging administration pages themselves are exempted from supporting logging. Otherwise, it's envisioned that no other applications will be exempt. Logging will be a global, system-wide feature on all SBA servers.

4.3.1 Turning On Logging Support – The "Master Switch"

Since the summer of 2005, CF pages have been required to perform the following include as soon as possible in the execution of the page:

```
<cfinclude template="/library/cfincludes/inc_starttickcount.cfm">
```

This is usually done in an Application.cfm file, so that all pages within a directory are automatically converted to compliance with this standard. Since this precedes the setting of configuration parameters (see 5.2.1), the path must be hard-coded.

The comment header (see 3.3.5) takes up no execution time, so it can precede this include, and applications that require setting CFOutputOnly mode must have that as the very first line. The master switch that turns on logging support is setting Request.SBALogSystemName before the include as well. Therefore, the order of statements at the beginning of the page (usually in Application.cfm) is as follows:

```
<cfsetting enablecfoutputonly="Yes"><!--- If the app uses this. --->
<!---
AUTHOR:       ... (etc, required comment header, see 5.2.1)
--->

<cfset Request.SBALogSystemName = "pronet">
<cfinclude template="/library/cfincludes/inc_starttickcount.cfm">

<!--- Configuration Parameters: --->
...
```

The reason why SBALogSystemName is in the Request scope, not the Variables scope, is so that we can add logging to custom tags, regardless of how deeply they're nested in other custom tags.

4.3.2 What to Use as the System Name – GLS Systems

For applications that require logging in through GLS, if there is only one GLS System name associated with it, the value of Request.SBALogSystemName. For example, in the Eligibility Checklist application, there is only one GLS System name ("Eligibility"), so you would set Request.SBALogSystemName to "Eligibility":

```
<cfset Request.SBALogSystemName = "Eligibility">
<cfinclude template="/library/cfincludes/inc_starttickcount.cfm">
```

If more than one GLS System name exists for the application, it must initially be set to "GLS" and then later reset to the specific GLS System name as soon as it's been determined. For example, in Electronic Lending, if the URL of the page is in the /elend/applications directory, or its subdirectories, the GLS System name is "LoanOrig". But if the URL of the page is /elend/servicing directory, or its subdirectories, the GLS System name is "LoanServ". So the overview of what needs to be done is:

```
<cfset Request.SBALogSystemName = "GLS">
<cfinclude template="/library/cfincludes/inc_starttickcount.cfm">

...
<cfswitch expression="#Variables.RequestedSubsystemDir#">
<cfcase value="applications">
    <cfset Request.SBALogSystemName = "LoanOrig">
    ... <!--- other inits, such as username and password --->
</cfcase>
<cfcase value="servicing">
    <cfset Request.SBALogSystemName = "LoanServ">
    ... <!--- other inits, such as username and password --->
</cfcase>
</cfswitch>
```

Note that you don't redo the include of inc_starttickcount.cfm. That must be done only once, as described in the previous section. The resets of Request.SBALogSystemName simply cause all FUTURE logging requests to go to the log file for the one appropriate to your system.

4.3.3 What to Use as the System Name – Non-GLS Systems

Your application will be issued a "Non-GLS System name" by the applications administrator for production applications. This is what you will use to set Request.SBALogSystemName. The Non-GLS System name will be composed solely of letters and numbers, without spaces or special characters. This name is also case-sensitive.

Examples of Non-GLS Systems include PRO-Net (admin functions), TECH-Net (admin and federal agency functions), the ELend Check-In/Check-Out Utility, the Stored Procedure Call File Generator, SrTars, etc.

As Non-GLS Systems are gradually, converted over to GLS, the list of Non-GLS Systems will get shorter and shorter, but it will never disappear. The reason is that there's at least one application that can never be made into a GLS System, namely, GLS itself. You can't require a user to be logged into GLS in order to log into GLS!

4.3.4 All Developers Will Be Application Administrators in Development

It is envisioned that ALL developers will be application administrators in development, so that you will be able to turn logging on and off for your own code for debugging purposes. This privilege, however, disappears in test and production. Only SBA employee managers will be application administrators in test and production.

Note that this means that you will be able to edit the list of Non-GLS System names. If you're working on a Non-GLS System, you will be able to name it whatever you like. For example, if you're working on PRO-Net, you can give it the Non-GLS System name of "pronet", or "PRONet" or "PRONET" (not "PRO-Net", however, due to the restriction that the name must be composed of letters and/or numbers).

This offers the illusion that you have more freedom than you actually have. In actuality, you don't pick the Non-GLS System names, the application administrator for production does. You have to go along with that name because the name in development must match the name in test and production.

Suppose you use "PRONET" in development, but it's defined as "PRONet" in production. When the production application administrator tries to turn on logging in production, logging won't turn on, because the case-sensitive name won't match.

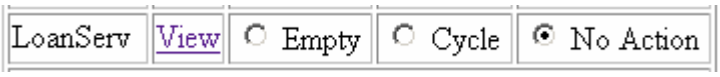
4.3.5 The CF/Logging Admin Pages

If you have been given the "SecurityAppAdmin Role" in GLS, when you go to the Choose Function page, you will see a hotlink "CF/Logging Admin". If you follow that hotlink, you will be taken to the CF/Logging Admin pages. In the AppNav (left) frame, you will see hotlinks for Non-GLS Systems, Logging and Log Files.

In the Non-GLS Systems page, you will be able to control the list of Non-GLS Systems:

In the Logging page, you will be able to turn logging on at a variety of levels:

In the Log Files page, you will be able to empty out log files or copy them to a cycled backup file name (examples, log_GLS.1.txt, log_GLS.2.txt, log_GLS.3.txt, etc) before emptying them out:



4.3.6 Logging Levels – Debug, Info, Warn, Error and Fatal

Logging is turned on at a particular LEVEL, which is currently Debug, Info, Warn, Error or Fatal (in order of increasing severity). When a particular level is turned on, every more severe level is also turned on. You may also set logging to All or Off, although these not actually logging levels, since you can't write log entries at the All or Off levels. All is just a synonym for whatever the lowest level is, and Off is just off, no logging will be done at all. In the Logging page, they're listed as follows:

In the future, CFMX may support a newer version of log4j which supports logging at the Trace level as well. (Trace is between All and Debug.) Trace allows you to enter tons of logging requests (one for each branch of a <cfif>, one for each <cfcase>, etc) but not turn the Trace logs on unless you absolutely have to. (You could turn Debug logs on without getting a full trace of everything that got done.)

But for now, the highest level of log4j that CFMX supports is 1.1.3, which doesn't support the Trace level. For now, you have to use Debug level if you want to trace execution flow in your application.

It is critical to remember that At The SBA, performance monitoring is logged at the info level.

4.3.7 Manual Logging Routines That You're Required To Add

ColdFusion features numerous automatic logging features. The Stored Procedure Call File Generator, for example, is going to be modified to do logging. With this feature in place, you won't have to add anything to support logging stored procedure calls, as you should already be using SPC files, you won't have to add a thing to support logging stored procedure calls.

For other log entry types, on the other hand, you will have to add things manually. Starting with the easiest one first:

4.3.7.1 At Start of Request - inc_starttickcount.cfm

Since the summer of 2005, CF pages have been required to cfinclude inc_starttickcount.cfm as soon as possible in the execution of the page: The original reason was so that it could call the built-in ColdFusion function TickCount(), which later allows cf_lastmodified to display the total elapsed time to process the page.

Later, this became a perfect place to add a logging call to log the start of execution of the request. Currently, this includes a standard for 2 reasons: Display of elapsed time and logging start of request.

See 4.3.1 for a discussion of precisely where it belongs (generally speaking, between the comment header and configuration parameters of Application.cfm).

4.3.7.2 At End of Request - OnRequestEnd.cfm

To capture end-of-request logs, create a file called OnRequestEnd.cfm in the same directory level as Application.cfm. In it, the only executable statement you need is as follows:

```
<cfinclude template="/library/cfincludes/OnRequestEnd.cfm">
```

In addition to doing an end-of-request log, it will also turn off ShowDebugOutput unless you specifically allow it with <cfset Variables.Debug = “Yes”>. This causes your page to display as it would in test or production if everything went well (so you get to end-of-request). But if you crash, this code isn’t done, so you still get to see debug output. It’s like having ColdFusion debugging on “only when you need it”. If you don’t want this feature, set Variables.Debug to “Yes” just before the include tag.

Because OnRequestEnd.cfm is executed on normal completion of every page, you have effectively added end-of-request logging to all of your application’s pages, all at once. OnRequestEnd.cfm file affects a page only if the associated Application.cfm in the same directory was used at start-of-request time. So if you have more than one Application.cfm file, you will need to create one OnRequestEnd.cfm for each of them.

4.3.7.3 log_SleQuery.cfm

log_SleQuery.cfm can be done with a global search and replace command. Just after every </cfquery>, add the following:

```
<cfinclude template="/library/cfincludes/log_SleQuery.cfm">
```

4.3.7.4 log_SleCatch.cfm

log_SleCatch.cfm can also be done using a global search and replace command. Just before every <cfcatch>, or before every </cfcatch> (but not both because you only want to do it once), add the following:

```
<cfinclude template="/library/cfincludes/log_SleCatch.cfm">
```

4.3.8 Manual Logging Routines That Are Optional

We also have other logging routines that you have to add manually, but only if you want to use them.

4.3.8.1 log_SleCustom.cfm

For anything you want.

Under construction.

4.3.8.2 log_SleTimeBeg.cfm, log_SleTimeEnd.cfm and log_SleTimeAccum.cfm

For timing segments of code, logging them one at a time or just totals at end-of-request.

Under construction.

4.3.8.3 log_SleTrace.cfm

We're planning on a way to support Trace logging even though it isn't supported yet by log4j 1.1.3.

Under construction.

4.3.9 Where the Log Files Reside

Logs reside at http://servername/logs/log_XXX.txt where XXX is the Request.SBALogSystemName. Examples:

http://danube.sba.gov/logs/log_LoanOrig.txt
https://yes.sba.gov/logs/log_SrTars.txt
http://tech-net.sba.gov/logs/log_TECHNet.txt

Logs don't exist until logging has been turned on for the particular GLS or Non-GLS System, so the hotlinks above may not yet exist. You can also browse to the directory name (we haven't turned of directory listing), so that you can see all existing log files.

Viewing them in a Web browser is convenient and acceptable when you're in a hurry, but the tab characters that the log files contain are rendered irregularly by browsers. If you want to see the columns more accurately, do the following:

- Do a File > Save As… in the browser, or download the file using FTP (if you have an FTP account on the server to do so).
- Once it's on your PC, right click on it.
- In the pop-up menu, select "Open With …" and choose Excel from the application list.

4.3.10 Cooperating With Other Developers in Development

Since all developers will be application administrators in development, the potential exists for logging conflicts in which different developers may log on or off a system simultaneously. To prevent this from occurring, it is essential that you do not touch a system if it's not your own.

Hence, if you go into the Logging page to turn on logging for GPTS at the Debug level, and logging is already on for LoanOrig at the Info level, simply turn on logging for GPTS. Don't touch the setting for LoanOrig.

Of course, there's still the possibility that 2 developers could encounter conflict if they both entered the Logging page at the same time. Neither would see the change being made concurrently. Whoever hit Save last would have his/her changes, that would become the current logging setting. Since there's no way to prevent this from occurring, it is recommended that you call the developer who is simultaneously logged in to notify them and coordinate your logging changes.

4.4 Standard Callbacks

4.4.1 dsp_LookupZipToDropdown.cfm

The ELend feature of looking up Zip codes and populating form fields was described above in section 4.1.16.4, Server Callbacks in the AppHidden Frame. Due to its usefulness, this was the first callback routine added to /library/callbacks. To use dsp_LookupZipToDropdown.cfm, there are only 2 things that a data entry form in the AppData region of SBA look-and-feel needs to do:

1. Define the LookupZipToDropdown script and the 2 other library scripts it uses (EditMask and SetFormEltValue). Some browsers compile JavaScript as soon as it loads. To be compatible with all browsers, EditMask and SetFormEltValue should be defined before LookupZipToDropdown, so that LookupZipToDropdown's references to the other 2 scripts won't result in JavaScript errors.
2. Call the JavaScript LookupZipToDropdown properly.

Everything else is handled by /library.

Defining the scripts if AppData is a frame: Somewhere in the <head>, add the following (if you don't already have them). If you've defined a configuration variable for /library/javascripts, you should use it instead of the hard-coded references shown here, of course:

```
<script src="/library/javascripts/EditMask.js"></script>
<script src="/library/javascripts/SetFormEltValue.js"></script>
<script src="/library/javascripts/LookupZipToDropdown.js"></script>
```

Defining the scripts if AppData is inline (almost the same):

```
<cfsavecontent variable="Variables.JSInline">
    <cfoutput>
<script src="/library/javascripts/EditMask.js"></script>
<script src="/library/javascripts/SetFormEltValue.js"></script>
<script src="/library/javascripts/LookupZipToDropdown.js"></script>
</cfoutput>
</cfsavecontent>
...
<cf_sbalookandfeel ... JSInline="#Variables.JSInline#" ... >
```

Calling LookupZipToDropdown properly: You would generally call LookupZipToDropdown in the onClick of a button or in the onChange of a text box for Zip and/or Zip+4. It could conceivably also be used in the onLoad of a <body> tag to initialize the form with values from the database. It has 2 mandatory arguments and up to 5 more optional arguments, described below.

If an optional argument is at the end, it can simply be omitted. For example:

```
LookupZipToDropdown ("90210", this.form.MyMenu);
```

But if an optional argument is in the middle (followed by other arguments), you have to pass the JavaScript keyword "null" (without quotes) instead. For example:

```
LookupZipToDropdown ("90210", this.form.MyMenu, this.form.StCd,
                      null, this.form.CtyNm);
```

The arguments are as follows.

1. In order for it to be usable in the widest range of circumstances, the first argument ("pZip") is a string. Its format is "99999" (Zip) or "99999-9999" (Zip/Zip+4). Accepting the pZip input as a string allows LookupZipToDropdown to be used with one form field to enter Zip/Zip+4 together, to 2 separate form fields, or even with a value returned from the database.
2. The dropdown menu (<select> form element) that will receive the output (<option> tags) is the second argument ("pEltDropdown"). The "Elt" part of its name in the formal arguments list reflects the fact that it is a form element reference. Since you will generally be calling LookupZipToDropdown in the onClick of a button or the onChange of a Zip code text field, your will generally be able to pass this argument as this.form.((dropdownmenuname)), where the ((dropdownmenuname)) is the name attribute of the <select> tag.
3. If given, the third argument ("pEltStCd") is a form element reference to the associated form element for inputting state code (text box or dropdown menu, usually).
4. If given, the fourth argument ("pEltCntyCd") is a form element reference to the associated form element for inputting county code (hidden field, usually, since users don't usually know county codes so we hide that level of complexity from them).
5. If given, the fifth argument ("pEltCtyNm") is a form element reference to the associated form element for city name (almost always a text box).
6. If given, the sixth argument ("pEltStrNm") is a form element reference to the associated form element for street name (almost always a text box).
7. If given, the seventh argument ("pEltStrSfx") is a form element reference to the associated form element for street suffix (almost always a text box).

Arguments 3 – 7 are used only if exactly one row was found on the database for the given Zip/Zip+4. In that case, the generated code will select the one line and propagate its values through to the form fields named in the call. In addition, if pEltStrNm is given but pEltStrSfx is NOT given, the street name and street suffix are concatenated (with a space between them) and both are put into the pEltStrNm form element.

Another noteworthy feature is that the generated code will create "custom properties" in the option elements it generates. The standard properties that HTML option elements have are selected, text and value. But the options generated in pEltDropdown also have custom properties StCd, CntyCd, CtyNm, StrNm and StrSfx. These can be used in an onChange JavaScript to set form fields when the user selects a different item from the dropdown. For example:

```
<select name="StCdCtyNm" id="StCdCtyNm" onChange="
if  (this.selectedIndex >= 0)
    {
    var sOpt = this.options[this.selectedIndex];
    SetFormEltValue(this.form.StCd,        sOpt.StCd);
    SetFormEltValue(this.form.IMUserCtyNm, sOpt.CtyNm);
    If  ((sOpt.StrNm.length + sOpt.StrSfx.length) == 0)
        SetFormEltValue(this.form.IMUserStr1Txt,'');
    else
        SetFormEltValue(this.form.IMUserStr1Txt,
                        sOpt.StrNm + ' ' + sOpt.StrSfx);
    }
">
```

(This example uses SetFormEltValue because it's independent of form element type. See 4.6.20.)

4.4.2 dsp_LookupZipToDropdown.ajax.cfm

This routine was created in response to a need to do the same thing as dsp_LookupZipToDropdown, but where the calling page is not necessarily using SBA look-and-feel, or maybe not even ColdFusion. This callback must be on a ColdFusion server, of course, but the page that uses it can actually be a plain HTML file that doesn't use frames at all.

It uses a new technology called AJAX (Asynchronous JavaScript And XML). It uses all the same inputs and outputs as the non-AJAX version (dsp_LookupZipToDropdown.cfm), just discussed. In fact, it was written to be identical except for the underlying technology. There is only one mandatory difference in how you use it, namely, what JavaScripts you use:

Instead of:

```
<script src="/library/javascripts/LookupZipToDropdown.js"></script>
```

you must use:

```
<script src="/library/javascripts/ajax/GetXMLHttpRequest.js"></script>
<script src="/library/javascripts/ajax/LookupZipToDropdown.ajax.js"></script>
```

All of our AJAX implementations will have the restriction that GetXMLHttpRequest.js must be included. Note that they reside in the ajax subdirectory of /library/javascripts, and that ajax is redundantly added to the name of the AJAX JavaScript (to avoid accidentally deleting the non-AJAX version if they ever reside in the same directory someday).

This routine can be used synchronously or asynchronously. Typically you would use it synchronously when initially loading a page, particularly if there's more than one call. Otherwise, the second call would be made immediately, before the first call had a chance to finish.

```
<script>
<!--
function DoThisOnLoad()
{
gLookupZipToDropdownAsyncly           = false;
with (document.forms[0])
    {
    LookupZipToDropdown(MailAddrZip.value, MailAddrDropdown);
    LookupZipToDropdown(PhysAddrZip.value, PhysAddrDropdown);
    }
}
// -->
</script>
```

However, once the page is fully loaded, and the user makes a change to the zip code, it can be called asynchronously:

```
<input type="Text" name="MailAddrZip" onChange="
gLookupZipToDropdownAsyncly     = true;
LookupZipToDropdown(this.value, this.form.MailAddrDropdown);
">
```

The JavaScript variable gLookupZipToDropdownAsyncly is defaulted to true (asynchronously), so you only have to worry about it if you need to do a synchronous call.

4.4.3 dsp_LookupNAICSDescTxt.ajax.cfm

This routine was created in response to a specific need to return the NAICSDescTxt (description) associated with a given NAICSCd. It also uses AJAX. Unlike dsp_LookupZipToDropdown.ajax.cfm, however, it is always called synchronously. The reason is, the function itself actually returns the description.

Preparing to use it:

```
<script src="/library/javascripts/ajax/GetXMLHttpRequest.js"></script>
<script src="/library/javascripts/ajax/LookupNAICSDescTxt.ajax.js"></script>
```

Using it:

```
<input type="Text" name="NAICSCd" onChange="
if  (!EditMask('NAICS Code', this.value, '9', 1, 6, 6))
    {
    this.focus();
    return false;
    }
this.form.DescTxt.value = LookupNAICSDescTxt(this.value);
return true;
">
```

The LookupNAICSDescTxt JavaScript optionally takes a second argument, the NAICS year. If given and it's a 4 digit number, it will participate in the search for the description. Otherwise, the first description found will be used. If a NAICS code exists in multiple years, its descriptions will probably be the same, but not necessarily. To be absolutely sure you get a particular NAICS year's description, use the year argument:

```
<input type="Text" name="NAICSCd" onChange="
if  (!EditMask('NAICS Code', this.value, '9', 1, 6, 6))
    {
    this.focus();
    return false;
    }
this.form.DescTxt.value = LookupNAICSDescTxt(this.value, 2002);
return true;
">
```

4.4.4 get_ArrayUserRoles.cfm

Occasionally, a user who's logged into GLS on an SBA server has to be transferred to another server that's outside the SBA domain. An example is the 8(a)/SDB application. In this case, there isn't any hole in the firewall which would allow the external application to call Jaguar and authenticate the user.

Another situation is where the user authenticated using the Federal E-Authentication Initiative and the SBA has been mandated (by OMB and GSA) to trust that authentication.

In situations such as these, it may be necessary to do a callback to the server on which the user is logged in and retrieve "ArrayUserRoles" (the array of systems, roles and privileges to which the user is entitled to access). It complicates matters that our externally visible servers that require logging in are load-leveled using BIG-IP. So you don't necessarily know which server to do the callback to. This is why it was necessary to create get_ArrayUserRoles.cfm. It handles the problems caused by our having multiple servers with the same name.

To avoid conflicts with CFID and CFTOKEN on the remote server you're on, GLS may pass you these values as A and B on the URL, for example. These must be translated back into CFID and CFTOKEN on the call to get_ArrayUserRoles. Example code:

```
<cfinclude template="/library/udf/bld_ListToArrayUDFs.cfm">
...
<cfif TestProdInd IS "T">
    <cfset PrevServer     = "enile">
<cfelse>
    <cfset PrevServer     = "eweb">
</cfif>
<cfset PrevServer         = "https://#PrevServer#.sba.gov/library"
                          & "/callbacks/get_ArrayUserRoles.cfm"
                          & "?CFID=#URL.A#&CFTOKEN=#URL.B#">
<cfhttp method="GET" url="#PrevServer#" ... >
<cfset ListUserRoles      = Trim(cfhttp.fileContent)>
<cfif Left(ListUserRoles, 10) IS NOT "Call error">
    <cfset ArrayUserRoles = ListToArray2D
                          (ListUserRoles, Chr(10), Chr(9), "null")>
</cfif>
```

The initial cfinclude is to define the ListToArray2D function, which makes it easy to convert the output of the callback into an array. If the call did not succeed, the data returned will begin with "Call error", as shown in the final cfif. The other things you would usually do (cftry/cfcatch, performance logging, etc) are not shown, so as to keep the example brief.

4.4.5 get_GLSSession.cfm

In actual practice, the previous callback routine, get_ArrayUserRoles, has proven to be not all we needed it to be. For this reason, a newer and improved version exists called get_GLSSession.cfm. Unline get_ArrayUserRoles, it returns EVERYTHING associated with the user's GLS Session in WDDX format.

Usage is very similar to get_ArrayUserRoles (the major differences are highlighted in bold):

```
<cfif TestProdInd IS "T">
    <cfset PrevServer     = "enile">
<cfelse>
    <cfset PrevServer     = "eweb">
</cfif>
<cfset PrevServer         = "https://#PrevServer#.sba.gov/library"
                          & "/callbacks/get_GLSSession.cfm"
                          & "?CFID=#URL.A#&CFTOKEN=#URL.B#">
<cfhttp method="GET" url="#PrevServer#" ... >
<cfset XMLFromOtherServer = Trim(cfhttp.fileContent)>
<cfif Left(ListUserRoles, 10) IS NOT "Call error">
    <cfwddx action="WDDX2CFML"
        input="#Variables.XMLFromOtherServer#"
        output="Variables.GLS">
</cfif>
```

After doing this, Variables.GLS will be a structure that contains all of the Session scope variables that were created by GLS. Other Session scope variables, created for subsystems of GLS, such as Session.ProNet or Session.HubZone, are not needed on servers other than the ones on which they reside. So they will NOT be part of Variables.GLS. Only the Session scope data created by GLS will be in Variables.GLS.

You will normally never have to call this routine. Our new Distributed GLS technology will call it for you if your application resides on a server other than the login server. So this documentation is primarily for the benefit of those who will be maintaining the files associated with Distributed GLS.

4.4.6 Future Callbacks

Numerous server callbacks have been written for the ELend application. Over time, they will be generalized and made available to all applications in the /library/callbacks directory. When that happens, they will be documented here.

4.5 Standard CFIncludes

4.5.1 bld_ServerCachedQueries.cfm

Certain queries, especially those in the sbaref database, are common enough (and small enough) to be worth caching in memory. To make them available to all applications, not just those where the user must log in, we cache them in the Server scope.

To make sure that they're all defined before you try to reference them, do the following:

```
<cfinclude template="/library/cfincludes/bld_ServerCachedQueries.cfm">
```

This cfinclude uses the **public_sbaref** datasource, which, in turn, uses the **sbaselect** login to read sbaref. It won't work correctly if you include it within a cftransaction that uses any other datasource or login. Therefore, this cfinclude should be done near the top of your CFM file, among the initializations.

The cfinclude function ensures that the cached queries are defined. It doesn't actually retrieve them into the Variables scope, because it doesn't know which ones you need. That's why it was called a "build" routine (prefixed by "bld_"), not a "get" routine (prefixed by "get_").

At the time this document was created, the list of server cached queries was/is as follows:

Server.Scq.ActvBusAgeCdTbl
Server.Scq.ActvBusTypTbl
Server.Scq.ActvCalndrPrdTbl
Server.Scq.ActvCitznshipCdTbl
Server.Scq.ActvCohortCdTbl
Server.Scq.ActvEconDevObjctCdTbl
Server.Scq.ActvIMCntryCdTbl
Server.Scq.ActvIMCrdtScorSourcTblBus
Server.Scq.ActvIMCrdtScorSourcTblPer
Server.Scq.ActvIMEPCOperCdTbl
Server.Scq.ActvEthnicCdTbl
Server.Scq.ActvGndrTbl
Server.Scq.ActvGndrMFCdTbl
Server.Scq.ActvInjctnTypCdTbl
Server.Scq.ActvLoanCollatTypCdTbl
Server.Scq.ActvLoanCollatValSourcTbl
Server.Scq.ActvLoanCrdtUnavRsnCdTbl
Server.Scq.ActvLoanFinanclStmtSourcTbl
Server.Scq.ActvLoanMFDisastrTypTbl
Server.Scq.ActvLoanMFPrcsMthdTypTbl
Server.Scq.ActvLoanPartLendrTypTbl
Server.Scq.ActvLoanPckgSourcTypTbl
Server.Scq.ActvLoanPrevFinanStatTbl
Server.Scq.ActvLoanProcdTypTbl
Server.Scq.ActvLoanStatCdTbl
Server.Scq.ActvMfSubPrgrmCdTbl
Server.Scq.ActvPrcsMthdTbl
Server.Scq.ActvPrgrmTbl
Server.Scq.ActvPrgrmValidTbl

Server.Scq.ActvRaceCdTbl
Server.Scq.ActvSpcPurpsLoanTbl
Server.Scq.ActvStatCdTbl
Server.Scq.ActvStTbl
Server.Scq.ActvVetTbl

After calling bld_ServerCachedQueries, you can do the following to get the current list:

```
<cfoutput>#Variables.ScqAvailableResultSetNames#</cfoutput>
```

This list can vary on a server-by-server basis. For example, on a server that doesn't contain Electronic Lending, most of the queries with "Loan" or "PrcsMthd" in their names would be unnecessary. This capability is not currently being used, but it is likely that it eventually will.

The "Scq" part of the name is short for "Server-Cached Queries". It's the name of the structure containing the cached queries. The "Actv" part of the name comes from the SBA database standard abbreviation for "Active". It means that, at the time the query was built, it was filtered, so all rows in the query are active. That's why they don't contain "StrtDt" and "EndDt" (or "ActvInactInd") columns. But there may someday be a need to cache all rows of a definition table, even the inactive ones. When and if that ever occurs, additional queries can be defined, such as AllPrcsMthdTbl, and the different prefix will allow existing apps that use ActvPrcsMthdTbl to remain unaffected.

If ColdFusion Server crashes, and your page happens to be the first page that includes it, bld_ServerCachedQueries will actually do the database calls to build the server cached queries. To keep the exclusive lock of the Server scope as brief as possible, everything that CAN be done outside the lock IS done outside the lock. In this one (rare) situation, the variables will be defined in the Variables scope as well as the Server scope.

You yourself don't have to use the names given above (unless you want to show the origin of the query). For example, to make your code more readable, you could change the name in the Variables scope:

```
<cfinclude template="/library/cfincludes/bld_ServerCachedQueries.cfm">

...

<cflock scope="Server" type="ReadOnly" timeout="30">
    <cfset getStates       = Server.Scq.ActvStTbl>
</cflock>
<cfloop query="getStates">
    ...
</cfloop>
```

Codes and descriptions will be available in these queries as their actual database names (example, "StCd" and "StNm") and as "code" and "description". Use database names if you want to avoid conflicts with other queries. Use "code" and "description" for code sharing. For example, the standard cfinclude dsp_options.cfm (section 4.5.3) defaults to using "code" and "description" for this reason.

In the future, there may be other application-specific cached queries as well. These will likely be defined in separate structures, such as Server.ScqEDMIS or Server.ScqELend. This will avoid naming conflicts when the same database table name is used in 2 different databases.

4.5.2 dsp_errmsg.cfm

Two early standard variable names used in SBA ColdFusion files were Variables.ErrMsg and Variables.Commentary. ErrMsg would contain what went wrong (if anything) and Commentary would contain what went right (if anything).

The purpose of this cfinclude is to display ErrMsg and Commentary in a standardized way, namely, enclosed in an HTML box, in bold and brick red. Typically this is done by a display page in the AppData frame of SBA look-and-feel, just after the custom tag but before the page's actual contents:

```
<sbalookandfeel ... >
<cfinclude template="/library/cfincludes/dsp_errmsg.cfm">
<cfoutput>
... ((Your page here.)) ...
</cfoutput>
</sbalookandfeel>
```

This is described in much greater detail in 4.1.15, Form Data Recovery, including how to supplement automatic form data recovery with ErrMsg and Commentary values built during the execution of the action page (using put_sbalookandfeel_messages.cfm).

4.5.3 dsp_options.cfm

Given a query of codes and descriptions, plus some extra useful variable names, this cfinclude builds the <option> tags for a drop-down menu (<select>). Only the <option> tags are generated, allowing you to control the <select> and </select> lines by coding them manually. You can control how the <option> tags get generated using variables in the Variables scope that begin with DspOpts (to avoid conflicts with other variables defined in your CFM file):

DspOptsCodeName	Optional (cfparam'd)	"code"	
DspOptsDescName	Optional (cfparam'd)	"description"	
DspOptsQueryName	MANDATORY	no default	
DspOptsNotSelText	Optional (cfparam'd)	"Not Yet Selected"	
DspOptsSelList	Optional (cfparam'd)	""	(= don't use this feature)
DspOptsShowCode	Optional (cfparam'd)	"No"	(= don't use this feature)
DspOptsSkipList	Optional (cfparam'd)	""	(= don't use this feature)
DspOptsTabs	Optional (cfparam'd)	""	(= don't use this feature)

<u>DspOptsCodeName:</u> Column name of the query that contains the codes. The codes will go into the option tag's value attribute, causing them to be what's returned to the server when the form is submitted.

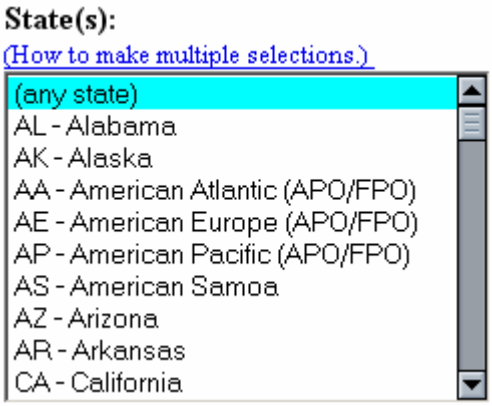
<u>DspOptsDescName:</u> Column name of the query that contains the descriptions. The descriptions will go between <option> and </option>, causing them to be visible to the user.

<u>DspOptsQueryName:</u> Name of the query. If you use a server cached query (described above, 4.5.1), do NOT give this as "Server.QueryName". To do so will cause implicit cflocking, and implicit cflocks are exclusive locks. Instead, do an explicit readonly cflock, copy the server cached query into the Variables scope and specify the Variables scope query name instead.

DspOptsNotSelText: This is what you want to have displayed as the very first <option value=""> option. Usually “Not Yet Selected” is fine, but sometimes the end user will insist upon “(all countries)”, as Trade Mission Online did, or “(any state)”, as PRO-Net did, for example. You can also set DspOptsNotSelText to “Suppress” if you don’t want an <option value=""> line generated.

DspOptsSelList: This is a comma-delimited list of codes that should have the “selected” attribute. If the drop-down allows the user to select multiple values (what JavaScript calls a “select-multiple”) it can have more than one value. Otherwise, if only one value can be selected, this list must NOT contain more than one element. For example, “DC,MD,VA” would be the DspOptsSelList for DC, Maryland and Virginia.

DspOptsShowCode: Sometimes, end users want the codes to appear with the descriptions, as in this example from the Dynamic Small Business Search (= “DSBS” = “PRO-Net”):



Setting DspOptsShowCode to “Yes” will cause dsp_options to prepend the code before the description, separated by a hyphen, as in shown above. Note: This feature does not affect the display order.

DspOptsSkipList: If you’re using a server cached query (see 4.5.1, above), you don’t have the ability to exclude certain rows from the result set using the where clause. This is a comma-delimited list to allow you to do so programmatically. In the DSBS example just given, you could exclude the APO/FPO lines by setting DspOptsSkipList to “AA,AE,AP”.

DspOptsTabs: If you like to keep your output HTML tidy, as if coded by hand, you may want to control the number of tabs before each line. This is not a number of tabs, but an actual string of tabs, which is more flexible. (It allows you to use spaces instead, if necessary.) So if you wanted to indent 3 tabs deep, you would set DspOptsTabs to Chr(9)&Chr(9)&Chr(9), or to RepeatString(Chr(9), 3), whichever you consider more readable.

In the future, there will be more server cached queries and more utility cfincludes to generate HTML from queries. For example, there could be dsp_checkboxes or dsp_radios routines for situations where those formats are more appropriate.

4.5.4 dsp_sbalookandfeel_variables.cfm
4.5.5 get_actual_server_name.cfm

The logical names of our servers are shared by multiple actual servers through a process called load leveling. Specifically, here are our current development, test and production Web server environments:

Development (danube.sba.gov):
- danube.sba.gov

Test (enile.sba.gov):
- rouge.sba.gov
- yukon.sba.gov

Production (eweb.sba.gov):
- riogrande.sba.gov
- wocs41.sba.gov

If you request a page on enile.sba.gov, the request might be handled by rouge.sba.gov or yukon.sba.gov. A load leveling hardware box, called BIG-IP, makes the decision as to which actual server to use.

Sometimes, however, you may need to know which actual server is servicing the request. To do this, do the following:

```
<cfinclude template="/library/cfincludes/get_actual_server_name.cfm">
```

After doing this, you will have 2 new SBA look-and-feel ("Slaf") variables defined:
- Request.SlafServerName for example, yukon
- Request.SlafLocalHost for example, yukon.sba.gov

The former is more useful in testing, because the names are shorter and don't include ".sba.gov". The second one is more useful in URLs, because you don't have to tack on ".sba.gov".

4.5.6 get_sbalookandfeel_variables.cfm
4.5.7 inc_starttickcount.cfm
4.5.8 inc_totaltickcount.cfm
4.5.9 OnRequestEnd.cfm

This include is related to logging. See 4.3.7.2.

In addition to doing end-of-request logging on normal completion of a page, it also turns off ColdFusion debug output unless specifically requested with Variables.Debug or URL.Debug being set to "Yes". The reason for this is, debug output is almost never needed on normal completion of a page. Failure to turn off debug output in development and test results in pages that cannot be adequately evaluated as to their appearance before releasing them to test. Debug output will still occur if an error happens or if requested.

4.5.10 put_sbalookandfeel_messages.cfm
4.5.11 put_sbalookandfeel_variables.cfm

4.6 Standard JavaScripts

We have a standard set of JavaScripts that are used for data validation and other uses on the browser. All were written to be cross-browser compatible, so that they can be used by the general public, who may have Netscape, Macs, etc.

4.6.1 Use onChange, Not onBlur

If you use onBlur to trigger client-side data validation, it's possible to get entangled in an infinite loop in which the user can't correct the form element because the JavaScript alert triggers another onBlur. The only way out of the infinite loop is to Control-Alt-Delete (PC) or Command-Opt-Escape (Mac) and kill the browser. In other words, onBlur can be so strict as to make it unusable with alerts. You need to loosen things up a bit so that the user can continue to work, entering data into other form elements.

Unfortunately, using onChange allows the user to tab out of the element after the error occurs. Because no change occurred, the onChange is not triggered at that time. This leaves the element containing the bad data that resulted in the error. It's possible that the user will never go back to the form element that's in error and correct it, which will result in a wasted hit on the server. That's where the next section (4.6.2) can help.

4.6.2 Code for Reuse in the Form's onSubmit

EditMask is the central routine for most of the other data validation routines. It returns true if the form element's value conforms to the edit mask, or it returns false if the value is in error (does not conform to the edit mask). So the traditional way to call EditMask used to be using exclamation mark ("!"), which means "not" in JavaScript, to detect an error and return the user to the same form element with the focus() method. Example (old way, do not use):

```
<input type="Text" name="SSN" ... onChange="
if  (!EditMask('Social Security',this.value,'999-99-9999',1,11,11))
    this.focus();
">
```

Translation: This form element is mandatory (the 1 before the two 11's). It must be a minimum of 11 characters long (the first 11) and a maximum of 11 characters long (the second 11). It must adhere to the edit mask pattern of "999-99-9999". If any of these characteristics is not true, pop up an error message in which the form element is referred to as "Social Security", and EditMask returns false. Because of the exclamation mark before EditMask, the if condition means "if EditMask fails", so this.focus() will be done, returning the browser to the field for the user to correct the mistake.

As mentioned above, the user can simply tab out of the form element at that point and leave bad data in the form element. Therefore, the preferred way to code is for reuse in the form's onSubmit trigger, so as to prevent the wasted hit on the server.

Code for Reuse in the Form's onSubmit, continued:

The following is an example of how you can allow the user to continue working (using onChange, not onBlur), but reuse the onChange in the form's onSubmit. This allows you to remind the user to change the element in error and avoid the wasted hit on the server:

```
<form ... onSubmit="
if  (!this.SSN.onchange())    return false;
if  (!this.ZipCd.onchange())  return false;
// etc.
return true;
">
...
<input type="Text" name="SSN" ... onChange="
if  (EditMask('Social Security',this.value,'999-99-9999',1,11,11))
    return true;
else
    {
    this.focus();
    return false;
    }
">
```

Note that, in this example, the exclamation mark is no longer needed, because both branches of the if are accounted for by means of the else condition. Instead, the onChange returns true if EditMask returned true, and it resets the focus and returns false if EditMask returned false. This is what allows it to be reused in the form's onSubmit trigger. Here's a more condensed version of the same technique:

```
<input type="Text" name="SSN" ... onChange="
if  (EditMask('Social Security',this.value,'999-99-9999',1,11,11))
    return true;// control will not continue, so no need for 'else'
this.focus();
return false;
">
```

Note that, in the onSubmit method, the JavaScript name of the onChange methods are in lower case. This spelling is required, because that's how JavaScript creates it. In the form element, the onChange is on the left hand side of the equals sign, which means that it's an HTML attribute. As with all HTML attributes, its spelling there is case insensitive. It doesn't matter whether you say

```
<input type="Text" name="SSN" ... onChange="
```

or

```
<input type="Text" name="SSN" ... OnChAnGe="
```

or

```
<input type="Text" name="SSN" ... ONCHANGE="
```

But when referencing the method defined by that attribute, you must always use lower case.

That's what the onSubmit of the form does. The most useful capability of the onSumbit is its ability to prevent the submission of the form by returning false. In this way, the onchange() method of the SSN, ZipCd and other form elements are reused, and the wasted hit on the server is avoided if any was previously left containing bad data by the user.

Most of the following are “Under construction.” Removing those comments to save paper printing:

4.6.3 EditDate

EditDate uses EditMask. If you don’t know how to use EditMask, you should read it first (below).

4.6.4 EditDateNonFuture

EditDateNonFuture uses EditDate. If you don’t know how to use EditDate, you should read it first (above).

4.6.5 EditList

EditList uses EditMask. If you don’t know how to use EditMask, you should read it first (below).

4.6.6 EditMask
4.6.7 EditPronetUserid
4.6.8 EditState

EditState is unique among our “Edit” JavaScripts, in that it validates an object reference to a form element, not a value. This is because it forces the state code the user entered to upper case. Example usage:

```
<input type="Text" name="State" ... onChange="
if  (EditState('State', this, 1))
    return true;// control will not continue, so no need for 'else'
this.focus();
return false;
">
```

In all of our other “Edit” routines, you would pass **this.value**, not **this**. But EditState will use the object reference to convert the input to upper case. (This conversion to upper case also decreases the number of state codes that EditState has to test for. For example, in the case of Texas, it only has to test for “TX”, not “TX”, “Tx”, “tX” or “tx”)

4.6.9 EditTin
4.6.10 ClearForm
4.6.11 DumpObject
4.6.12 FormSynopsis
4.6.13 GetXMLHttpRequest

This routine is required by all of our AJAX JavaScripts.

4.6.14 LookupNAICSDescTxt

This is your point of interface to the server callback that does NAICS code lookup. See 4.4.3, above.

4.6.15 LookupZipToDropdown

This is your point of interface to the server callback that does Zip code lookup. See 4.4.1 and 4.4.1, above.

4.6.16 NumToDollars

If the most recent call to EditMask was to validate a numeric value, EditMask will have set a global variable, called gValueInCents. Its name suggests its value. If the last valid numeric value was “5.23”, gValueInCents will contain 523. If the last valid numeric value was $12,345, it will contain 1234500.

This is a necessary evil in JavaScript, because all numbers with a decimal point are represented in double precision floating point. Sometimes, in floating point, 2 + 2 isn’t 4. Sometimes, 2.0 + 2.0 is 3.999999999, which confuses and upsets many of our users. To prevent these spurious representations and allow arithmetic, particularly in the case of money, we build gValueInCents to ALWAYS be an integer value. That’s because, in integer arithmetic, 2 + 2 is always 4.

Now suppose you want to calculate values for your users. For example, in a financial balance sheet, you may want to roll up assets to Total Assets, roll up liabilities to Total Liabilities and calculate Net Worth. You want to enter these values into the balance sheet, but you don’t want to enter them as integers. That’s where NumToDollars comes in. NumToDollars takes an integer in gValueInCents format and adds in commas and floating dollar sign. This is the format in which financial users like to see money amounts.

You don’t have to use it in roll-ups. You can even use it within a given field, so that the field’s value presents a nice, neat appearance. Users appreciate it, because the addition of commas helps them spot accidentally entering an extra zero, for example. Here’s an example of just such a usage:

```
<input type="Text" name="Salary" ... onChange="
if  (EditMask('Salary',this.value,'$',1,15,15))
    {
    this.value = NumToDollars(gValueInCents);// From EditMask
    return true;
    }
this.focus();
return false;
">
```

Note that, if the value fails EditMask validation, gValueInCents cannot be trusted. In fact, it will probably contain 0. So you should never trust gValueInCents unless EditMask returns true.

4.6.17 RoundTo2DecimalPlaces
4.6.18 RoundToNearest
4.6.19 RoundUpToNearest
4.6.20 SetFormEltValue

This JavaScript was written for cross-browser compatibility. Microsoft Internet Explorer (MSIE) allows you set a form element’s value in a way that violates the JavaScript Document Object Model (DOM). For example, suppose you have defined the following set of radio buttons:

```
Women-Owned Business?
```

```
<label><input type="Radio" name="WOB" value="Y"> Yes </label>
<label><input type="Radio" name="WOB" value="N"> No </label>
```

MSIE will let you say the following (WHICH VIOLATES THE DOM, SO DON'T DO THIS):

```
formname.WOB.value = "Y"; // only works in MSIE
```

MSIE treats this as equivalent to clicking on the Yes radio button, which has the value "Y". The right way to do this is to loop through the formname.WOB array and set the checked property. For example:

```
for (var i = 0; i < formname.WOB.length; i++)
    if  (formname.WOB[i].value == "Y")
        {
        formname.WOB[i].checked = true; // works in any browser
        break;
        }
```

Needless to say, it's a lot more work to do it correctly. To make it as easy in all browsers as it is in MSIE, we have the script SetFormEltValue. For example:

```
SetFormEltValue (formname.WOB, "Y");
```

4.7 Standard UDFs and Other Utilities

4.7.1 bld_GetCFDirectoryActionList.cfm

4.7.2 bld_GetCFFileActionRead.cfm

4.7.3 bld_JaguarUDFs.cfm

4.7.4 bld_ListToArrayAllowingNulls.cfm

4.7.5 bld_ProcessDirectory.cfm

4.7.6 val_char.cfm

4.7.7 val_date.cfm

4.7.8 val_email.cfm

4.7.9 val_num.cfm

4.7.10 val_phone.cfm

4.7.11 val_state.cfm

4.7.12 val_taxid.cfm

4.7.13 val_url.cfm

4.7.14 val_zip.cfm

5 Best Practices

The preceding were Coding Standards (mandatory). This section covers the best ways to get things done, but which are not at the mandatory level of Coding Standards.

5.1 Improving Performance

5.1.1 Eliminate Redundancies, Share Code

When you have a need for a new page that does pretty much the same thing as an older, existing page, the temptation is very strong to do a File > Save As … and modify the copy to do what you need the new page to do. You have to learn to resist that temptation.

Not only do multiple copies of redundant code take up disk space, they also take up too much of a far more precious resource, ColdFusion Page Cache space. When ColdFusion runs out of that space, it begins throwing away the compiled versions of pages to make room for the compiled versions of new pages. If this situation gets bad, the performance of the entire ColdFusion Server is adversely affected… Every application, even those that were designed cleanly and implemented efficiently, will slow down because one poorly designed, poorly implemented application is wasting the page cache.

If 2 pages do essentially the same thing, try to find a way to share that code they have in common, so that only one file is used, or so that a core set of common code is in a shared file. Then ColdFusion Server doesn't need to keep 2 versions of the same file in memory.

The following is an example of how to share the code of the graphics and text-only versions of the same file. Instead of doing a File > Save As on dsp_search.cfm to create a new dsp_search.textonly.cfm file that's just as big (but doesn't have any graphics), almost all code is shared in dsp_search.cfm:

File dsp_search.textonly.cfm:

```
<cfset Variables.TextOnly = "Yes">
<cfinclude template="dsp_search.cfm">
```

File dsp_search.cfm:

```
<cfparam name="Variables.TextOnly" default="No">
<cf_sbalookandfeel TextOnly="#Variables.TextOnly#" . . .>
. . .
<cfif Variables.TextOnly>
   <a href="dsp_search.cfm">Graphics</a> Version
<cfelse>
   <a href="dsp_search.textonly.cfm">Text Only</a> Version
</cfif>
. . .
</cf_sbalookandfeel>
```

This example assumes you have some reason not to use 4.1.14 Automatic Text Only and Screen Resizing, above. For example, if you aren't using SessionManagement, this technique is ideal.

5.1.2 Eliminate Redundancies, Share Code, part 2

Another example of eliminating redundancy and sharing code is to use the actual standard libraries, rather than your own copy of them. Not long after the Stored Procedure Call File Generator was created to generate SPC files for use with CFINCLUDE, other programmers created their own versions of the generator that generated custom tags or ColdFusion Components (CFCs). Not only was this a waste of effort and disk space, it was also an enormous waste of ColdFusion Page Cache.

All of those systems will have to be reprogrammed to use SPC files to be in compliance with the Shared Code chapter of this Standards document. Be mindful that what is not a standard right now might become a standard in the future.

Even when you have a need for an application-specific version of an otherwise standard routine, share it. Don't make multiple copies of it and use it in several different locations. Not only does that make it hard to update all the copies when you need to make a change, it also wastes the page cache.

5.1.3 Limit Record Set Size

To prevent server performance degradation due to large searches, limit the number of rows returned in a record set to a reasonable number as determined by the management of your application. In a CFQUERY, this is accomplished by the MAXROWS attribute.

An alternative, where the search criteria are based on user inputs, is to "SELECT COUNT(*)" with the same FROM and WHERE clauses as the intended search. Then, if the count doesn't exceed the management determined maximum, perform the intended search. This method can be used to give the user an idea of how unrestricted the search would have been by displaying the count. This encourages a more restrictive search. (Example limits: The ColdFusion version of PRO-Net restricts searches to 1500 rows. In previous versions, it was 5,000, unless the searcher is at the SBA, in which case the limit was 25,000.)

Where the result set is inherently limited by there being only a small amount of data (such as getting all state codes), there is no requirement to limit the search.

5.1.4 Caching Result Sets

If a search engine pages its results (First 25 Matches, Next 25 Matches, etc), the same SQL is going to be resubmitted over and over again. That's a situation in which it makes sense to cache result sets with the CACHEDWITHIN attribute of CFQUERY.

As compared to caching the result set in the session scope, CACHEDWITHIN offers a finer level of control. Even though the session won't time out for an hour, you can timeout the cached result set at 5 or 10 minutes, for example.

5.1.5 Explicitly Scope Variables

In general, you should give the explicit scope of the variable you're defining or using. That is, say **Variables.GLSAuthorized** or **Session.GLSAuthorized**, not simply **GLSAuthorized**. This prevents ColdFusion from searching every scope until it finds the definition (for better performance) and makes the context explicit (for more reliable and predictable behavior).

5.2 Code for Ease of Maintenance

5.2.1 Parameterize Directory Names and Paths

As described above, all of our Shared Code directories are in the same location on all SBA ColdFusion Servers. But hard coding those paths makes it difficult to respond to changing maintenance needs. Instead, you can ease your maintenance chores by parameterizing directory names and paths, typically in Application.cfm. The following example has "Variables." removed to fit on this page better:

```
<cfset AppDir            = "/myapp">
<cfset AppImagesDir      = AppDir           & "/images">
<cfset AppIncludesDir    = AppDir           & "/cfincludes">
<cfset AppJavascriptDir  = AppDir           & "/javascripts">
<cfset AppMainNavJSDir   = AppJavascriptDir & "/sbalookandfeel">
<cfset AppSearchDir      = AppDir           & "/public">
<cfset AppUpdateDir      = AppDir           & "/sbaonly">
<cfset LibDir            = "/library">
<cfset LibImagesDir      = LibDir           & "/images">
<cfset LibIncludesDir    = LibDir           & "/cfincludes">
<cfset LibJavascriptDir  = LibDir           & "/javascripts">
<cfset LoginDir          = "/gls">
<cfset LoginURL          = LoginDir         & "/dsp_login.cfm">
```

To be effective, these parameterized directory names and paths have to be used everywhere, of course:

```
<img src="#AppImagesDir#/welcome.gif" ... >
<cflocation url="#LoginURL#">
<script src="#SysJavascriptDir#/EditMask.js"></script>
<cf_sbalookandfeel MainNavJSURL = "#AppMainNavJSDir#" ... >
```

<u>Two examples of how this can greatly ease maintenance:</u>

At one point in time, Electronic Lending was called "LMS". Its application directory (AppDir in the example above) was /lms. Later, it was decided that many other SBA applications could benefit from the LMS login screens and role management. So the General Login System (GLS) was split out of LMS and given its own directory, /gls. Because LMS had been written using the technique above (not using the same names, but the same technique), all that was necessary to adapt all GLS code to its new top level directory was to change the parameter for the application (AppDir in the example above).

For various reasons, the name of Electronic Lending itself kept changing. At one point it was called "EAM" (the "Electronic Application Module"), "LOS" (the "Loan Origination System") and eventually, ELend. But just as before, when the top level directory name was changed to /elend, all that was necessary was to change one parameter, and all code automatically adapted to the change.

Although there are no plans to move /library and other shared code, coding in this manner allows us to do so, should the need ever arise.

5.2.2 Indent Properly

Even though we ourselves wrote a passage of code, we all tend to forget exactly what it does while we're working on other projects. When we return, we have to figure out our own code as if it were written by someone else. We will have to return to our old code, because the last person who worked on it is the first one called when there's a problem or a modification request. So, although you may THINK that tidying up your code is just to make things easy on "the next guy", keep your code readable to make things easy on YOURSELF.

Indenting your code properly allows you to visually see the scope of a <cfif> or <cftransaction> during maintenance and allows you to see where new code belongs.

To minimize horizontal scrolling in text editors, note that you don't always have to indent. For example, <cfcase> has no meaning except within a <cfswitch>. So the scope of the <cfswitch> is very obvious in the following, even though the <cfcase> statements aren't indented:

```
<cfif IsDefined("Form.FieldNames")>
    <cfswitch expression="Form.StCd">
    <cfcase value="DC,MD,VA"><cfset Local = "Yes"></cfcase>
    <cfdefaultcase>          <cfset Local = "No"> </cfdefaultcase>
    </cfswitch>
    <cfif Local>
        ...
    </cfif>
</cfif>
```

The goal in indenting code is to make it easy to determine the scope of paired tags. To the extent that paired tag scope remains obvious, your indenting is good.

5.2.3 Line Up Code to Make It Easier to Read and Spot Errors

In the parameterized directory names of section 5.2.1, note that the "=" and "&" signs are lined up. That was on purpose. Compare this:

```
<cfset AppImagesDir = AppDir & "/images">
<cfset AppIncludesDir = AppDiv & "/cfincludes">
<cfset AppJavascriptDir = AppDir & "/javascripts">
```

to this:

```
<cfset AppImagesDir      = AppDir & "/images">
<cfset AppIncludesDir    = AppDiv & "/cfincludes">
<cfset AppJavascriptDir  = AppDir & "/javascripts">
```

In which example is it easier to see where AppDir is misspelled? If AppDiv is also defined (so that you don't get a crash), locating the error may take time. Lining up code helps.

5.2.4 Define Configuration Parameters at Top of Page

Rather than scouring a file to ascertain where a configuration parameter is being set, get into the habit of putting all configuration parameters at top of page, just below the mandatory program description (comment header).

They don't have to all be simple <cfset> tags. Sometimes they can be calculated, as in the case of MaxRows or UploadDir in the following example. But if they're configuration parameters, they belong up top:

```
<!---
AUTHOR:           Marco Ortiz
DATE:             08/26/2001
... (rest of comment header, as described in section 3.3.5.)
--->

<!--- Configuration Parameters --->

<cfset CacheTimeSpan = CreateTimeSpan(0,0,5,0)><!--- 5 minutes --->
<cfif Left(CGI.Remote_Addr, 8) IS "165.110."><!--- SBA User --->
    <cfset MaxRows   = 25000><!--- We get to see more rows. --->
<cfelse>
    <cfset MaxRows   = 1500>
</cfif>
<cfset RowsPerPage   = 25>
<cfset UploadDir     = Variables.AppUploadDir & "/xmlfiles">
<cfset UploadMaxSize = 1000000><!--- bytes --->
```

5.2.5 Make LOTS of Things Configurable

Anything you keep repeating in the code should probably be a configuration parameter. For example, suppose that, to avoid being right up against the left edge of the AppNav region in SBA look-and-feel, you've coded a lot of non-breaking spaces ( ) to indent your hotlinks, like so:

```
    <a ...>Print Application</a><br>
    <a ...>Credit Report</a><br> ... (etc)
```

That's tedious to do and tedious to change. It's much easier to define a configuration parameter:

```
<cfset AppNavIndent = "    ">
```

and use the parameter in your code instead, allowing easy change of indention:

```
#AppNavIndent#<a ...>Print Application</a><br>
#AppNavIndent#<a ...>Credit Report</a><br> ... (etc)
```

5.2.6 Document Your Code: Use "Hints"

The hint attribute of CFFUNCTION and CFARGUMENT tags can be retrieved by the user of the component / Web Service. So you can use hint attribute to document a function and its arguments to the users. This gets you out of the business of writing interface documentation. Using hints makes documentation automatically available to those who need it most.

5.2.7 Document Your Code: Use Comments for Actual Comments

You don't have to put your initials and the date in a comment on every line you change. If the comment header says that you added special code for tax-exempt Native American owned businesses, and there's only one place in the file that has tax-exempt Native American owned business coding, that pretty much speaks for itself.

Comments make code more maintainable when they impart knowledge about what's going on, as in the following example:

```
<!---
If we only scan through the XML and apply validation rules to the
XML fields we see, we won't detect missing mandatory fields. So
another pass is required, in which we scan through the validation
rules looking for mandatory fields for which there is no XML field.
In the process, set missing optional fields to nullstring, so that
they'll be defined when we call the SPC files:
--->
```

Without even seeing the code that follows, you already know what it's trying to do. This is the best kind of comment: the kind that contains an actual comment, the kind that makes code easy to maintain.

5.2.8 Document Your Code: Use Descriptive Datanames

Under construction.

5.3 The "Right Way To Do It"

The following are not just Best Practices. They are what works. They are included here to save all SBA ColdFusion developers the slow learning curve that results from trial and error.

5.3.1 Use CFLOCK to Lock Server, Application, and Session Variables

ColdFusion Server is a multi-threaded web application server that can process multiple page requests at any given time. Use CFLOCK to guarantee that multiple concurrently executing requests do not manipulate these shared memory scopes in an inconsistent manner.

```
<cflock Scope="Application" TIMEOUT="30" TYPE="Exclusive">
   <cfparam name="application.number" default="”1”>
</cflock>
```

If you don't code a CFLOCK, newer levels of ColdFusion will implicitly lock that scope for you, but the default lock is Exclusive. Coding your own CFLOCK allows you to use the less restrictive ReadOnly lock.

Even when an Exclusive lock might be necessary, you may be able to avoid an Exclusive lock in a majority of cases. See section 3.1 "Session Control (CF 4.x and 5.x)" for an example of how to use a ReadOnly lock to determine whether an Exclusive lock is necessary.

5.3.2 Structured Query Language (SQL) in JDBC

Because CFMX uses JDBC, you have to use single quotes to delimit non-numeric literals in SQL. Also in JDBC, NULL is not a value. So you have to say "TaxId is null", not "TaxId = null". Similarly, NULL will not be found in IN or NOT IN clauses, which are implicit equality checks. So you have to say "StCd is null or StCd not in ('DC','VA','MD')".

The list of JDBC differences goes on and on. See 7.3 ColdFusion MX 6.x (and Conversion to ColdFusion MX in General), below, for a more detailed list and workarounds.

5.3.3 Checking for Existence of CGI Variables

Don't say <CFIF IsDefined("CGI.xxxx")>. To keep code from crashing on different Web servers that support different CGI environment variables, CGI.variablename is always defined, regardless of what "variablename" is. So instead of IsDefined(), check whether its length is greater than 0: <CFIF Len(CGI.xxxx) GT 0>. (IsDefined doesn't hurt anything. It just doesn't give you what you wanted in this case.)

5.3.4 How to Break Out of Frames

Frames manipulation can ONLY be done on the browser. If a hotlink loads a page in a given frame, there's nothing the page can do in CFML to change that fact. The page WILL be loaded into that frame, no matter what you do in CFML. Therefore, if you want to break out of frames, you must use HTML or JavaScript to do so.

5.3.4.1 Breaking Out using HTML

Before the user ever requests the page (presses the hotlink or submits the form), the HTML can specify the TARGET attribute to tell the browser where to load the page. The following target values are especially useful:

```
<a target="_top"   ... > or <form target="_top"   ... >
<a target="_blank" ... > or <form target="_blank" ... >
```

The first one breaks out of frames by loading the page all-by-itself in the current window. The second one breaks out of frames by loading the page all-by-itself in a new window.

5.3.4.2 Breaking Out using JavaScript

The target attribute value can also be a frame or window name, so specifying the target attribute has many other uses. It can be used by hotlinks in AppNav to load pages into AppData, for example, as follows:

The following, in AppNav:

```
<a target="AppData" href="dsp_feedback.html">Feedback</a>
```

has the same effect as the following, in AppData:

```
<a href="dsp_feedback.html">Feedback</a>
```

In either case, the browser has been instructed to load dsp_feedback.cfm into the AppData frame. There is absolutely nothing that dsp_feedback.cfm can do (on the server) to keep this from happening. But once the page is on the browser, dsp_feedback.cfm can include the following JavaScript to break out of the AppData frame, so that it is all-by-itself in the same window:

```
<script language="JavaScript">
<!--

if  (self != top.self)
    top.location.href = self.location.href;

// -->
</script>
```

5.3.4.3 Always Use the HTML Method Wherever Possible

If you know in advance that a page needs to be outside of frames, ALWAYS use target="_top" in the HTML, even if that page has JavaScript to break out of frames. The reason is, Microsoft Internet Explorer users will not be able to return using the Back button if JavaScript broke the page out of a frame. Instead, MSIE returns to the frame, the page re-executes the JavaScript, and the user will be tossed back into the same page again. To the user, it seems that the page broke the Back button (and it did). Use target="_top".

5.3.5 How to Change the "RequestTimeout" of a Page

The maximum time for the execution of a page is called the "RequestTimeout" value, because that's the actual name used to set the time. The mechanism has changed from version to version of ColdFusion. The SBA is currently running versions CF 4.5, 6.1 and 7.0 on various ColdFusion servers.

Here are the rules for setting RequestTimeout:

Under CF 4.0, 4.5 and 5.0, you pass a URL variable RequestTimeout to the page. So if you want to give dsp_mandlebrot.cfm 5 minutes (600 seconds) to run, you would use the URL:

http://server/path/dsp_mandelbrot.cfm?RequestTimeout=600

The problem with this method is that it's already too late to increase the timeout value once you're inside the page. For this reason, Macromedia created 2 new ways to increase the timeout under ColdFusion MX:

<cfset Variables.RequestTimeout = "600">

or

<cfsetting RequestTimeout = "600">

Both of these allowed changing the timeout from within the page itself. The first method works under CFMX 6.0 and 6.1, but not under 7.0. The second method doesn't work under 6.0, but works under 6.1, 7.0 and is the approved way to do it for all future versions.

You can use the Server.ColdFusion.ProductVersion list to determine what version of CF is running and set the timeout value accordingly. The first element of the list is the major product version. So if you want RequestTimeout on the URL to work under all levels, the following example shows how that can be done:

```
<cfif IsDefined("URL.RequestTimeout")>
   <cflock scope="Server" type="ReadOnly" timeout="30">
      <cfset VersionList = Server.ColdFusion.ProductVersion>
   </cflock>
   <cfswitch expression="#ListGetAt(VersionList,1)#">
   <cfcase value="1,2,3,4,5">
      <!--- Do nothing, because RequestTimeout was set by URL. --->
   </cfcase>
   <cfcase value="6">
      <cfset Variables.RequestTimeout = URL.RequestTimeout>
   </cfcase>
   <cfdefaultcase><!--- 7 or greater --->
      <cfsetting RequestTimeout       = URL.RequestTimeout>
   </cfdefaultcase>
   </cfswitch>
</cfif>
```

There is now a custom tag <cf_SetRequestTimeout seconds="((seconds))"> to set the RequestTimeout for you. The example above simply serves to show how it works under a variety of ColdFusion versions.

In most cases, you don't have to set RequestTimeout from URL.RequestTimeout, by the way. If you cfinclude get_sbalookandfeel_variables.cfm or get_sbashared_variables.cfm, and URL.RequestTimeout is defined, a call will be generated for you: <cf_SetRequestTimeout seconds="#URL.RequestTimeout#">

5.3.6 Dynamic HTML

5.3.6.1 General Principles and Section 508 Issues

DHTML requires that JavaScript must be active. It may not be. Some users are paranoid about JavaScript and turn it off. Some users are annoyed at advertizing pop-up windows and turn it off because their browser doesn't support pop-up blocking. Some users are blind and using a Web page reader that gets messed up by JavaScript. So although 95% – 98% of all users have JavaScript on these days, that means that there are 5% - 2% who don't. So your pages can't become totally non-functional if JavaScript is off.

There are other SBA documents which specify how to code for Section 508 compliance. That's "out of scope" for this document. This section is about how to use DHTML in a way that doesn't conflict with Section 508 rules.

5.3.6.2 Left-Side Navigation Trees

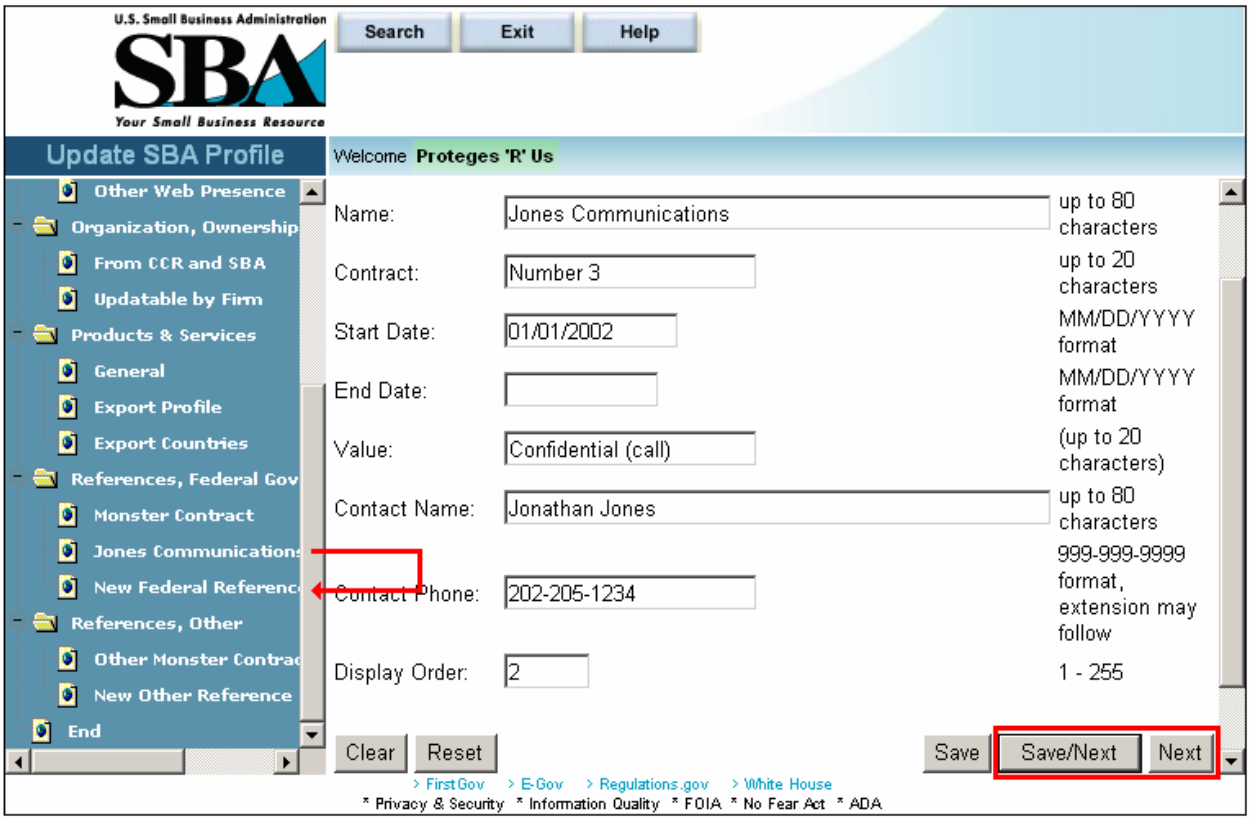
If you use the left-side navigation tree (<cf_sbatree> and <cf_sbatreeitem>), you are intrinsically using DHTML. It's a DHTML tree. If JavaScript is off, it won't work. Therefore, if you use the left-side navigation tree, or if your hotlinks in AppNav use JavaScript, you must provide an alternate way to navigate through all of your pages.

The Best Practice for this is to implement "Next" and "Save/Next" logic in your pages. This is how it's done in Electronic Lending (ELend), the SBA Supplemental Pages of PRO-Net and many other SBA applications.

The first part of doing this is in how you organize your left-side navigation tree. If there's a repeating group that's variable in size, put the repeating group in a folder. This allows it to be collapsed when the user doesn't care about the group. More importantly, put the "New" page (the page that allows the user to add a new member to the repeating group) at the end of the repeating group. This isn't an ease of use issue or a Section 508 issue. It's a consistent behavior at SBA sites issue. All of our applications add on to repeating groups at the end of the group.

The user enters at a particular page of a multi-page. Which page is up to you. Generally it's the first page, but there might be a better page that most users would prefer to go to first. What's important is, the user doesn't have to use left-side navigation to get to the first page.

From any given page that the user is seeing, “Next” and “Save/Next” should (by default) take the user to the next page in sequence in the navigation tree.



They aren’t using the navigation tree to get to the next page, but the navigation tree still functions as a guide. That is, it’s your documentation as to which page you mean when you say “Next”. This is so intuitive, it generally requires no explanation. Users will generally adapt to this intuitive behavior without even realizing it.

Note that you go to the next page “by default”. It may be necessary to return to the same page if the user enters bad data. In the example above, if the user entered “two” instead of “2” as the Display Order, it would be impossible to save the data as entered. In that case, you would return to the same page with an error message, to give the user the opportunity to correct the mistake.

Last but not least, when you’re on the last page of the navigation tree, “Next” and “Save/Next” take you back up to the top of the navigation tree. This is what makes the set of pages Section 508 compliant. It means, no matter where you start in the navigation tree, you can always get to every page (eventually) by repeatedly hitting “Next” and/or “Save/Next”.

TESTING:

- Start at the default first page and begin pressing the “Next” button repeatedly.
- Verify that you always get to the next page in the navigation tree, never skipping a page, never staying on the same page.
- Verify that, at the bottom page of the navigation tree, “Next” takes you to the top page of the tree.
- When you reach the page where you started, you’ve proven that you don’t need JavaScript to get to every page.

5.3.6.3 A Trick to Make Sure the Left-Side Navigation Tree Matches "Save/Next"

One way to make sure that your Left-Side Navigation Tree matches your "Next" and "Save/Next" logic is to generate both from a common source. Here's a made-up example:

```
<cfset PageOrder = "Welcome,CompanyInfo,AffiliateFolder,"
                 & "Affiliate1,Affiliate2,AffiliateNew,ContactInfo,"
                 & "PrincipalFolder,Principal1,PrincipalNew,Done">
```

In this example, you've established a naming convention that, if the name ends in "Folder", you're going to generate a folder in the left-side navigation tree. After a folder, every item that begins with the same thing as the folder will be a subitem of the folder. So Affiliate1, Affiliate2 and AffiliateNew will be subitems of the Affiliate folder, but ContactInfo doesn't begin with "Affiliate", so it moves up a level in the navigation tree.

In the "Next" and "Save/Next" logic, you simply do a ListFind on the current page, add one to its value and that's the next page in sequence, the page you go to by default.

Of course, you can't always do this. In Electronic Lending (ELend), for example, any principal can be an owner of any borrower, guarantor or affiliate. The structure of the navigation tree gets quite complex and can't be reduced to a simple ColdFusion list, as in this example. But if you CAN represent your navigation tree as a list, it's certainly a Best Practice to do so, as it greatly simplifies matching up the tree to your Save/Next logic.

5.3.6.4 Putting Data Elsewhere on a Page

As long as data gets onto a page (everywhere it's needed) at the time the time it's loaded, it's Section 508 compliant. That is, data cannot be put somewhere ONLY by JavaScript. As long as the user has the option to save the current page and return to it at some later time, the distribution of data to somewhere else on the page is not being denied to the user. If the user DOES have JavaScript on, however, they just get to see it a little sooner, that's all.

All of the browsers you're required to support now implement the document.getElementById method and the innerHTML property of the object. So that way of putting data elsewhere on a page is always available to you. But it's not the simplest way to do things, not by a long shot. The Best Practice is, always put data elsewhere by the simplest available method that works in all supported browsers.

EASIEST (if you're in a form), putting data into form elements:

- **Text and textarea:** Use this.form.ElementName.value from a different form element in the same form, or this.ElementName.value from the form's onSubmit handler. In the following, to save space, ((form)) will be used to mean "this.form in an element, or this in a form's onSubmit".
- **Radio buttons:** Use ((form)).ElementName[number].checked = true/false, where number is the 0-based array index of the radio button you want to check. Radio button groups always have the same element name, so they are represented by arrays. Microsoft Internet Explorer lets you set ((form)).ElementName.value = something, and it will find the radio button that contains that value and check it. But ONLY Internet Explorer allows this. So don't do it that way. Use the checked attribute instead.
- **Checkboxes:** Use ((form)).ElementName.checked = true/false if there's only one checkbox with that element name. It's also possible to define multiple checkboxes with the same name, in which case, use the same syntax as for radio buttons, above.

- **Single Value Dropdown Menus:** If a dropdown menus ("select list") does not contain the attribute "multiple", it can have only one value. In that case, use the syntax ((form)).ElementName.options[number].selected = true to set its value, where number is the 0-based array index of the <option> tag you want to be selected. As with radio buttons, Microsoft Internet Explorer lets you set ((form)).ElementName.value = something, and it will find the <option> that contains that value and select it. But ONLY Internet Explorer allows this. So don't do it that way. Use the selected attribute of the option you want to select instead.
- **Multiple Value Dropdown Menus:** If a dropdown menu DOES contain the "multiple" attribute, you have no choice. You must use ((form)).ElementName.options[number].selected = true/false syntax for each <option> you want to select or deselect. Note that, with single value dropdown menus, you never have to set any selected property to false. All you have to do is set the one you want to set to true, and any other option that's selected will automatically be deselected. But in a multiple value dropdown menu, you have to explicitly deselect options by setting their selected property to false.

SIX-OF-ONE, HALF-A-DOZEN-OF-THE-OTHER (but only if you're in a form):

- **ReadOnly Text Displays:** It's considerably easier to put data into a readonly or disabled text field (syntax above) than it is to use document.getElementById and innerHTML. But it can create the impression that the user could get to the data and change it under some conditions. In addition, a clever user who knows HTML can save the page to their hard drive, edit it and make the field updatable. Conceivably, if the action page retrieves the values of readonly fields, it would allow such a hacker to succeed at editing the field that you wanted to be readonly.

SAFEST WAY (and the only way if the target destination is not in a form):

- **ReadOnly Text Displays:** It's not that hard to use ((ref)) = document.getElementById("idname") to retrieve an HTML object reference to the HTML element that contains the matching id="idname" attribute. Of course, this requires that the value of the id attribute ("idname" in this example) must be unique in the Web page or frame. (Each frame has its own document, so the getElementById is restricted to the document you're searching.) If the call to getElementById returns a value other than the JavaScript keyword null, it will be an object reference. You can then use ((ref)).innerHTML = "data you want to display" to display readonly data, presumably in a box with class="viewdata" to mark it as a readonly data display.

Lately there's been a tendancy for some folks to use the getElementById and innerHTML method in all circumstances, as if the form element techniques have been deprecated or something. That's NOT the case. Form element techniques are still, by far, the most widely supported ways of putting data elsewhere on the screen. They are NOT going away in future version of HTML. You SHOULD use the older, simpler form element techniques, particularly where the form element values are not readonly.

The next section "How to Show and Hide Page Elements Dynamically" will go into greater detail on how to use getElementById.

5.3.6.5 How to Show and Hide Page Elements Dynamically

All of the 5 browsers and all versions listed in 3.5.1 (Browser Support) now allow showing and hiding elements of a Web page with the following cross-browser-compatible techniques:

- The id attribute (, <p id="…">, <td id="…">,<tr id="…">, etc).
- document.getElementById("…") to retrieve the page element by its id attribute.
- CSS: style="display:none" to hide the element in its initial state.
- JavaScript: ((element)).style.display = "none"; to hide the element dynamically (based on changing situations on the page).

These techniques affect ONLY THE USER'S AWARENESS of the page element. That is, it is simply not rendered on the screen and not read aloud in Web page readers for the blind. The space normally taken up by the page element disappears, so visually a page appears to expand or collapse, or in a Web page reader the next page element in sequence is immediately read.

But the page element itself is still on the page. If it contains form elements, those form elements are still a part of the form, and if they would normally be sent based on their state (a checked checkbox, for example), then they'll be sent when the form is submitted.

The following example shows Initial Interest Rate only if the Interest Type is Fixed, and shows Spread over WSJ Prime if Interest Type is Variable. Initially, neither is displayed:

```
<tr>
    <td class="MandLabel">Interest Type:</td>
    <td class="MandData" nowrap>
        <input type="Radio" name="FixVarInd" value="F" onClick="
        document.getElementById('RowIIR').style.display = 'block';
        document.getElementById('RowWSJ').style.display = 'none';
        "> Fixed
        <input type="Radio" name="FixVarInd" value="V" onClick="
        document.getElementById('RowIIR').style.display = 'none';
        document.getElementById('RowWSJ').style.display = 'block';
        "> Variable
    </td>
</tr>
<tr id="RowIIR" style="display:none;">
    <td class="MandLabel">Initial Interest Rate:</td>
    <td class="MandData">
        <input type="Text" name="IIR" ...>
    </td>
</tr>
<tr id="RowWSJ" style="display:none;">
    <td class="MandLabel">Spread over WSJ Prime:</td>
    <td class="MandData">
        <input type="Text" name="WSJ" ...>
    </td>
</tr>
```

Note that the onClick JavaScripts use single quotes to delimit id names, ‘none’ and ‘block’, because the HTML is using double quotes to delimit the JavaScripts. Escaping double quotes using the JavaScript convention of \" doesn’t work in this situation, because the onClick is delimited by HTML double quotes and HTML doesn’t understand \".

Also note that it’s getElementById, not getElementByID (a common mistake).

If you’re already using SBA look-and-feel (as required by this standards document), you already have another example on your pages. The “10-pixel white margin” regions of the page (that is, the white space outside the 1-pixel black border) is now a page control that maximizes the AppData region of the page. You can see this by mousing over any margin and seeing the tooltip “Maximize data entry region of screen”:

Clicking on it hides the regions called SBALogo, MainNav, AppName, AppInfo, AppNav and BotMost, then resizes AppData to take advantage of the added space (and changes to tooltip to say “Minimize”):

In an example of coding for flexibility, the code that does this does not assume that the style.display will always be either “none” or “block”. It simply saves the initial value of style.display (for use when restoring the region to visibility), like so:

```
var gRowSBALogo;
var gRowSBALogoSaveDisplay;
...
function InitDynamicContent ()
{
...
gDivSBALogo            = document.getElementById("DivSBALogo");
gDivSBALogoSaveDisplay = gDivSBALogo.style.display;
...
```

You might want to do the same sort of thing to keep your own show-and-hide code flexible.

5.3.6.6 How to Change Page Element Classes Dynamically

SBA look-and-feel calls for mandatory form elements to be labeled with class="mandlabel" and for the form element to be encased in a nested table with class="manddata". For example:

```
<tr>
    <td id="LabelEIN" class="mandlabel">
        <label for="EIN">EIN:</label>
    </td>
    <td>
        <table id="DataEIN" border="1" class="manddata"><tr><td>
            <input type="Text" id="EIN" name="EIN"
                value="" size="10" maxlength="10">
        </td></tr></table>
    </td>
</tr>
```

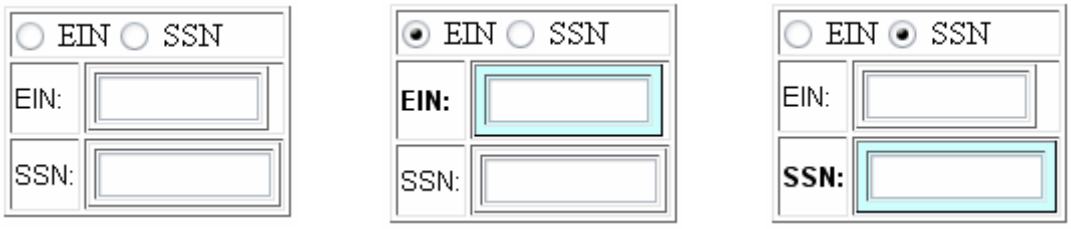
Although you're not required to place a nested table around optional form elements which use classes "optlabel" and "optdata", to doin so poses as an interesting alternative to the conventional technique. For example, you could define an optional field with a nested table:

```
<tr>
    <td id="LabelEIN" class="optlabel">
        <label for="EIN">EIN:</label>
    </td>
    <td>
        <table id="DataEIN" border="1" class="optdata"><tr><td>
            <input type="Text" id="EIN" name="EIN"
                value="" size="10" maxlength="10">
        </td></tr></table>
    </td>
</tr>
```

This allows dynamically changing it to mandatory using JavaScript and className (not just class):

```
document.getElementById('LabelEIN').className  = 'mandlabel';
document.getElementById('DataEIN').className   = 'manddata';
```

A brief example page is available online at http://danube.sba.gov/testlookandfeel/ChangingClasses.html It uses radio buttons and toggles which fields are mandatory according to which radio button you click, via the onclick event handler. Screen snapshots are from Opera 8.5.1 for the PC, showing that this technique is also cross-browser compatible:



5.3.7 How to Create an HTML Equivalent of a Graphic for TextOnly Mode

Using the Paint program and Calculator, graphics which reduce page loading speed may be replaced with an HTML equivalent that contains identical colors and the original graphic.

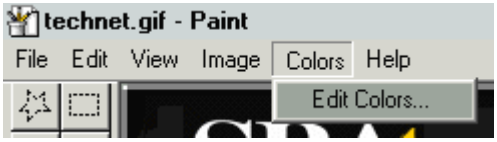
The following is an image for which we want to have an HTML equivalent. (If you're reading a paper copy of this document printed on a black and white printer, you can see this same image in color at http://tech-net.sba.gov/index.cfm). As you can see, we start out with it opened in the Paint accessory:

Steps to create an HTML equivalent of a graphic:

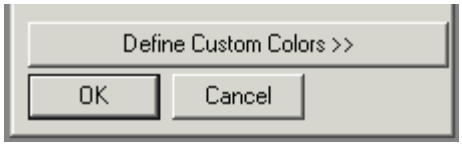
- 1. Select the eyedropper tool, which has the tool tip "Pick Color". See the circle labeled "1" (upper left).
- 2. Click on any color that you want to know the RGB (red, green, blue) values for. See the circle labeled "2" (upper right).
- 3. This causes Paint to switch the foreground color to the color you clicked. See the circle labeled "3" (lower left).

Steps to create an HTML equivalent of a graphic, continued:

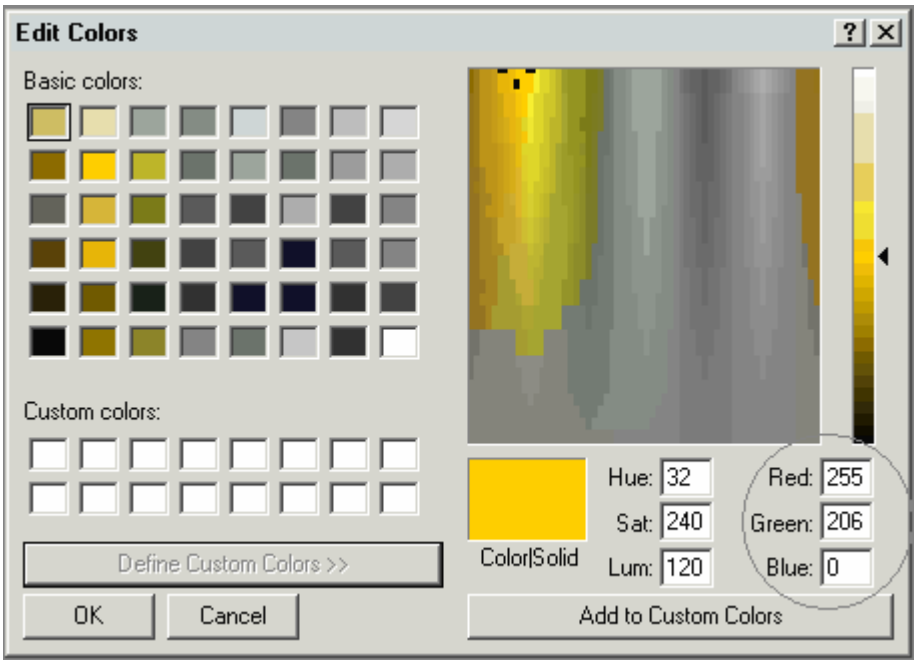
- 4. From the menu bar, select Colors > Edit Colors.



- 5. At the bottom of the Edit Colors dialog box, press the Define Custom Colors button.

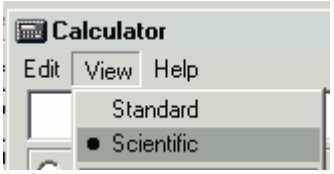


- 6. The RGB values for the color you clicked on will be on the right side of the dialog box. Write down the numbers, because you'll need them in step 8.

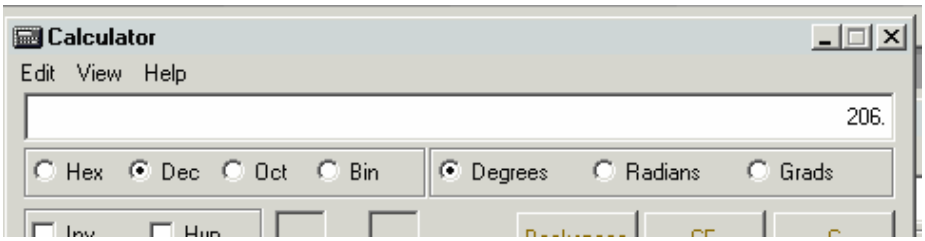


Steps to create an HTML equivalent of a graphic, continued:

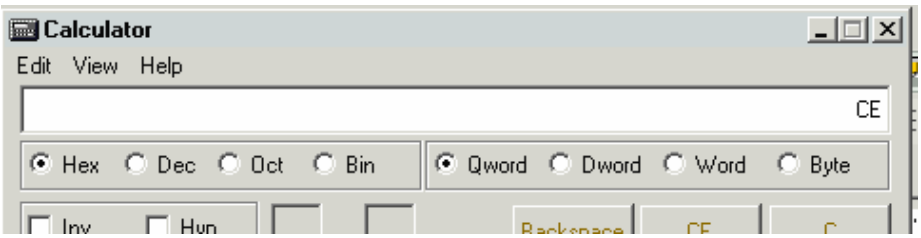
- 7. In the Calculator program, select View > Scientific if that's not already the view you normally use:



- 8. To convert decimal to hexadecimal using the Calculator accessory, enter the decimal value while the Dec radio button is selected:



- 9. Then press the press the Hex radio button to see the hex value:



- 10. So the red, green, blue values (255, 206, 0) are FF, CE, 00 in hex, coded adjacently in HTML:

```
<td bgcolor="#ffce00"> ... </td> <!-- Use ##ffce00 in CFML. -->
```

The following is an HTML-only equivalent of the original graphic. If you're reading a paper copy of this document printed on a black and white printer, you can see this same HTML table in color by going to http://tech-net.sba.gov/index.cfm and clicking on "Text Only" hotlink. Do a View Source, if you like, to see the bgcolor attribute:

5.3.8 BLOBs, CLOBs and Text Datatypes, and CFQueryParam

As mentioned above in Section 3.4.1 ("Structured Query Language (SQL) versus Stored Procedure Calls"), Sybase stored procedures cannot handle BLOB and Text datatypes as parameters. This is also true of the Oracle CLOB datatype. In these 3 cases, you have no choice but to use CFQuery and SQL to do inserts and updates.

In the case of Oracle CLOB data, if the amount of data exceeds 4000 characters, Oracle will limit the amount of data transferred to 4000 characters. In this case, you MUST use CFQueryParam. This is in "The Right Way To Do It" section because nothing else seems to work. Examples:

```
<cfquery ...
insert into EvtTbl
    (
    EvtId,
    EvtDesc,
    EvtDt
    )
Values
    (
    53,
    <cfqueryparam value="#Variables.EvtDesc#"
                  cfsqltype="cf_sql_clob">,
    '#Variables.EvtDt#'
    )
</cfquery>
```

or

```
<cfquery ...
update EvtTbl set
    EvtDesc = <cfqueryparam value="#Variables.EvtDesc#"
                            cfsqltype="cf_sql_clob">
Where EvtId = 53
</cfquery>
```

Note that quotes are NOT put around the <cfqueryparam> tag. All of the quote handling is done by the <cfqueryparam> tag itself.

Note the similarity of CFQueryParam to CFProcParam, which performs a similar function. You don't really have to code CFProcParams, however, because we have a Stored Procedure Call File Generator that does that for you. Historically, CFProcParam became a part of the ColdFusion language first, many years before CFQueryParam. This may explain why many ColdFusion developers are unaware that CFQueryParam even exists. But passing more than 4000 characters to an Oracle CLOBs is a situation where you have no choice but to use it.

The following section (5.3.9, "SQL Injection, Data Validation and CFQueryParam") documents another reason why you may want to use CFQueryParam.

5.3.9 SQL Injection, Data Validation and CFQueryParam

CFQueryParam is also useful to prevent "SQL Injection" attacks. SQL Injection is a technique used by hackers to sabotage other people's databases. There are 2 ways to do it, one way for numeric fields and another way for non-numeric fields. Suppose you have a text box on a form (either a <textarea> form element or an <input type="text"> form element without the maxlength attribute). Suppose further that a hacker knows the name of a database table used by your Web site. Say, for example, that table name is PersonTable. The hacker could attack your database as follows:

If the field is numeric, enter **123); delete from PersonTable;**
If the field is non-numeric, enter **Gotcha'; delete from PersonTable;**

When it comes time to update the database, the hacker's manually-entered SQL is "injected" into the ColdFusion page's SQL statement to do the insert or update, like so:

```
If the field is numeric:

    insert into CompanyTable
        (
        ...,
        AnnualGrossRevenue
        )
    values
        (
        ...,
        #Variables.AnnualGrossRevenue#
        )

becomes:

    insert into CompanyTable
        (
        ...,
        AnnualGrossRevenue,
        ...
        )
    values
        (
        ...,
        123); delete from PersonTable;,
        ...
        )
```

```
If the field is non-numeric:

    update CollateralTable set
        CollateralDesc = '#Variables.CollateralDesc#'
    where ...

becomes:

    update CollateralTable set
        CollateralDesc = 'Gotcha'; delete from PersonTable;'
    where ...
```

Note that it doesn't matter whether it's an insert or an update. It also doesn't matter whether you're updating the same table name. As long as PersonTable is in the same database as CompanyTable or CollateralTable, the hacker's manually-entered SQL will be executed. The delete without a "where" clause throws away all data in the PersonTable. In the non-numeric example, the hacker does twice the damage: By keeping the update statement from reaching its own "where" clause, all CollateralDesc columns in CollateralTable will be also set to "Gotcha". The SQL that FOLLOWS the SQL Injection will be syntactically incorrect, but by then the damage is done.

Of course, SQL Injections rely on knowing the names of database objects. DO NOT REVEAL OUR REAL DATABASE OBJECT NAMES IN ANY COMMUNICATION THAT THE PUBLIC CAN SEE. And if you have a "Show SQL" debugging feature, hide it from the public as well. (And don't publicize its existence.) In addition, there are 3 ways to preventing SQL Injection attacks in ColdFusion: careful data validation, avoidance or careful use of PreserveSingleQuotes and CFQueryParam.

Careful Data Validation: If a numeric field is validated to be numeric, it cannot contain SQL Injection code, period.

Avoidance or Careful Use of PreserveSingleQuotes: Normally, ColdFusion will double all occurrences of single-quotes coming from variables between <cfquery> and </cfquery>. So, for example, If CompanyName contains "Joe's Diner", between <cfquery> and </cfquery>, **'#CompanyName#'** becomes **'Joe''s Diner'**. (between the e and the s, those are 2 single quotes.) By the standardization of SQL, doubling single quotes is how a single single-quote can be saved to a non-numeric column. Normally, this behavior of ColdFusion prevents SQL Injection using non-numeric columns. But sometimes there are situations where you need to control the number of occurrences of single-quotes. In those situations, ColdFusion lets you say **where #PreserveSingleQuotes(Variables.WhereClause)#**, for example. If you use PreserveSingleQuotes, you MUST, MUST, MUST double all single-quotes that were input by the user. For example:

```
<cfset Variables.WhereClause = Variables.WhereClause
       & "and (CompanyName = '"
       & Replace(Form.CompanyName, "'", "''", "ALL")
       & "') ">
```

CFQueryParam: The ColdFusion documentation asserts that a major reason to use CFQueryParam is that it will prevent SQL Injection attacks. See the previous section (5.3.8, "BLOBs, CLOBs and Text Datatypes, and CFQueryParam") for example code. However, the "right way to do it" is to always do careful data validation, and to always avoid or carefully use PreserveSingleQuotes, regardless of whether or not you use CFQueryParam.

5.3.10 Cross-Browser HTML and JavaScript for Internet Sites

See section 3.5.1, above, for current browser support standards. The following are specific explanations and examples to help you bring your code into compliance with that section.

If your code will be seen only by SBA employees and contractors behind the SBA firewall, it's said to be an "Intranet" application. Intranet applications must be compliant with the SBA's official browser, which is currently Microsoft Internet Explorer ("MSIE"), version 6.

If your code will also be seen by the public outside the SBA firewall, it's said to be an "Internet" application. Internet applications must be compatible with the 3 most recent major versions of MSIE and Netscape, for both PC and Macintosh. Another browser that we often support, though it's not mandatory to do so, is Safari for the Macintosh, simply because of its market share in the Mac community. Developers of Intranet-only applications should also learn the Internet rules, in case your application is ever opened up to the public, and so that you will someday be trusted to work on public SBA sites.

This section is pertinent to our ColdFusion developers. isn't a "Right Way To Do It" about ColdFusion, per se. It's HTML and JavaScript associated with application development. So the audience is right, even though it's a little off-topic for a ColdFusion document.

HTML To Be Avoided in Public Applications

`<marquee>`

Reason: MSIE only
Instead: Use a Java applet or Flash animation.
Examples: Specific-purpose Flash animation "sbapartner.swf" at http://www.sba.gov/
General-purpose "mflip.class" Java applet at http://www.sba.gov/tmonline/

`<textarea wrap="virtual/physical">`

Reason MSIE only
Instead: Use wrap="soft/hard".

`<layer>`

Reason: Netscape only
Instead: Use with CSS-P (cascading style sheet positioning).

`<formelement ... disabled ...>`
`<formelement ... readonly ...>`

Reason: MSIE (and some versions of Netscape) only
Explanation: Some Netscape versions support this, but not all, so it's not guaranteed to work. It's okay to use disabled and readonly in HTML and the true/false properties disabled and readonly in JavaScript, but it's not okay to rely on those techniques to work. See the JavaScript formelement.disabled item below for a JavaScript addition that supports Netscape as well.

JavaScript To Be Avoided in Public Applications

`window.location.href (URL);`

Reason: MSIE only

Explanation: Within window.location, href is a property, not a method.

Instead: Use href as a property, assigning the URL to it:

```
window.location.href = "dsp_gotothispage.cfm";
```

`dropdownmenu.value`

Reason: MSIE only

Explanation: <option> tags have values, not the <select> tag (drop down menu) that contains them.

Instead: Use selectedIndex to retrieve the selected option. If selectedIndex >= 0, use:

```
dropdownmenu.options[dropdownmenu.selectedIndex].value
```

`history.go(0); // (to force a hard page reload)`

Reason: MSIE only

Explanation: Like the Back and Forward browser buttons, the go method is designed to be the least destructive of all page loads. It requests a reload from the memory copy of the page as last imaged, with form elements' contents preserved, assuming that the page isn't expired from cache. Only MSIE implements go(0) as a destructive reload of the current page. Therefore, it appears that Netscape's go(0) doesn't work, when in fact Netscape is doing it right.

Instead: Use the method specifically provided for a hard reload of the current page:

```
window.location.reload(true);
```

Clever trick: Request the current page with an ignored parameter tacked onto the URL. Set the ignored parameter to the current time, so that the generated URL won't match any URL in cache. This forces a browser request all the way back to the server, the hardest of all possible reloads, even if the request passes through a proxy server that also caches pages:

```
window.location.href = "dsp_mypage.cfm?Time=125703";
```

`radiobutton.value`

Reason: MSIE only

Explanation: A group of multiple radio buttons with the same name is an Array object. The Array object doesn't have a value, the individual elements of the array do. When it sees this syntax, MSIE scans the array, finds the one with the checked attribute equal to true, and returns that as the value. Netscape does not.

Instead: You have to do it manually, in case the browser is Netscape:

```
var sValue = null;
for (var i = 0; i < radiobutton.length; i++)
    if  (radiobutton[i].checked)
        {
        sValue = radiobutton[i].value;
        break;
        }
// Now sValue can be used as the value of the group.
```

JavaScript To Be Avoided in Public Applications, continued:

`document.all.formelementname`

Reason: MSIE (and some Netscape versions) only

Explanation: Some Netscape versions support this, but not all. So it's not guaranteed to work.

Instead: The new cross-browser way to do this is to give the element an id attribute that's unique within the page. (An id attribute must be unique within the page. That's why it's called the element's id.). Then use document.getElementById() to retrieve it. This has the advantage of working on HTML elements as well as form elements, as in the following example:

```
<span id="WSJ">
    Spread over Wall Street Journal Prime:
</span>
...
<label><input type="Radio" ... onClick="
document.getElementById('WSJ').innerHTML =
  'Spread over Wall Street Journal Prime '
  + '<b>(MANDATORY)</b>';
"> Variable Interest</label>
```

`<script language="VBScript"> (etc)`

Reason: MSIE only

Explanation: To be cross-browser compatible, don't use any scripting languages other than JavaScript. Don't give any language attribute other than "JavaScript" (the default if the language attribute is not given), "JavaScript1.1" or "JavaScript1.2". Don't give "JavaScript1.1" or "JavaScript1.2" versions without first defining an equivalent "JavaScript" version, in case the browser doesn't support "JavaScript1.1" or "JavaScript1.2" (unlikely).

`top.AppData.document.myform.StCd.options[i]=new Option("Utah","UT");`

Reason: Netscape only

Explanation: As of version 5.0 of MSIE, you can no longer directly populate drop-down menus that reside in a different frame.

Instead: Define a function in the frame that contains the drop-down menu. Because it resides in the same frame, the new function is able to manipulate options in the drop-down menu. From other frames, call that function.

`formelement.disabled`
`formelement.readonly`

Reason: MSIE (and some versions of Netscape) only

Explanation: Some Netscape versions support this, but not all. So it's not guaranteed to work. It's okay to use disabled and readonly in HTML and the true/false properties disabled and readonly in JavaScript, but it's not okay to rely on those techniques to work.

In addition: Add JavaScript to reset the form element if changed:

```
<input type="Text" ... disabled ... onChange="
this.value = this.defaultValue;">
```

5.3.11 Suppressing Extraneous "White Space"

The term "white space" refers to spaces, tabs, carriage returns and line feeds, all of which result in white space when you select View Source on a Web page. In some cases, this is merely a nuisance, forcing you to scroll down to see the HTML you wanted to see. But in other cases, such as comma-separated-values for Excel spreadsheets and Word wizards, or XML generation, it can render your generated output totally worthless and unusable.

5.3.11.1 What Causes It

Here's an example of what causes extraneous white space. Suppose you have a query GetEmps of 25 employees and their salaries. To avoid hitting the database a second time to calculate sum(AnnualSalary), you choose instead to code the following loop without any white space suppression at all:

```
<cfset TotalOfSalaries = 0>
<cfloop query="GetEmps">
    <cfset TotalOfSalaries = TotalOfSalaries + GetEmp.AnnualSalary>
</cfloop>
<cfoutput>
Total salaries during fiscal year: #DollarFormat(TotalOfSalaries)#.
</cfoutput>
```

This results in 53 blank lines preceding the line that begins "Total". Why? The reason is, **without any white space suppression, ColdFusion treats anything and everything between CFML tags as data to go out to the Web page**. You don't see the white space in the example code above, because it's white on white, so let's highlight it. Suppose **e** means "end of line", generally carriage-return-line-feed on Windows or just line-feed on Unix. Suppose **b** means blank. Here's where the hidden white space is:

```
<cfset TotalOfSalaries = 0>e
<cfloop query="GetEmps">e
bbbb<cfset TotalOfSalaries = TotalOfSalaries + GetEmp.AnnualSalary>e
</cfloop>e
<cfoutput>e
Total salaries during fiscal year: #DollarFormat(TotalOfSalaries)#.e
</cfoutput>e
```

The second and third lines (cfloop and cfset) are executed 25 times, so that results in 50 end-of-line characters. The fourth line (/cfloop) is also executed 25 times, but only once does the end-of-line after it get sent to the Web page (after the last loop). So the first, fourth and fifth lines each contribute one more end-of-line to the output Web page, for a total of 53 blank lines before the "Total" line. The blanks at the start of the third line also get sent, but they're white on white, so you don't see them. The same would be true if **bbbb** was one tab instead of 4 blanks.

5.3.11.2 When It Can Be a Serious Problem

When you're generating comma-separated-values, every blank and every end-of-line is significant. When you're generating XML, blanks and end-of-line can get into the data (between XML tags), sometimes with disastrous results when stored to the database. When you're doing this within an application with session control, all of the white space from the Application.cfm file sneaks into the output before your page ever even gets a chance to suppress it.

Therefore, if ANY file in an application's directory is used to generate comma-separated-values or XML, or if it's POSSIBLE that there may be a file there SOMEDAY to generate comma-separated-values or XML, you need to attack the white space suppression problem at the application level.

5.3.11.3 Suppressing White Space with CFSilent (Easy, But Not Too Flexible)

One of the easier techniques for suppressing white space is <cfsilent> … </cfsilent>. Basically, <cfsilent> turns off all output until </cfsilent>. It's a mode that propagates through to all shared code as well. So if you say <cfsilent> … <cfinclude template="dsp_something.cfm"> … </cfsilent>, nothing from dsp_something.cfm will get output either, even if it's in a <cfoutput> block.

Hence, one approach to white space suppression would be to enclose all of Application.cfm in a <cfsilent> … </cfsilent> block, being careful not to have an end-of-line after </cfsilent>. Suppose | represents the absolute end-of-file of Application.cfm. It would look something like this:

```
<cfsilent>e
Application.cfm code e
</cfsilent>|
```

If you do this, no white space from Application.cfm will sneak into any of the files that it controls, and it's up to the requested page to decide whether or not to do its own white space suppression. In the case of comma-separated-values or XML generation, it certainly would do its own white space suppression.

What makes <cfsilent> relatively inflexible is the fact that you can't turn it off from within the enclosed block, which makes it difficult if not impossible to throw in debugging displays when necessary. If everything is at the same nesting level, you could code </cfsilent>, followed by the debug code you wish to display, and then <cfsilent again. The problem is that it's very unlikely that the place you want to throw in a debugging display is at the same nesting level as <cfsilent>. Therefore, you would have to nest turning <cfsilent> off and on at all levels, so as to balance out ever start tag with an end tag. It really hampers debugging in an emergency. If the problem is in a cfincluded file, there's no way to turn it off at all.

5.3.11.4 Suppressing White Space with CFProcessingDirective (Doesn't Propagate)

A noble attempt to get around these problems was <cfprocessingdirective suppresswhitespace="Yes"> … </cfprocessingdirective>. It allows a nested <cfprocessingdirective suppresswhitespace="No"> block to turn it off, regardless of nesting level.

The problem with this technique is that it DOESN'T propagate through to shared code, such as cfincluded files, custom tags, cfmodule, etc. The result is that all shared code files have to also use this same technique. (It would have been ideal if this mode propagated through to shared code, but then could be overridden in the shared code, but that's not how they made it work.)

5.3.11.5 Suppressing White Space with EnableCFOutputOnly Mode (Absolute Control)

Of all the many ways to suppress white space, <cfsetting enablecfoutputonly="Yes"> was the first, and is still by far the best for the following reasons:

- It has the virtue of propagating through to shared code automatically.
- It can be overridden in shared code (by using cfoutput).
- You can throw in debug displays anywhere.
- You have absolute control over what gets into the output.
- It's not a start tag. It doesn't have to be balanced with an end tag. If you want to turn it off, you do that with <cfsetting enablecfoutputonly="No">.

All you have to realize is this one simple rule. **After <cfsetting enablecfoutputonly="Yes">, NOTHING gets into the output except what's between <cfoutput> and </cfoutput>.** (In a cfscript block, you have to use the WriteOutput function, but that's cfscript's way to do cfoutput.)

So, if you want to allow generation of comma-separated-values or XML output, but you want a majority of your pages not to do white space suppression, you can use the same technique as with <cfsilent> (Remember that **e** is end-of-line and | is end-of-file):

```
<cfsetting enablecfoutputonly="Yes">e
Application.cfm code e
<cfsetting enablecfoutputonly="No">|
```

Then each subordinate page is free to do its own cfsetting, or not, according to its own needs. But an even more powerful usage is to turn it on application wide:

```
<cfsetting enablecfoutputonly="Yes">e
Application.cfm code e
|
```

By not turning it off at end of Application.cfm, EnableCFOutputOnly mode propagates through to of the application's pages, and every page starts out with EnableCFOutputOnly mode on. Each page is free to turn it off to do Web page output, or simply enclose the whole Web page within <cfoutput> and </cfoutput>.

EnableCFOutputOnly mode imposes a coding style of always using <cfoutput> to get something onto the output Web page. This can be disconcerting to newcomers who aren't used to coding <cfoutput> all the time. They may wonder why their new Web page is blank, for example.

All of the /library shared code routines were written to be compatible with EnableCFOutputOnly mode. That way, even if you're generating comma-separated-values or XML, you have all of the /library routines available to you.

How to code for EnableCFOutputOnly mode compatibility:

- If cfoutput is active, turn it off before doing a cfinclude, then turn it back on again. Under CFMX 7 and higher, this prevents the comment header of the included file from generating white space. Example: </cfoutput><cfinclude template="includedfile.cfm"><cfoutput>
- When generating HTML, put all HTML inside the cfoutput block, not just #name# references.
- Don't turn EnableCFOutputOnly mode off or on in shared code.

5.4 Debugging

5.4.1 Don't Turn On CF Debugging Unless You Absolutely Have To

It has recently been revealed that there's a "memory leak" in ColdFusion Server if you have CF Server Debugging turned on. (This is the feature that prints out the contents of variables at the end of a CFML page.)

If you're running your own copy of CF Server on your PC, don't turn on CF Debugging unless you absolutely have to. Turning it on and leaving it on will eventually result in the "Running Out of Resources" alert, and the only way to recover lost memory is to reboot your PC.

If you're experiencing a difficult problem and request that the administrators turn on CF Debugging, they may be reluctant to do so for this exact reason. Use CFDUMP on the Variables scope, Session scope, etc, manually instead.

5.4.2 Use CFDUMP to Debug in ColdFusion MX

CFDUMP actually became available in ColdFusion 5.0, but the SBA skipped 5.0 and went straight to ColdFusion MX. So for us, CFDUMP becomes available in ColdFusion MX. It's VERY useful in debugging.

When your code is misbehaving and you can't figure out why, the usual reason is that some variable somewhere doesn't contain the value you think it contains. CFDUMP is a convenient way to display the entire contents of structures and arrays, including those that are complex and nested.

Sometimes CFDUMP can produce much too much information, but the problem in debugging is usually not enough information.

Examples:

```
<cfdump var="#Variables#"          label="Entire Variables Scope">
<cfdump var="#Variables.DOMObject" label="Parsed XML">
<cfdump var="#getDistOfcCounties#" label="Counties Result Set">
```

In light of 5.4.1, above, and considering that you have to get your IP address configured just to use CF Debugging in the first place, CFDUMP is really a much better way to go.

6 Application Deployment

Application.cfm: Because cfcrypt.exe in c:\cfusion\bin (Windows) and cfencode in /opt/coldfusion[mx]/bin/ (Unix) have proven to be very weak and very easily decrypted, the previous requirement to encrypt Application.cfm in this way is rescinded. Instead, retrieve shared logins through the firewall. See 3.2, Shared (or "Generic") Logins.

"Stage" Directories: Because some directories contain /save subdirectories, experimental files, debug files, etc, that should not be released to the test or production environments, most applications have associated subdirectories of the development server's document root with "stage" prefixed to their names. Only files copied to the stage directory will be released to subsequent environments. Generally, stage directories are controlled by government employees, not developers.

"EAR" and "WAR" File Deployment: ColdFusion MX 7 now supports application deployment as EAR (Enterprise Archive) or WAR (Web Archive) files. These are special kinds of JAR (Java Archive) files. Basically, all 3 formats are Zip files containing Java class files and directories. This means that a given application can be released to test and production environments without source code. This is a much more secure way to deploy, so there's a good chance that we will use this feature some time in the future. Note that the EAR or WAR file is very much like one of our stage directories, except that it's only one file, and it contains only executables (and a stage directory contains only source code).

7 Programming Cautions ("Gotchas" We've Discovered)

7.1 All Versions of ColdFusion

Several factors have been found to cause ColdFusion Server to crash in the past. Some of them may only be a problem under CF 4.5, while other issues may still be a problem under ColdFusion MX (we don't know yet). Regardless, the consequences of being unable to resolve issues stemming from development environment crashes are significant. Therefore, you MUST avoid making the following mistakes at all costs:

7.1.1 CFPROCRESULT

Never code a CFSTOREDPROC without a CFPROCRESULT. Under CF 4.0, this would only result in a cryptic error message if the stored procedure returned a result set and there was no CFPROCRESULT. Under CF 4.5, the same situation crashes ColdFusion Server. Even if the stored procedure doesn't currently return a result set, the database group could add one at any time. (It's their prerogative to change stored procedures, after all.) So, always code a CFPROCRESULT. If you aren't using it, a typical one is

```
<CFPROCRESULT NAME="Ignored">
```

Since you should be using shared code to call stored procedures (see section 4.1.17 Stored Procedure Call Files), and since SPC files always have a CFPROCRESULT, you don't have to worry about this problem if your application is compliant with SBA ColdFusion Standards.

7.1.2 Calling a Java Method

Never code the wrong number of parameters to a Java object method. When the GLS Jaguar method cfAuthenticate.initComponents was changed from 4 parameters to 5 parameters, old calls that had only 4 parameters crashed ColdFusion 4.5. Always make SURE you call Java object methods with the correct number of parameters.

7.1.3 Frequent Server Crashes

If every time you test a certain page in your application, you're asked to log in again, consider that maybe your code is crashing ColdFusion Server. This is the primary symptom that you've found something that crashes CF Server. When you find out what it is, document it here!!

7.2 ColdFusion 4.5

7.2.1 The "Randomly Zeroed Out Money Fields" Problem

In ColdFusion 4.5, with Sybase native driver datasources, record sets containing money fields randomly get zeroed out. The solution is to substitute

```
CONVERT(NUMERIC(15,2), moneyfieldname) AS moneyfieldname
```

for moneyfield, if you control the SQL, or to use output parameters of a stored procedure to retrieve the money field, or to use an ODBC datasource instead.

7.2.2 Sometimes You Get Errors on the Next Database Call

An exceedingly bizarre behavior of CF 4.5 with Sybase drivers was that you could get an error on one database call, but the error would not be reported until the next database call (example: "foreign key constraint violation, insert fails" when the call was actually a select). This typically happened when an underlying table was recompiled, but the stored procedures that reference it were not recompiled.

Therefore, when you get a Sybase error message that makes no sense, consider the possibility that it could have occurred on the prior database call for the same pooled connection. Figure out what was the previous database call in that situation. This may get you closer to where the error actually occurred.

7.2.3 Sybase Error 3621

In addition, you might see Sybase error 3621, which could be caused by the need to recompile database objects dependent on a changed object.

7.2.4 "Unknown Connect error!"

In addition, you might see "Unknown Connect error!", which could also be caused by the need to recompile database objects dependent on a changed object.

But "Unknown Connect error!" can also mean that the datasource is not defined.

7.3 ColdFusion MX 6.x (and Conversion to MX in General)

7.3.1 JDBC: Like ODBC, Delimit Non-Numeric Literals with Single Quotes

Like ColdFusion, Sybase intrinsically supports both single quotes and double quotes to delimit strings. Under Sybase native drivers, double quotes were allowed. In fact, because they occur less often in data, double quotes were considered the preferred way to delimit Sybase strings.

But as of ColdFusion MX, which uses JDBC drivers, single quotes aren't just allowed. They're required. Double quotes won't work.

7.3.2 JDBC: Parameters to Stored Procedures Must Be in Correct Order

In ColdFusion 4.x and 5.x using Sybase native drivers, the CFPROCPARAM DBVARNAME attribute allowed parameters to stored procedures to be out of order. Under ColdFusion MX, we have to use JDBC drivers, and JDBC doesn't yet support this feature of Sybase. The result is, under CFMX, CFPROCPARAMs are passed to stored procedures in the order they're coded, and DBVARNAME is ignored.

We dealt with this problem using Stored Procedure Call files. See 4.1.17 Stored Procedure Call Files, above, for detailed information about how to call SPC files.

7.3.3 JDBC: Designation of Input and Output Parameters Must Be Correct

A runtime error will be generated if the type="In", type="Out" or type="InOut" attribute is incorrect.

Since the use of SPC files is now a Coding Standard (see 4.1.17 Stored Procedure Call Files, above), you don't have to worry about this problem if your application is compliant with SBA ColdFusion Standards.

7.3.4 JDBC: CFSQLTYPE="CF_SQL_DATE" Is No Longer Supported

CFSQLTYPE="CF_SQL_DATE" Is No Longer Supported. The workaround is to use CFSQLTYPE="CF_SQL_TIMESTAMP". This was also added to the SPC file generator.

Since the use of SPC files is now a Coding Standard (see 4.1.17 Stored Procedure Call Files, above), you don't have to worry about this problem if your application is compliant with SBA ColdFusion Standards.

7.3.5 JDBC: Nullstring Passed in CFPROCPARAM Behaves Like Space

In CF 4.5, passing VALUE="" in CFPROCPARAM behaved exactly like passing NULL="Yes". Now it does not. Rather, it behaves like passing VALUE=" " (single space). This is an apparent compatibility of JDBC with ODBC behavior, since the same thing can be observed in Microsoft SQL Server using ODBC drivers.

Since this was a systemic problem affecting hundreds of stored procedure calls, and since the DBVARNAME problem required going to generated Stored Procedure Call files anyway, the SPC file generator was modified to generate the following on all non-bit input parameters:

```
<cfif Len(Variables.varname) GT 0>
    <cfprocparam ... value="#Variables.varname#" ...>
<cfelse>
    <cfprocparam ... null="Yes" ...>
</cfif>
```

Since the use of SPC files is now a Coding Standard (see 4.1.17 Stored Procedure Call Files, above), you don't have to worry about this problem if your application is compliant with SBA ColdFusion Standards.

7.3.6 JDBC: the Syntax "= NULL" Is No Longer Allowed

Use "IS NULL" instead.

7.3.7 JDBC: NULL Is Not a Value of a List, Either

Under CF 4.5, the following would be true if LoanAppEligEvalInd is NULL:

```
and LoanAppEligEvalInd NOT IN ("Y","N")
```

This is not true under CFMX, however, because NULL isn't a value. Under CFMX, you need to specify:

```
and (   (LoanAppEligEvalInd IS NULL)
     OR (LoanAppEligEvalInd NOT IN ("Y","N"))
```

7.3.8 JDBC: Stored Procedures Behave Differently Because of JDBC

The aforementioned details pertaining to JDBC apply to stored procedures too. Sybase somehow detects that the call was via JDBC, and all of the JDBC rules apply within the stored procedure as well. No saying "= NULL". No treating NULL as a value of a list.

If you execute a stored procedure in a database tool, such as RapidSQL, DBArtisan or ISQL from a command prompt, and it works just fine, but you do the exact same thing in ColdFusion MX and it misbehaves, look for JDBC-sensitive code in the stored procedure.

7.3.9 StructKeyList

In ColdFusion 4.x and 5.x, the StructKeyList of a structure is in all upper case. But in ColdFusion MX, it's in the same case as the last CFSET that defined each element. (The reason is for compatibility with XML, in which elements are case sensitive.) This can cause your code to malfunction if you are expecting upper case and do a Find(), for example, instead of a FindNoCase(). Beware of this difference if you work with structures a lot, particularly structures generated from XML.

7.3.10 JSessionId

At the discretion of the system admins, ColdFusion MX may use JSessionID, rather than CFID and CFToken, to establish a session. See 3.2.10 Session Control (CFMX), above, for how to deal with JSessionId in a way that won't cause your code to fail.

7.3.11 Periods in Variable Names

Prior to ColdFusion MX, periods in variable names were allowed. In CFMX, they are not allowed.. If CFMX locates a period and the left side of the period is not a scope or known structure, it creates a new structure and defines everything to the right of the period as an element of that structure.

As a result, <cfset Variables.MyNewVar.WithAPeriod = 0> is legal before and after CFMX. Under 4.5, it defines a new variable "MyNewVar.WithAPeriod" that contains 0. Under MX, it defines a new structure "MyNewVar", which contains a variable "WithAPeriod" that contains 0. Though not significantly different, it allows variable names "containing periods", so to speak, to be much longer.

7.3.12 When Calling Java Methods, Datatype May Not Be String

To deal with problems associated with datatype in previous versions of ColdFusion, all of our Java object methods take Strings as input and return Strings as outputs. However, under some circumstances, such as data from the database whose data type was datetime or int, ColdFusion may internally "type" the data as non-String objects. Normally this doesn't present a problem until you call a Java method.

Java methods can be "overloaded" (given the same name) and to distinguish 2 methods with the same name, the parameters to the method (called the "signature" in Java terminology) are looked at. If a data item is typed as a non-String datatype, the method will not be found (because, as said before, all of our input parameters and return values are Strings).

The solution is to use a built-in ColdFusion function JavaCast that was specifically created for this exact purpose. Ideally, the JavaCast call should be done as close to the Java method call as possible, so as to eliminate the possibility that CF might retype the data as a non-String datatype. Ideally, the actual parameters should be coded with JavaCast if there's any chance that they could be considered non-String:

```
<cfset PrtId = MyJavaObj.GetPrtByLocId(JavaCast("String",LocId))>
```

7.3.13 The Data Validation for CFFORM Date Elements Is Incorrect

CFFORM generates its own JavaScripts for client-side data validation. In CFMX, the JavaScript for validate="Date" is incorrect. The workaround is to use our own EditDate or EditDateNonFuture JavaScripts (See 4.6.3 and 4.6.4, above.)

In general, CFFORM doesn't allow turning CFOUTPUT off and on, so if you're going to use the SBA's JavaScripts anyway, it doesn't make much sense to use CFFORM. If you can structure the code so as not to turn CFOUTPUT off and on in the range of CFFORM, it might be worthwhile to use CFFORM in the future, to take advantage of CFMX 7's support of X/Forms.

7.4 ColdFusion MX 7.x (and Conversion to 7.x)

7.4.1 CR and LF Can No Longer Appear in CFLOCATION URLs

ColdFusion 4.5 through 6.1 would allow Carriage Return (CR) and/or Line Feed (LF) to be in CFLOCATION URLs, provided that they were encoded with URLEncodedFormat. That's no longer allowed, because it was perceived to be a potential security vulnerability.

For more information, see
http://www.macromedia.com/cfusion/knowledgebase/index.cfm?id=46ba02fd

7.4.2 Double Slash in a Path Is No Longer Treated the Same as One Slash

Because a directory must have a name, Unix treats multiple adjacent slashes as equivalent to only one slash. For example, /export//home/username is equivalent to /export/home/username. However, ColdFusion is not Unix. It does its own parsing of paths, and as of CFMX 7, it treats multiple adjacent slashes as significant.

This sort of thing happens when a directory name is supplied by a configuration parameter. Out of habit, people typically add a trailing slash, to indicate that it's a directory. Then later on, in the context of a larger URL, an extra slash is given as a separator. For example:

```
Application.cfm:
<cfset Variables.LibURL = "/library/">

In a display page:
<cfmodule template="#Variables.LibURL#/customtags/mainnav.cfm"
```

In this example, because LibURL already contained a trailing slash, the resulting code in the display page was, in effect:

```
<cfmodule template="/library//customtags/mainnav.cfm"
```

resulting in an error.

The solution is NOT to add a trailing slash in the definition of the configuration parameter. This allows the slash to be added later when referenced, which makes for more easily readable code.

7.4.3 CFOUTPUT Mode Partially Propagates to Included Files

If you had <cfsetting enablecfoutputonly="Yes"> mode turned on under CF 4.x, 5.x or 6.x, and CFOUTPUT mode was on, neither the generation of Web page output nor the resolution of #Variables.something# would propagate through to a CFINCLUDE'd file. You would have to specifically turn it back on with another CFOUTPUT in the included file.

But as of CFMX 7, the generation of Web page output propagates through to an include file. (The resolution of #Variables.something# does not propagate through, but Web page output does.) This can result in unwanted white space, which is what <cfsetting enablecfoutputonly="Yes"> was meant to eliminate.

The solution is to turn off CFOUTPUT mode before the CFINCLUDE and turn it back on afterwards, as in this example:

```
</cfoutput><cfinclude template="myinclude.cfm"><cfoutput>
```

7.4.4 <cfset Variables.RequestTimeout = seconds> No Longer Works

This undocumented technique (publicized by Ben Forta) no longer works. For a complete discussion of the problem and what to do about it, see 5.3.4 How to Change the "RequestTimeout" of a Page, above.

7.4.5 Web Services: Arguments Scope Evaluated Ahead of Variables Scope

It was actually documented in ColdFusion MX 6.1 that the Arguments scope was evaluated ahead of the Variables scope. This was not the case in 6.1. In fact, the Variables scope was evaluated ahead of the Arguments scope. But in CFMX 7, it's true.

7.4.6 Web Services: Error Messages Have Gotten More Generic

In the past, Web Service error messages would often be quite specific, saying exactly the same error that would appear in the "Gray Screen of Death" when a Web page errors. But sometimes now under CFMX 7, the only error we get is that the client was unable to invoke the Web Service, giving the impression that there's some sort of system problem, when the problem is really an unclosed quote, for example, or an equals sign where there ought to be an "IS".

This is why it's advisable to have a Web page file upload interface to the same Web Service capability. The Web page interface gives the "Gray Screen of Death", with plenty of debugging information. This provides a far more convenient way to debug code than the Web Services interface, at least until CFMX Web Services are given back their more detailed error messages.

7.4.7 Web Services: An Empty XML Namespace URL Crashes Axis

If the XML that calls a Web Service has an empty URL, the Web Services system software (called "Axis") encounters an org.xml.SAXParseException and crashes. Advise clients not to do this. Example:

```
<MyPrefix:SBA_ETran xmlns:MyPrefix="" version="3.0">
```

The purpose in defining a prefix is to distinguish SBA's XML from that of SOAP, the XML of Web Services. This isn't necessary, and the client doesn't give the URL in the namespace declaration, it's worse than unnecessary, because it crashes the call. Advise clients to use the "default namespace" (that is, don't define a prefix), as in this example:

```
<SBA_ETran version="3.0">
```

7.4.8 Web Services: Application.cfc OnRequest Messes Up Web Services

ColdFusion MX 7 allows for a new and more powerful application framework. Instead of using Application.cfm, CFMX 7 also allows using Application.cfc. If both are found, the newer technique (Application.cfc) will be used and the older technique (Application.cfm) will be completely ignored.

The problem is that Application.cfc files are now automatically invoked by Web Services, and if the OnRequest method is defined, Web Services crash. There is absolutely no way a Web Service can run if an Application.cfc file exists in its path with OnRequest defined. This is very unfortunate, because the OnRequest method is among the most useful and powerful of all Application.cfc methods. (OnRequest can set Variables scope variables, making it ideal for <cfinclude template="Application.cfm">. In other words, OnRequest is the quick upgrade path to Application.cfc that doesn't require a major rewrite.)

Our current application organization structure is to put Web Services into a "/ws" subdirectory of the application directory (example, /elend/ws). Therefore, if you want to use OnRequest, you cannot put Application.cfc at the root directory of the application. In other words, this doesn't work:

```
/myapproot
   Application.cfc (with OnRequest)
   /application_specific_shared_code_directories
   /ws
      mywebservices.cfc
```

You have to do it this way instead:

```
/myapproot
   /application_specific_shared_code_directories
   /web
      Application.cfc (with OnRequest)
   /ws
      mywebservices.cfc
```

8 Example Files

A common concern is that we have too many standards requirements for our programmers to remember everything. Also, sometimes things have to be done in a particular order, and that information gets lost in the maze of requirements. This section attempts to bring everything together into a few example files.

In each of the examples, you will see comments of the form (Std x.y.z) or (BP x.y.z). Std stands for "Standard". BP stands for "Best Practice". The x.y.z part is a reference to the section of this document that explains the standard or best practice in detail. In the electronic version of this document, you can hold down the control key and click on the reference, and it will take you to that section. That is, the control key turns the reference into a hotlink.

8.1 Web Page User Interfaces

In Web Page User Interfaces, you can have Session variables.

Under construction.

8.1.1 Example GLS Application.cfm

```
<cfsetting EnableCFOutputOnly="Yes"> (BP 5.3.11.5)
<!--- (Std 3.3.5)
AUTHOR:      Jeremy Programerton
DATE:        09/20/2004
...
--->

<cfset Request.SBALogSystemName  = "GLSSystemName"> (Std 4.3.1)
<cfinclude template="/library/cfincludes/inc_starttickcount.cfm"> (Std 4.3.7.1)

<!--- Configuration Parameters: ---> (BP 5.2.4)

<cfset Variables.LibURL          = "/library"> (BP 5.2.1)
<cfset Variables.IncURL          = "#Variables.LibURL#/cfincludes">
<cfset Variables.SpcURL          = "/cfincludes">
<cfset Variables.RefURL          = "#Variables.SpcURL#/sbaref">
<cfset Variables.FeedbackEmail   = "jeremy.programmerton@sba.gov"> (BP 5.2.5)
<cfset Request.Version           = "4.2.3"> (Std 3.2.6)

<!--- Establish Application and Session Scopes: --->

<cfapplication
       name                      = "GLS"
       clientmanagement          = "No"
       applicationtimeout        = "#CreateTimeSpan(0,1,0,0)#"
       sessionmanagement         = "Yes"
       setclientcookies          = "No"> (Std 3.3.7)
<cfinclude template="#IncURL#/get_sbalookandfeel_variables.cfm"> (Std 4.1.14)
```

8.1.2 Example Non-GLS Application.cfm

Under construction.

8.1.3 Example Display Page (dsp_xxx.cfm)

Under construction.

8.1.4 Example Display Page in a Frame (dsp_xxx.cfm)

Under construction.

8.1.5 Example Action Page (act_xxx.cfm)

Under construction.

8.1.6 Example OnRequestEnd.cfm

In every directory in which you have an Application.cfm file that calls inc_starttickcount.cfm, create an OnRequestEnd.cfm file containing the following:

```
<cfinclude template="/library/cfincludes/OnRequestEnd.cfm"> (Std 4.3.7.2)
```

Or, if you've parameterized your directory names and paths, as recommended in 5.2.1, use the name you've defined. Example:

```
<cfinclude template="#Variables.IncURL#/OnRequestEnd.cfm"> (Std 4.3.7.2)
```

8.2 Web Services

In Web Services, you generally don't have Session variables.

Under construction.

8.2.1 Example CFC File (xxx.cfc or wbs_xxx.cfc)

Under construction.

8.2.2 Example Included Function File (wbs_xxx.cfm)

Under construction.

9 Guidelines for Editing this Document

9.1 Headers and Footers

After the title page, the Table of Contents section begins top of page header. Every section after the Table of Contents has “Same as previous” as its header. Therefore, if you change the Table of Contents header, it will automatically affect the entire document (except for the title page).

The footers, however, are just the opposite. Each section after the title page has its own footer that names the section (Table of Contents, 1. Introduction, 2. Naming Conventions, etc). Therefore, if you want to change version number or modified date, you will have to edit every section’s footer. Fortunately, global search and replace works on the headers and footers. Furthermore, the version numbers on the title page and all footers are consistent (“Version”, colon, space, version number), as are the last modified dates (“Modified”, colon, space, date). This consistency was intentional, to aid in global search and replace.

9.2 Use of Microsoft Word “Styles” Feature

Insert > Reference > Cross-reference, Automatic numbering and Table of Contents generation will all be automatically accurate if you adhere to the Styles defined for section/subsection numbering:

x	is “Heading 1”
x.x	is “Heading 2” (or, optionally “Heading 2 Page Break Before”)
x.x.x	is “Heading 3” (or, optionally “Heading 3 Page Break Before”)

To set a paragraph to a particular style, it’s generally easiest just to use the Styles drop-down menu (to the left of the Font drop-down menu) at the time you begin the paragraph.

Other useful styles are already defined:

```
<This style is called "CodeExample" (Courier New and indented)>
   <And this one is "CodeExample1", because it has one more indent>
      <And there’s also "CodeExample2", with 2 more indents>
         <This is "CodeExample3">
            <This is "CodeExample4">
               <But "CodeExample5" is as deep as it goes>
               <You wouldn’t want to go deeper than "CodeExample5",
               because the line would have tendency to wrap>
```

Every major tabular display has its own style associated with it, so that you can reformat just that tabular display without affecting others. For example:

- SharedCodeLocation
- StepsList
- ToBeAvoidedH1 (not related to “Heading 1”, special emphasis Style used in 5.3.10)
- ToBeAvoidedH2 (not related to “Heading 2”, special emphasis Style used in 5.3.10)
- ToBeAvoidedH3 (not related to “Heading 3”, special emphasis Style used in 5.3.10)
- UtilityFilePrefixList

Rather than modifying these tables one paragraph at a time, it’s much easier in every way to modify the Style and globally affect all paragraphs with that Style.

9.3 Page Breaks

The following Styles were defined using Format > Paragraph > Line and Page Breaks > “Page break before”. Other than that, they have the same properties as the Styles on which they were based:

> “Heading 2 Page Break Before”
> “Heading 3 Page Break Before”
> “Normal Page Break Before”

In general, you should use these styles in preference to Insert > Break > Page break.

9.4 Default Font and Size

The default font and size (defined in the “Normal” Style) is called “Times 7/11” by law firms. That is, the font is Times New Roman, and the font size is 11-point for normal characters and 7-point for superscripts, subscripts, etc. This is the closest approximation to book type, which most people can read fairly easily. 10-point is too small for some older folks’ vision, and 12-point generally seems too large. In addition, the use of a serif font increases character recognition, again to help older folks’ vision, which is a Section 508 issue.