BlueDragon, BEA WebLogic® Edition 6.2.1

CFML Enhancements Guide

BlueDragon, BEA WebLogic® Edition 6.2.1 CFML Enhancements Guide

Published April, 2006

100 Prospect Place • Alpharetta, Georgia 30005-5445
Phone 678.256.3011 • Fax 678.256.3012
http://www.newatlanta.com

Further, New Atlanta Communications, LLC reserves the right to revise this document and to make changes from time to time in its content without being obligated to notify any person of such revisions or changes.

The Software described in this document is furnished under a Software License Agreement ("SLA"). The Software may be used or copied only in accordance with the terms of the SLA. It is against the law to copy the Software on tape, disk, or any other medium for any purpose other than that described in the SLA.

Contents

BlueDragon, BEA WebLogic® Edition 6.2.1
CFML Enhancements Guide

1 Introduction

BlueDragon is family of server-based products for deploying dynamic web applications developed using the ColdFusion® Markup Language (CFML). BlueDragon features native technology platform integration on the operating system, web server, and database of your choice. CFML is a popular server-side, template-based markup language that boasts a rich feature set and renowned ease-of-use.

In addition to CFML, some BlueDragon editions also implement the Java Servlet API and JavaServer Pages™ (JSP) standards defined by Sun Microsystems, Inc. as component technologies of the Java 2 Platform, Enterprise Edition (J2EE™). BlueDragon for the Microsoft .NET Framework also integrates with ASP.NET pages and .NET components.

BlueDragon provides a high-performance, reliable, standards-based environment for hosting CFML web applications, and enables the integration of CFML with J2EE and Microsoft .NET technologies.

1.1 About This Manual

The *BlueDragon, BEA WebLogic Edition 6.2.1 CFML Enhancements Guide* presents information about the enhanced CFML tags and functions enabled in the implementation of CFML in BlueDragon compared to what developers may expect when using Macromedia ColdFusion MX 6.1. Developers currently working with ColdFusion should also be aware of differences in CFML compatibility, which are discussed in the associated manual, *BlueDragon, BEA WebLogic Edition 6.2.1 CFML Compatibility Guide*.

1.2 BlueDragon Product Configurations

BlueDragon is currently available in four product configurations. Details about these configurations—BlueDragon Server, BlueDragon Server JX, BlueDragon for J2EE Servers, and BlueDragon for the Microsoft .NET Framework—are provided in other related manuals, discussed in the “Additional Documentation” section below. Except where explicitly noted, all references to “BlueDragon” in this document refer to all product configurations.

1.3 Technical Support

If you're having difficulty installing or using BlueDragon, visit the self-help section of the New Atlanta web site for assistance:

http://www.newatlanta.com/products/bluedragon/self_help/index.cfm

In the self-help section, you'll find documentation, FAQs, a feature request form, and a supportive mailing list staffed by both customers and New Atlanta engineers.

Details regarding paid support options, including online-, telephone-, and pager-based support are available from the New Atlanta web site:

http://www.newatlanta.com/biz/support/index.jsp

```
For BEA Systems technical support, go to:

   http://support.bea.com
```

1.4 Additional Documentation

The other manuals available in the BlueDragon documentation library are:

- *BlueDragon, WebLogic Edition 6.2.1 Compatibility Guide*
- *BlueDragon 6.2.1 Server and Server JX Installation Guide*
- *BlueDragon, WebLogic Edition 6.2.1 User Guide*
- *BlueDragon, WebLogic Edition Deploying CFML on J2EE Application Servers*
- *Deploying CFML on ASP.NET and the Microsoft .NET Framework*
- *Integrating CFML with ASP.NET and the Microsoft .NET Framework*

Each offers useful information that may be relevant to developers, installers, and administrators. BlueDragon, BEA WebLogic Edition documents are available at edocs.bea.com and additional documents are available in PDF format from New Atlanta's web site:

http://www.newatlanta.com/products/bluedragon/self_help/docs/index.cfm

2 Overview of Enhancements

As you consider the many enhancements that BlueDragon offers, some will be more compelling than others. While this document presents them in alphabetical order by tags and functions, the following highlights some of the more significant enhancements.

Additionally, developers moving to BlueDragon from older releases of ColdFusion will enjoy the benefits of the language enhancements introduced in ColdFusion 5 and MX, as

discussed briefly later in this section. These offer additional, if indirect, benefits of upgrading to BlueDragon from older releases of ColdFusion.

2.1 Enhanced Features In BlueDragon

Among the most compelling enhancements that will appeal to all CFML developers are

- CFCs enhancements (they can be serialized, duplicated, and more)
- Application-level path mapping with `CFMAPPING`
- Site spidering via `CFINDEX` or admin console
- Support for `CFQUERYPARAM` within a Cached Query
- Enhanced query caching and cache management (see `CFQUERY`)
- Enhanced page content caching (including caching to disk, see `CFCACHECONTENT`)
- Ability to render CFML dynamically from a variable or query (see `Render()`)
- Ability to send `CFCONTENT` data from a variable (like a database image column)
- Easy include of JSP or ASP.NET page output with `CFINCLUDE PAGE`
- Easy transfer of control to another CFML, JSP, or ASP.NET page via `CFFORWARD`
- Error logging and processing (see `CFERROR`)
- Available CFML execution tracing with `CFDEBUGGER`
- Various `CFDUMP` enhancements
- Support for assertions (see `CFASSERT` and `Assert()`)

There are still other enhancements that might appeal to many developers:

- Support for request throttling (see `CFTHROTTLE`)
- Image processing via `CFIMAGE`
- Zip file creation and extraction using `CFZIP`
- IMAP mail processing with `CFIMAP`
- XMLRPC request processing using `CFXMLRPC`
- Enhanced page buffering via `CFFLUSH`
- Enhanced mail file attachments via `CFMAILPARAM`
- Enhanced control over `CFSEARCH` and enhanced metadata in returned results
- Freedom to use relative paths in many tags, using `URIDIRECTORY` attribute
- Ability to continue a `CFLOOP` with `CFCONTINUE`
- Ability to temporarily halt page execution with `CFPAUSE`

- Ability to sort query results with `QuerySort()`
- Ability to delete rows from a query resultset using `QueryDeleteRow()`
- Ability to remove duplicate list entries with `ListRemoveDuplicates()`
- Ability to pass in XSTL arguments on `XMLTransform()`
- Ability to process `Application.cfm` even when a requested file is not present

These are just some of the enhancements. There are dozens more, as discussed throughout this document.

Finally, the .NET edition of BlueDragon offers many unique enhancements. While many of them are about integration with ASP.NET and the .NET Framework, as spelled out in the manual, *Integrating CFML with ASP.NET and the Microsoft .NET Framework*, some are simply enhancements to CFML that could benefit traditional CFML developers:

- Support for DSN-less connections in database tags like `CFQUERY, CFSTOREDPROC`
- Extension of `CFOBJECT, createObject()` to call .NET objects
- Support of `TIMEOUT` on `CFINVOKE` of Web Services
- Support of `DBVARNAME` on `CFSTOREDPROC` to support/validate named arguments
- Available `GetHttpContext` function to provide additional request metadata

2.2 CFML Enhancements in ColdFusion 5 and MX

Finally, developers upgrading from earlier releases of ColdFusion, especially ColdFusion 5 and before, should bear in mind that BlueDragon brings nearly all the enhancements that were added in CF5 and CFMX.

2.2.1 CFML Enhancements Added in CF 5

For instance, developers upgrading from releases before ColdFusion 5 should seek available resources that discuss the new language features added in that release, which are available in BlueDragon, including `CFDUMP`, `CFSAVECONTENT`, `CFFLUSH`, `CFLOG`, and more, as well as functions like `IsCustomFunction()` and `GetHTTPRequestData()` and others.

2.2.2 CFML Enhancements Added in CF MX 6/6.1

Developers upgrading from ColdFusion 5 and before should similarly seek available resources that discuss the many new language features introduced in CFMX, also available in BlueDragon, including `CFCOMPONENT`, `CFFUNCTION`, `CFXML`, and more, as well as functions like `GetHTTPContext`, `URLSessionFormat`, and others.

3 CFML Variables

3.1 Variable Names

In ColdFusion, a variable name must start with a letter and can only contain letters, numbers and the underscore (_) character. In BlueDragon, a variable name may additionally contain the dollar sign ($) character and a variable name may start with an underscore, dollar sign, or letter.

3.2 SERVER Variables

BlueDragon offers its own identifying structure within the predefined `Server` scope, as `Server.BlueDragon`, which contains the following variables:

`Server.BlueDragon.Edition` identifies the edition:

6 - BlueDragon Server (FREE edition)
7 - BlueDragon Server JX
8 - BlueDragon/J2EE
9 - BlueDragon for .NET

`Server.BlueDragon.Mode` identifies the license mode:

0 – development
1 - evaluation (time-limited)
2 - full production

As in ColdFusion, these pre-defined `Server` scope variables are read-only.

4 CFML Tags

4.1 Enhancements Regarding ColdFusion Components (CFCs)

There are a few enhancements in CFC (ColdFusion Component) processing in BlueDragon:

- CFC instances can be duplicated using the `Duplicate()` function
- CFC instances can be serialized (useful with J2EE and .NET session support, and where session replication or persistence requires this)
- CFC instances can be correctly passed roundtrip using web services (from CFMX to BD, but not the other way around)
- BlueDragon does not restrict use of tags before use of `CFSET VAR`

- Variables may be used in the component name on invocation, as well as in a method's `ReturnType` and `CFARGUMENT Type`, and in the CFC named in a `CFCOMPONENT Extends` attribute

4.2 Enhanced CFML Tags

This section lists CFML tag enhancements that are unique to BlueDragon.

4.2.1 CFARGUMENT

BlueDragon permits an expression (variable) to be used in the `Type` attribute. See additional discussion in section 4.1.

4.2.2 CFCOLLECTION

In BlueDragon, `CFCOLLECTION` does not require use of a `PATH` attribute (for indicating where the collection should be stored). If not specified, it defaults to creating the collection in `[bluedragon]\work\cfcollection\`.

On the other hand, only collections created in that default directory are listed in the BlueDragon Administration console, or when `CFCOLLECTION ACTION="list"` is used. Collections created by specifying the `PATH` attribute (placing the collection in another directory) will still be available for use by `CFINDEX` and `CFSEARCH` , but they will not be displayed by these two approaches.

`CFCOLLECTION` also supports a new, optional `WAIT` attribute. See the discussion under `CFINDEX` in section 4.2.10.2.

4.2.3 CFCOMPONENT

BlueDragon permits an expression (variable) to be used in the `Extends` attribute. See additional discussion in section 4.1.

4.2.4 CFCONTENT

Both ColdFusion and BlueDragon support an available `FILE` attribute for `CFCONTENT`, to name a file whose content should be sent to the browser (with its mime type optionally indicated with the available `TYPE` attribute). BlueDragon takes this a step further and lets you send the value of a variable, using a new `OUTPUT` attribute.

An example of using it might be when a `CFQUERY` retrieves a column of binary type from a database (perhaps a graphic). Assuming the variable is `myquery.mygraphic`, you could then send that to the browser in a single step with:

```
<CFCONTENT OUTPUT="#myquery.mygraphic#" type="image/jpeg">
```

4.2.5 CFDUMP

BlueDragon's `CFDUMP` output is enhanced in various ways.

4.2.5.1 AutoExpansion of Nested Structures and Arrays

BlueDragon expands the values of arrays, structures, and other nested objects, providing more information to assist in debugging.

4.2.5.2 VAR is Optional, Automatic Dump of Several Scopes

While the CFDUMP tag VAR attribute is required in ColdFusion, is optional in BlueDragon; if omitted, variables in all scopes (except the CGI and SERVER scopes) are displayed:

```
<CFDUMP VAR="#SESSION#">   <!--- display SESSION variables --->
<CFDUMP> <!--- display variables in all scopes but cgi, server --->
```

Of course, it's permissible to dump the CGI and SERVER scopes by specifying either of them in the VAR attribute. They're just not dumped automatically with the special form of CFDUMP.

4.2.5.3 Additional Information Offered in Dump of Queries

The CFFDUMP output for query result sets shows additional information about the query including the datasource name, the SQL processed, the execution time, the number of records found, and the size in bytes.

4.2.5.4 Available VERSION Attribute to Expand Queries and XML

BlueDragon's dump can show all the records in a query resultset, as well as expanded information about XML objects. This is controlled with an optional VERSION attribute, which takes two values: LONG and SHORT. The default for query result sets is LONG. It applies to CFDUMP both with and without use of the VAR attribute, as described above.

In the SHORT version, a dump of query result set will not show the actual records from the query, but will show other useful information about the query (records found, execution time, SQL string, etc.).

The dump of an XML object will work similarly to the long and short versions available in ColdFusion MX. Whereas in ColdFusion MX, you would click on the displayed XML object to cause it to switch between short and long versions, in BlueDragon you choose the alternative using the VERSION attribute. The default is SHORT.

Currently, only query result sets and XML documents are affected by the VERSION attribute, and it has no effect for other variable types (they can be dumped, but the result is not varied by specification of the VERSION attribute). Similarly, the VERSION attribute setting does not affect query resultsets or XML documents that are contained within another variable or structure being dumped. They use their respective defaults.

4.2.6 CFERROR, CFTRY/CFCATCH, and try/catch

BlueDragon offers an enhancement in error log processing, in that it writes out to a log file the entire error page that would be displayed to a user if the error was not handled. See section 5.7 for additional information.

As of 6.2, that error log page is written even when the error is handled, as with CFERROR. A new variable is available in the CFERROR and ERROR scopes (as well as the cfcatch scope) called ErrorLogFile, which returns the name and location of the logfile that's written. Note that while you cannot include that log file in a CFMAIL from within an error handler (as the write of the file will not be complete until the end of the error handler request processing), you can offer the path and filename of the error log page in a CFMAIL, such as to share with developers for their use in problem resolution.

4.2.7 CFFLUSH

BlueDragon offers an option in the administration console to control whether the generation of HTML output on a page is buffered to page completion or not, and it defaults to buffering the entire page, like ColdFusion. See the *BlueDragon 6.2.1 User Guide* for more information on the topic of page buffering.

If you choose to change the server-wide behavior to buffer less than the entire page (such as to speed delivery of pages to the client or to reduce memory burden in buffering the entire pages to completion), there may be a negative impact on your application in the use of some tags in some situations. To change the behavior on a page-by-page basis to revert to buffering the entire page, BlueDragon offers a new PAGE attribute for the INTERVAL attribute of CFFLUSH, as in:

```
<CFFLUSH INTERVAL=”page”>
```

BlueDragon also supports use of the CFFLUSH INTERVAL="n" attribute, which enables page-level control of the flushing of the buffer after a given amount of generated content. This would be used when the default server-wide setting is set to buffer the entire page but you want to enable buffering on the current page. Note that the maximum value BlueDragon allows for "n" is 128K (128*1024): if you set a larger size then BlueDragon buffers the entire page.

4.2.8 CFFUNCTION

BlueDragon permits an expression (variable) to be used in the ReturnType attribute. See additional discussion in section 4.1.

4.2.9 CFINCLUDE

BlueDragon allows you to include in your CFML pages the output of Java servlets or JavaServer Pages (JSP), or ASP.NET pages in the .NET edition, via the new PAGE attribute of the CFINCLUDE tag. The page attribute specifies the URI for the page to include. It cannot be an absolute file path but instead must be a web server-based path.

Paths that start with “/” start at the document root directory of the web application or web server; paths that don’t start with “/” are relative to the current CFML document

```
<CFINCLUDE PAGE=”/menu.jsp”>
<CFINCLUDE PAGE=”footer.aspx”>
```

This is essentially a simplification of the `GetPagecontext().include()` function, introduced in CFMX and supported also in BlueDragon.

As when using `CFFORWARD` (see section 4.3.6), note that variables set in the included page's `REQUEST` scope will be available on the calling page. To access data from the included page's other scopes (like `FORM` or `URL`), simply copy them to the `Request` scope before performing the `CFINCLUDE`.

When including plain HTML pages, it's best to simply use the more traditional `CFNCLUDE TEMPLATE` approach.

In the J2EE edition, `CFINCLUDE` can also refer to the `WEB-INF` directory in a web app, for example:

```
<CFINCLUDE TEMPLATE="/WEB-INF/includes/header.cfm">
<CFMODULE TEMPLATE="/WEB-INF/modules/navbar.cfm">
```

The advantage of using `WEB-INF` is that files within it are never served directly by the J2EE server, so a user cannot enter a URL to access them directly.

The `CFINCLUDE PAGE` attribute can be used to include CFML pages, in which case the included page's `Application.cfm` (and any `OnRequestEnd.cfm`) will be processed, unlike a typical `CFINCLUDE TEMPLATE`. This behavior is the same as using `GetPagecontext().include()` function.

4.2.10 CFINDEX

4.2.10.1 Spidering a Web Site

BlueDragon now adds the ability to index/spider the web pages of a web site. `CFINDEX` has traditionally been used to index the content of files within a file system. If you indexed a directory of CFML files, you were indexing the source code, not the result of running the pages. Spidering a site actually executes the pages in the site and indexes the results.

Spidering is supported by way of a new value for the `TYPE` attribute: `website`. The `KEY` attribute is used to specify the URL of the site to be spidered, and it must contain the full URL of the web site to index, including `http://` or `https://`. (Spidering is also supported by way of the BlueDragon admin console page for creating collections.)

When spidering a web site, the URL provided in the `KEY` attribute indicates the starting page, which doesn't necessarily have to be the home page of the web site. For example, you could create separate search collections for sub-sections of a web site. The `KEY` value must specify a page; if you want to specify the default document for a directory, the URL must end with a "/". For example, the following are valid `KEY` values:

```
<CFINDEX TYPE="website" KEY="http://www.newatlanta.com/index.html">
```

```
<CFINDEX TYPE="website"
         KEY="http://www.newatlanta.com/bluedragon/index.cfm">

<CFINDEX TYPE="website" KEY="http://www.newatlanta.com/">

<CFINDEX TYPE="website" KEY="http://www.newatlanta.com/bluedragon/">
```

The following is not valid (no trailing "/"):

```
<CFINDEX TYPE="website" KEY="http://www.newatlanta.com">
```

The spidering process simply follows the links found in the starting page, processing any links that result in text or html files formats (`.cfm`, `.htm`, `.jsp`, `.asp`, `.aspx`, `.php`, etc.).

Note that it can be used to spider your own site or someone else's. Please use this feature responsibly when spidering the web sites of others. The spidering engine does not currently honor the `robots.txt` file exclusion standard, but this will be added in the future.

4.2.10.2 Asynchronous Index Processing

Index creation (spidering a web site or indexing a file collection or query resultset) can take a long time, so BlueDragon adds an optional `WAIT` attribute to `CFINDEX`, which takes a boolean value (such as `true` or `false`) that defaults to `true` (or `yes`).

If WAIT is `true`, processing of your CFML page will wait until the indexing operation is completed. If WAIT is set to false, processing continues immediately (as in ColdFusion) and the indexing is done on a background thread (a message is printed to `bluedragon.log` when the indexing operation is complete).

Be aware that by specifying `WAIT="false"`, it would be inappropriate to try in the same request to perform a `CFSEARCH` of the same collection. Setting `WAIT` to `false` is appropriate only on pages that kick off the indexing of, rather than search against, a collection.

The `WAIT` attribute only applies when the value of `ACTION` is `Update`, `Refresh`, or `Purge`. It is ignored for other `ACTION` values.

The `WAIT` attribute is also available for `CFCOLLECTION ACTION = "Create"`, with the same semantics described above.

4.2.11 CFINVOKE

The .NET edition of BlueDragon supports a `TIMEOUT` attribute when using `CFINVOKE` against a web service. The attribute specifies the maximum number of seconds to wait, before the invocation will fail with a runtime error.

4.2.12 CFLOCK

BlueDragon supports the full syntax and semantics of `CFLOCK`, but like CFMX does not always require the use of `CFLOCK` for all attempts to read or write variables in the Session,

Application, and Server scopes (as developers were expected to in CF 5 and before). BlueDragon manages concurrent access to these variable scopes internally.

As in CFMX, you would still use CFLOCK to prevent "race conditions" where two templates or concurrent users of a given template might both try to update the same persistent-scope variable at once.

4.2.13 CFMAIL

BlueDragon has added two new attributes to the CFMAIL tag to allow you to store sent mail in an IMAP server folder. In order to use these attributes you must first open a connection to the IMAP server using the CFIMAP tag (see below). These two new attributes are used in conjunction with the existing CFMAIL attributes to send an email message and have it saved on an IMAP server:

```
<CFMAIL  IMAPCONNECTION="name"
         IMAPFOLDER="fullfoldername"
         ...>
```

4.2.14 CFMAILPARAM

BlueDragon has added two attributes to the CFMAILPARAM tag to support mail file attachments:

```
disposition="disposition-type"

contentID="content ID"
```

The DISPOSITION attribute specifies how the file content is to be handled. Its value can be INLINE or ATTACHMENT. The CONTENTID attribute specifies the mail content-ID header value and is used as an identifier for the attached file in an IMG or other tag in the mail body that references the file content. This ID should be globally unique.

4.2.15 CFOBJECT

In BlueDragon for the Microsoft .NET Framework, a new type value of .net is supported for calling .NET objects. Use of type="java" is supported for backward compatibility and is synonymous. See *Integrating CFML with ASP.NET and the Microsoft .NET Framework* for more information.

4.2.16 CFOBJECTCACHE

CF5 introduced a new tag, CFObjectCache, with an available Action="clear" attribute/value pair used to clear all cached queries for all pages and applications. BlueDragon supports this tag with an additional new attribute, CacheDomain, which allows you to name a server whose cache you wish to flush. If you don't specify it, it will default to the on which the request is processing.

4.2.17 CFPROCPARAM

The CFPROCPARAM tag is used to pass parameters to a stored procedure. When nested within CFSTOREDPROC, the tags can pass the parameters either positionally or by name,

using the DBVARNAME attribute. In *BlueDragon for the Microsoft .NET Framework*, the DBVARNAME attribute is supported, while it's ignored in CFMX and the Java editions of BlueDragon. There are performance benefits in specifying the DBVARNAME attribute. See a more detailed discussion in the *BlueDragon 6.2.1 CFML Compatibility Guide*.

4.2.18 CFPROCESSINGDIRECTIVE SuppressWhiteSpace Attribute

While in CF5 and CFMX, CFPROCESSINGDIRECTIVE whitespace settings do not apply to templates included by CFINCLUDE or called as CFC methods or custom tags/CFMODULE, BlueDragon does propagate this setting into templates executed this way. This is generally an enhancement and a desirable feature, but it can cause problems with applications that are not expecting it.

If you have nested templates that you would not want to inherit this behavior, simply use a CFPROCESSINGDIRECTIVE tag within that nested template to set the desired value (yes or no) for that template. For a more complete discussion of white space handling in BlueDragon, see the *BlueDragon 6.2.1 CFML Compatibility Guide*.

4.2.19 CFQUERY

BlueDragon offers various enhancements regarding CFQUERY, with respect to query resultsets, query processing, and query caching.

4.2.19.1 DSN-less Connections Supported in .NET Edition

Although CFMX removed support for DSN-less connections (which was added in CF 5), the .NET edition of BlueDragon does support this feature (through the dbType="dynamic" and connectString attributes). With it, you can use a database in CFQUERY, CFINSERT, CFUPDATE, and CFSTOREDPROC without having to define a datasource in the BlueDragon Admin console.

An example of this feature follows, which queries an access database in a given absolute path to a directory:

```
<CFQUERY NAME="getemp" dbtype="dynamic"
ConnectString="DRIVER=Microsoft Access Driver (*.mdb);
DBQ=absolutpath\cfsnippets.mdb">
  SELECT      *
  FROM         Employees
</CFQUERY>

<cfdump var="#getemp#">
```

The following demonstrates using the text driver to be able to read a CSV file using CFQUERY, which turns it into a query result set. You name the directory in the dbq argument, then name the file in the SELECT … FROM clause:

```
<CFQUERY NAME="get" dbtype="dynamic"
connectstring="Driver={Microsoft Text Driver (*.txt;
*.csv)};Dbq=absolutepathtodirectory">
         SELECT      *
         FROM         test.csv
</CFQUERY>
```

```
<cfdump var="#get#">
```

For more information on using DSN-less connections in CFML, see the following:

http://livedocs.macromedia.com/coldfusion/5.0/Developing_ColdFusion_Applications/queryDB5.htm#1108627

http://www.atlantisnet.com/kb/archives/000011.html

For information on valid connection string values for various databases, see the following resource (though it shows ASP-oriented syntax, the information is useful):

http://www.carlprothman.net/Default.aspx?tabid=90

Again, the Java editions of BlueDragon do not support DSN-less connections.

4.2.19.2 Query Caching Enhancements

BlueDragon offers improved caching for CFQUERY tags which provides greater control over flushing cached queries. In ColdFusion, the only way to flush the query cache is with CFOBJECTCACHE. While BlueDragon supports that, it also adds new cacheName and action attributes to CFQUERY.

The optional cacheName attribute can be used to assign a unique name for cached CFQUERY results, to facilitate later flushing of that specific cache:

```
<CFQUERY NAME="users" DATASOURCE="mycompany" CACHENAME="usercache">
SELECT * FROM USERS
</CFQUERY>
```

Of course, the cachedWithin and cachedAfter attributes as implemented by ColdFusion can be used in conjunction with CACHENAME to add time-based caching.

In the above example, the CFQUERY results will be cached under the name "usercache" and when this query is run again the results from the cache will be used. You must specify a unique value for CACHENAME; if the same value for CACHENAME is specified for multiple CFQUERY tags, whether on the same or different CFML pages, the results in the cache will be overwritten.

A CFQUERY cache can be flushed using the new optional action attribute:

```
<CFQUERY ACTION="flushcache" CACHENAME="usercache">
```

All CFQUERY cached results can be cleared using a singe tag:

```
<CFQUERY ACTION="flushall">
```

A CFQUERY tag that uses the action attribute to flush a cache can appear on the same or a different CFML page from the CFQUERY tag that defines the cache. BlueDragon also supports the CFObjectCache tag introduced in CF5, used to clear all cached queries, and it

adds a new attribute (`CacheDomain`) for controlling cache clearing on multiple servers. See the discussion of `CFObjectCache` in 4.2.16 for more information.

4.2.19.3 Query `ExecutionTime` Variable

While BlueDragon supports the `cfquery.executiontime` variable, which was added to ColdFusion MX, it also provides an `executiontime` variable for in each query's resultset (so a query named `GetEmp` would have an available `GetEmp.executiontime` variable.)

4.2.19.4 New `PreserveSingleQuotes` Attribute

BlueDragon, like ColdFusion, automatically "escapes" single-quote characters within CFML variables used to create SQL statements within `CFQUERY` tags. For example, the following SQL will work correctly because the single quote within the string, "O'Neil", will be escaped before being passed to the database:

```
<CFSET EmployeeName="O'Neil">

<CFQUERY NAME="employees" DATASOURCE="MyCompany">
SELECT * FROM Employees
WHERE LastName = '#EmployeeName#'
</CFQUERY>
```

If you have code where this behavior is undesirable, you can change it with the available `PreserveSingleQuotes()` function, which when used against a variable within a `CFQUERY` will stop the automatic escaping of quotes. For example, consider an instance when a variable used in SQL (such as an incoming form field or other variable) may have a list of values presented as a single-quote delimited list. Escaping single-quotes in this case will produce incorrect results:

```
<CFSET NameList=" 'Peterson','Smith' ">

<CFQUERY NAME="employees" DATASOURCE="cfsnippets" >
SELECT * FROM Employees
WHERE LastName IN (#PreserveSingleQuotes( NameList )#)
</CFQUERY>
```

As an enhancement, if you would like all variables within a query to automatically preserve any single quotes, BlueDragon 6.2 added a new `PreserveSingleQuotes` attribute that can be specified on the `CFQUERY`. The new attribute simply applies a global change of behavior in SQL processing than might otherwise be achieved with one or more uses of the `PreserveSingleQuotes()` function; for example:

```
<CFSET NameList=" 'Peterson','Smith' ">

<CFQUERY NAME="employees" DATASOURCE="cfsnippets"
         PRESERVESINGLEQUOTES="Yes">
SELECT * FROM Employees
WHERE LastName IN (#NameList#)
</CFQUERY>
```

4.2.20 CFQUERYPARAM

BlueDragon supports use of CFQUERYPARAM within cached queries (using CFQUERY's CACHEDWITHIN attribute, for instance). ColdFusion does not. The benefit here is that cached queries can benefit from the enhanced security and performance features enabled by CFQUERYPARAM.

4.2.21 CFSEARCH

BlueDragon adds predefined RecordCount and ColumnList columns to the results of a CFSEARCH, with values identical to those returned in traditional query result sets.

On the other hand, a new variable has been added, SearchCount, which reflects the total number of items in the search result, irrespective of any MaxRows value that may have been specified in the CFSEARCH. This makes it possible to better manage the resultset, such as when providing a paging interface.

CFSEARCH also supports an additional attribute, MinScore, which accepts a number between 0 and 1.0 and will return only those results with a score greater than this. By default, all query results are returned regardless of score.

Finally, BlueDragon adds SORT, SORTDIRECTION and SORTTYPE attributes to support sorting of search results.

4.2.22 CFSET (Multi-dimensional arrays)

ColdFusion limits multi-dimensional arrays to three dimensions; BlueDragon does not impose any limit. For example the following tags are supported by BlueDragon, but will generate errors in ColdFusion:

```
<CFSET myArray=ArrayNew(8)>
<CFSET myArray[2][3][4][4][2][3][4][4]="BlueDragon">
```

4.2.23 CFXML

BlueDragon offers additional functionality with respect to case sensitivity, node processing, and array handling. See section 5.4 for more information.

4.3 New CFML Tags

This section lists new CFML tags that are unique to BlueDragon.

4.3.1 CFASSERT

CFASSERT is a new CFML tag introduced by BlueDragon that can be used as a testing tool to enhance the reliability and robustness of your applications. The concept of using *assertions* is frequently found in more advanced languages, and it's critical to effective *unit-testing* of your applications. Complete discussion of the benefits and uses of assertions is beyond the scope of this manual, but a brief explanation follows.

CFASSERT (and its corresponding assert() function discussed in section 4.5.1) takes an expression that is expected to evaluate to a Boolean result (true or false). CFASSERT has

no attribute, rather on simply provides the expression to test within the tag, as in <CFASSERT *someexpression*>. An example is <CFASSERT testvar is expectedval>.

An assertion tests throw an exception if the result is false but do nothing if the result is true. They are also ignored if assertions have not been enabled in the BlueDragon Administration console, as discussed at the end of this section. The intention is that you can place these assertions in your code to help ensure that some expected state of the application is indeed occurring as expected. More accurately, they cause failure if the state is not as expected.

4.3.1.1 Understanding Assertions

A typical use is during testing, when you expect that a given variable will have a given value (or perhaps a range of values), perhaps after calling a custom tag, UDF, or CFC method. Another example is when you want to test the mere existence of a given variable (such as an expected session or application variable).

The difference between assertions and using a CFIF is that the CFIF is intended to control the flow of the logic, executing code depending on a condition that may or may not be true. An assertion test is intended to simply throw an error is the expected condition is not (and never should be) true. In other words, the CFIF test handles expected conditions, while the assertion flags unexpected conditions.

An assertion *could* be surrounded by a CFTRY to catch and handle the error that will be thrown, or the error can be allowed to pass up to the caller of the code throwing the exception. It could also be left to be handled by any CFERROR or site-wide error handler, or if unhandled will simply result in a BlueDragon runtime error. (There is a body of opinion in the industry that suggests that assertion failures should not be caught at runtime, or at least ought not be used to alter the flow of processing and allow execution to continue. These proponents expect that an assertion failure should result in a cease of processing because an unexected error has occurred.)

4.3.1.2 Controlled By Admin Console Setting

Execution of CFASSERT (and the assert() function) is controlled by the Enable Assertions setting on the Debug Settings page of the BlueDragon Administration console. After changing this setting, **you must restart the server** for it to take effect.

If the "Enable Assertions" option is checked, then CFASSERT tags and assert() functions are enabled, otherwise they are not and are simply ignored when encountered. This means that assertions can be left in code placed into production, where the Admin setting would be set to disable assertions. There is no cost to assertions existing in code when they are disabled. Assertions are supported in all editions of BlueDragon.

4.3.2 CFBASE

CFBASE is a CFML tag introduced by BlueDragon that is primarily intended for use in *BlueDragon for J2EE Servers*. The CFBASE tag can be used to create an absolute URL that serves as the base for resolving relative URLs within a CFML page (such as in IMG

tags). The absolute URL created by the CFBASE tag includes the J2EE web application context path. See the document *Deploying CFML on Application J2EE Application Servers* for a detailed discussion of CFBASE.

4.3.3 CFCACHECONTENT

CFCACHECONTENT is a new tag introduced by BlueDragon to cache blocks of html for a given time without having to regenerate it every time. While it may seem a melding of CFSAVECONTENT and CFCACHE, this tag is more powerful as it can cache not only in memory but to a database table as well.

Additionally, flushing of the cache is enhanced as well, with two available attributes: CACHENAME and GROUP. GROUP allows you to gather together cached elements and treat them as a single logical unit. For example, you may wish to cache various elements for a given user, but should that user change something, you can flush and clear all their cached elements in a single operation.

```
<CFCACHECONTENT
        ACTION = "action"
        CACHENAME = "name of cache"
        GROUP = "group name">
        CACHEDWITHIN = "timeout for cache"
…some content
</CFCACHECONTENT>
```

The following table lists the CFCACHECONTENT tag attributes.

Attribute	Description
Action	Required. Operation to perform; possible operations are CACHE, FLUSH, FLUSHGROUP, FLUSHALL, and STATS: CACHE – Default value. Caches the data within the tag FLUSH - Flushes cached items designated using the given CACHENAME. All items cached with this CACHENAME are removed FLUSHGROUP - Flushes cached items designated using the given GROUP. All items cached with this GROUP are removed FLUSHALL - Flushes the entire cache, including all data in the database cache STATS - This will return a structure, CFCACHECONTENT, with two fields, MISSES and HITS, which detail the number of times requests have found data using the cache and the number of times requests called for a cached result to be generated. This gives you an indication of how efficient your cache is performing RESET - resets the HIT and MISS counts to zero, for all GROUP/CACHENAMEs or individual ones.
CacheName	Required for ACTION=CACHE. The name to be given for the item being cached, which should be unique across all GROUPs.
Group	Optional. A name given to group cached results together. It defaults to the name of the server, as determined in cgi.server_name.
CachedWithin	Optional. Used with ACTION=CACHE. The maximum time to maintain this cached result. If the cached data is found to be older than this when a request attempts to use the cached result, the cached content will be regenerated. Specified using #CreateTimeSpan()#.

This tag requires an end tag.

4.3.3.1 Database Persistence of Cached Data

You can cause cached data to be persisted to a database, to support caching data across server restarts, by declaring a datasource in the bluedragon.xml file using:

```
<system>
      <cfcachecontent>
            <datasource>datasourcename</datasource>
            <total>5</total>
      </cfcachecontent>
</system>
```

The `TOTAL` value specifies the number of cached items that will be persisted to memory before being paged out to the database. A LeastRecentlyUsed algorithm is used for paging out data. When a flush occurs, the items in the database are also removed. A table, `LRUCACHE`, will be automatically created in the database if one is not already present. Data persisted in the database will be maintained over server-restarts. The table has a DATETIME field associated with cached items which can be used to manually process cached items.

The following examples illustrates caching content. This will cache data for 4 minutes

```
<CFCACHECONTENT CACHENAME="abc"
      CACHEDWITHIN="#CreateTimeSpan(0,0,4,0)#">
   <CFOUTPUT>#now#</CFOUTPUT>
</CFCACHECONTENT>
```

If this code was processed on multiple servers but cached to the same database, the cached results would be unique to each server (because the GROUP attribute was allowed to default to the current servername).

To flush this cache manually we would call:

```
<CFCACHECONTENT ACTION="flush" CACHENAME="abc"/>
```

Or to flush all cached data for the current server, regardless of `cachename`, use:

```
<CFCACHECONTENT ACTION="flushgroup"/>
```

To flush all cached data for all servers (groups), regardless of `cachename`, use:

```
<CFCACHECONTENT ACTION="flushall"/>
```

4.3.4 CFCONTINUE

The new `CFCONTINUE` tag works similarly to `CFBREAK` in that it can only be used within the body of a `CFLOOP` tag (it cannot be used in a `CFOUTPUT QUERY` loop.) `CFBREAK` terminates execution of the current iteration of the `CFLOOP` body and continues execution *after* the closing `CFLOOP` tag. `CFCONTINUE`, on the other hand, terminates execution of the current iteration of the `CFLOOP` body and continues execution *of the next iteration* of the `CFLOOP` body from the opening `CFLOOP` tag.

4.3.5 CFDEBUGGER

CFDEBUGGER is a CFML tag introduced by BlueDragon that adds a powerful new weapon in CFML debugging. In simple terms, it writes a trace to a log file indicating each CFML line of code that's been executed.

Consider the following simplified example of its use:

```
<CFDEBUGGER LOGFILE="#expandpath('trace.log')#">
<CFSET name="bob">
```

This two-line template will create an entry in a file named trace.log (as indicated in the LOGFILE attribute, which in this case will store the logfile in the same directory as the page running the tag). In this case, the log file will include the following info:

```
#0: CFDEBUGGER trace started @ 19/Aug/2003 15:03.19
#1: active.file=C:/Inetpub/wwwroot/regression/cfdebugger.cfm
#2: tag.end=CFDEBUGGER; L/C=(1,1);
File=C:/Inetpub/wwwroot/regression/cfdebugger.cfm
#3: tag.start=CFSET; L/C=(2,1);
File=C:/Inetpub/wwwroot/regression/cfdebugger.cfm
#4: tag.end=CFSET; L/C=(2,1);
File=C:/Inetpub/wwwroot/regression/cfdebugger.cfm
#5: file.end=C:/Inetpub/wwwroot/regression/cfdebugger.cfm
#6: Session Ended
```

Note that it indicates the time the template was run and the template's name. More important, the trace shows, for each CFML tag it encounters, its start and end lines in the given template. Beware that the log could accumulate a large amount of information, as it starts logging once its set for the remainder of the request, and it appends data for all subsequent requests executed, until the tag is removed.

Note as well that if you don't specify a path for the file (or use a relative path), the destination for the logfile will be vary depending on the version of BlueDragon you're using. Using an absolute path, or expandpath() as above, will be most straight-forward.

For more information on the CFDEBUGGER tag, see the November 2003 ColdFusion Developers Journal article on the subject:

http://coldfusion.sys-con.com/read/42101.htm

4.3.6 CFFORWARD

CFFORWARD is a tag introduced by BlueDragon that allows you to do a "server-side redirect" to another CFML page, or in some BlueDragon editions a Java servlet or JavaServer Page (JSP), or in the .NET edition an ASP.NET page. In a "client-side redirect," which is done using the CFLOCATION tag, a response is sent to the browser telling it to send in a new request for a specified URL. In contrast, CFFORWARD processing is handled completely on the server.

The advantages of CFFORWARD over CFLOCATION are:

- There is no need for extra messaging between the server and browser
- Variables in the REQUEST scopes are available to the target of the CFFORWARD tag

To pass data from other scopes (like FORM or URL), simply copy them to the Request scope before calling CFFORWARD.

CFFORWARD has a single attribute, page, which specifies the URI of the target page. Paths that start with "/" start at the document root directory of the web application or web server; paths that don't start with "/" are relative to the current CFML document:

```
<CFFORWARD PAGE="/nextpage.cfm">
<CFFORWARD PAGE="nextpage.jsp">
<CFFORWARD PAGE="nextpage.aspx">
```

Like CFLOCATION, processing of the current page is terminated as soon as the CFFORWARD tag is executed.

4.3.7 CFIMAGE

CFIMAGE is a tag introduced by BlueDragon that allows you to modify an existing GIF or JPEG image file to produce a new image file that is resized and/or has a text label added to the image. Variables returned by this tag provide information about the new image file.

The following table lists the CFIMAGE tag attributes.

Attribute	Description
SrcFile	Required. The file name of the source image file that is to be modified. Can be either a full physical path or a relative path (see the URIDirectory attribute).
DestFile	Required if ACTION=EDIT, Optional if ACTION=INFO. The file name of the new image file to be created by the CFIMAGE tag. Can be either a full physical path or a relative path (see the URIDirectory attribute).
Action	Optional. The action to be taken by the CFIMAGE tag. The value INFO populates the CFIMAGE variables with information about the image file specified by the srcFile attribute without modifying the image. The value of EDIT creates a new image file by resizing and/or adding a text label to the source image file. Defaults to EDIT.
Type	Optional. The image file type, either GIF or JPEG. If this attribute is not specified, the CFIMAGE tag attempts to determine the image type based on the file name extension.
Width	Optional. The width of the new image, can be specified either in pixels or as a percentage of the source image width. Defaults to "100%".
Height	Optional. The height of the new image, can be specified either in pixels or as a percentage of the source image height. Defaults to "100%".
FontSize	Optional. An integer value that specified the font size of the text label to be added to the image. Defaults to 12.
FontColor	Optional. Specifies the font color of the text label to be added to the image. Accepts any value that is valid for use in the FONT tag. Defaults to "black".
Text	Optional. The text label to add to the image.
Position	Optional. The position of the text label to add to the image; valid valued are "north" and "south". Defaults to "south".
NameConflict	Optional. Indicates the behavior of the CFIMAGE tag when the file specified by destFile

	already exists. Valid values are ERROR, which generates a runtime error; SKIP, which causes the CFIMAGE tag to do nothing without generating an error; OVERWRITE, to overwrite the existing image; and, MAKEUNIQUE, which causes CFIMAGE to create a new unique file name for the new image file. Defaults to ERROR.
URIDirectory	Optional. If YES, relative paths specified in `srcFile` and `destFile` are calculated from the web server document root directory. If NO, relative paths are calculated as relative to the current file. Defaults to NO.

The following table lists the variables returned by the `CFIMAGE` tag.

Variable	Description
CFIMAGE.SUCCESS	Contains the value TRUE or FALSE to indicate whether image processing was successful.
CFIMAGE.ERRORTEXT	If processing was unsuccessful, contains a text message describing the error.
CFIMAGE.WIDTH	For ACTION=EDIT, the width in pixels of the new image. For Action=INFO, the width in pixels of the image.
CFIMAGE.HEIGHT	For ACTION=EDIT, the height in pixels of the new image. For Action=INFO, the height in pixels of the image.
CFIMAGE.PATH	The full physical path to the image.
CFIMAGE.NAME	The name of the new image file.
CFIMAGE.FILESIZE	The size in bytes of the new image file.

The following example displays two images – the original image “picture.gif”, and the processed image “newPicture.gif”.

```
<cfimage action="edit"
         srcfile="picture.gif"
         destfile="newPicture.gif"
         uridirectory="yes"
         text="Copyright 2003"
         width="50%"
         height="50%"
         fontsize=20
         fontcolour="violet"
         position="SOUTH"
         nameconflict="overwrite">

<img src="picture.gif">
<img src="newPic.gif">
```

The following example displays information about an existing image file named “picture.jpg”.

```
<cfimage action="info" srcfile="picture.jpg">

<cfoutput>
Success : #cfimage.success# <BR>
Dimensions : #cfimage.width# x #cfimage.height#  <BR>
Path : #cfimage.filepath# <BR>
Name : #cfimage.filename# <BR>
```

```
Size (bytes) : #cfimage.filesize# <BR>
Error message : #cfimage.errortext# <BR>
</cfoutput>
```

While CFIMAGE can read a PNG file (with TYPE="GIF"), it cannot write PNG files through any means.

4.3.8 CFIMAP

The CFIMAP tag allows you to interact with both IMAP and POP mail servers (CFIMAP may be used instead of CFPOP to interact with POP mail servers). Generally, the sequence of steps to interact with a mail server is:

1. Open a connection to the mail server (OPEN action).
2. Get a list of folders from the mail server (LISTALLFOLDERS action).
3. Get a list of messages within a specific folder (LISTMAIL action).
4. Perform actions with specific messages (READMAIL, MARKREAD, DELTEMAIL, and MOVEMAIL actions).
5. Perform actions with folders (DELETEFOLDER and RENAMEFOLDER actions).
6. Close the connection (CLOSE action).

Each of these steps is described below.

4.3.8.1 Opening a Connection

Before performing actions such as reading mail, you must first open a connection with the IMAP or POP server. Specify a value of OPEN for the action attribute. The name specified for the connection attribute will be used to refer to this connection when performing subsequent actions with the IMAP or POP server, such as reading mail.

```
<CFIMAP ACTION="OPEN"
        SERVICE="POP3 or IMAP"
        CONNECTION="name"
        SERVER="mail.yourdomain.com"
        USERNAME="username"
        PASSWORD="password">
```

Two variables are always returned by the CFIMAP tag:

IMAP.SUCCEEDED – "true" or "false" depending on whether the previous action succeeded

IMAP.ERRORTEXT – an error message, if the previous action failed

Please note that the Connection attribute is intended to be used to create a name to uniquely distinguish this connection from any other. The value does not become a variable that can be accessed in any way. Further, to distinguish CFIMAP calls from each other, you can use a variable for the name, such as #session.sessionid#.

4.3.8.2 Closing a Connection

An IMAP or POP server connection can be closed by specifying ACTION="CLOSE" and the name of the connection:

```
<CFIMAP ACTION="CLOSE"
        CONNECTION="name">
```

After closing a connection, any attempts to use the connection will generate an error.

4.3.8.3 Listing Mailbox SubFolders

Use ACTION="LISTFOLDER" to get a list of subfolders under a given folder, or all the top level folders on the IMAP or POP server:

```
<CFIMAP  ACTION="LISTFOLDER"
         CONNECTION="name"
         FOLDER="fullname/"
         NAME="queryname">
```

If a FOLDER name is used, note that it must be a fullname. See the discussion of fields returned in the query structure, later in this section. Additionally, note that you may need to use a closing slash at the end of the folder name, depending on the mail server.

Note that the FOLDER attribute is optional. If note used, the will return all the folders (but not subfolders) at the root level. See Action="ListAllFolders" in the next section to list all folders and subfolders.

The folder list is returned in a Query structure with the name you specified in the NAME attribute. The fields of the Query structure are:

FULLNAME – the full path to the folder (used to retrieve folder message info)

NAME – shortcut name to the folder

TOTALMESSAGES – total messages this folder is holding

UNREAD – total unread messages in this folder

NEW – total new messages in this folder

The FULLNAME field is used for making subsequent calls to folders with other CFIMAP action parameters.

4.3.8.4 Listing All Mailbox Folders

Use ACTION="LISTALLFOLDERS" to get a list of folders on the IMAP or POP server:

```
<CFIMAP  ACTION="LISTALLFOLDERS"
         CONNECTION="name"
         NAME="queryname">
```

See the discussion in the previous section about the query structure returned in the variable specified in the NAME attribute.

4.3.8.5 Listing Mail Messages

You can retrieve high-level information about the messages within a folder by specifying ACTION="LISTMAIL"; this action does not retrieve the message bodies. To read a message body you must first get an email message ID using the LISTMAIL action and then specify the message ID in the READMAIL action as described in the next section.

The folder attribute must contain the name of a folder as contained in the FULLNAME field of the Query structure returned by ACTION="LISTALLFOLDERS".

```
<CFIMAP ACTION="LISTMAIL"
        CONNECTION="name"
        FOLDER="fullname"
        NAME="queryname">
```

The message information is returned in a Query structure with the name you specified in the name attribute. The fields of this Query structure are:

SUBJECT – subject line of the mail message

ID – unique ID of this mail message (used to retrieve the message body)

RXDDATE – the date this mail message was received

SENTDATE – the date this mail message was sent

FROM – address structure (see below)

TO – array of address structures (see below)

CC – array of address structures (see below)

BCC – array of address structures (see below)

SIZE – size in bytes of this mail message

LINES – number of lines of this mail message

ANSWERED – boolean flag if this mail message has been answered

DELETED – boolean flag if this mail message has been deleted

DRAFT – boolean flag if this mail message is an unsent draft

FLAGGED – boolean flag if this email has been flagged

RECENT – boolean flag if this email is recent

SEEN – boolean flag if this email has been seen (read)

Internet email addresses are stored as structures with two fields:

NAME – name of the person

EMAIL – email address of the person

The TO, CC, and BCC fields contain arrays of these structures.

4.3.8.6 Reading a Mail Message

You can read a specific email message by specifying `ACTION="READMAIL"`, the folder name, and the email message ID as returned by the `LISTMAIL` action:

```
<CFIMAP ACTION="READMAIL"
        CONNECTION="name"
        FOLDER="foldername"
        MESSAGEID="messageid"
        ATTACHMENTSURI="uritofolder"
        NAME="messagename">
```

This action will retrieve the given message and fill in a structure variable containing information regarding the retrieved email message. In addition to this, should the message have any attachments, you specify the URI of the folder you wish the email attachment to be stored in. Note this is a URI and not a real directory. The fields of the returned structure are:

`SUBJECT` – subject of the email

`ID` – unique ID to this mail

`RXDDATE` – the date this mail was received

`SENTDATE` – the date this email was sent

`FROM` – address structure (see below)

`TO` – array of Address Structures (see below)

`CC` – array of Address Structures (see below)

`BCC` – array of Address Structures (see below)

`SIZE` – size in bytes of this email

`LINES` – number of lines of this email

`ANSWERED` – boolean flag if this email has been answered

`DELETED` – boolean flag if this email has been deleted

`DRAFT` – boolean flag if this email is a draft

`FLAGGED` – boolean flag if this email has been flagged

`RECENT` – boolean flag if this email is recent

`SEEN` – boolean flag if this email has been seen

`BODY` – array of Body structures (see below)

The body of the email is treated with some consideration. Due to the various properties a MIME type email message can have, each element in the array is effectively the MIME part that was transmitted with the email.

`MIMETYPE` – the MIME type of this part

CONTENT – the content of this email if not an attachment

FILE – boolean flag to indicate if there is a file attached

FILENAME – the name of the attached file

URL – the URI to the saved file

SIZE – the size of the saved file

To loop through the message body array elements, you might use the following code:

```
<CFOUTPUT>
<CFLOOP INDEX="X" FROM="1" TO="#ArrayLen(msg.body)#">

     <br>#msg.body[X].mimetype#

     <br>#msg.body[X].file#

     <br>#msg.body[X].content#

   </CFLOOP>
</CFOUTPUT>
```

This action will not overwrite any existing files; instead, it will create a unique name for it.

4.3.8.7 Marking Mail Messages as "Read"

You can mark messages as having been read by specifying ACTION="MARKREAD", a folder name, and a list of message IDs:

```
<CFIMAP ACTION="MARKREAD"
        CONNECTION="name"
        FOLDER="toplevelfoldername"
        MESSAGELIST="list of IDs">
```

The message list is either a single message ID or a comma-separated list of IDs.

4.3.8.8 Deleting Mail Messages

You can delete messages by specifying ACTION="DELETEMAIL", a folder name, and a list of message IDs:

```
<CFIMAP ACTION="DELETEMAIL"
        CONNECTION="name"
        FOLDER="toplevelfoldername"
        MESSAGELIST="list of IDs">
```

The message list is either a single message ID or a comma-separated list of IDs.

4.3.8.9 Setting Message Flags

You can set the status of various aspects of a message (such as read, seen, answered), using ACTION="SETFLAGS", along with a folder name, a message ID, and any of the

following flagname attributes indicated as a Boolean value: ANSWERED, DELETED, DRAFT, FLAGGED, RECENT, SEEN:

```
<CFIMAP ACTION="SETFLAGS"
        CONNECTION="name"
        FOLDER="toplevelfoldername"
       MESSAGEID="messageid"
        flagname=”boolean”>
```

4.3.8.10 Moving Mail Messages between Folders

You can move a list of messages from one mail server folder to another by specifying ACTION=”MOVEMAIL”:

```
<CFIMAP ACTION="MOVEMAIL"
        CONNECTION="name"
        FOLDER="toplevelfoldername"
        DESTFOLDER="toplevelfoldername"
        MESSAGELIST="list of IDs">
```

The message list is either a single message ID or a comma-separated list of IDs.

4.3.8.11 Creating a Folder

Specifying ACTION=”CREATEFOLDER” will create a folder on the mail server:

```
<CFIMAP ACTION="CREATEFOLDER"
        CONNECTION="name"
        FOLDER="fullfoldername">
```

The folder name is the complete path to the folder.

4.3.8.12 Deleting a Folder

Specifying ACTION=”DELETEFOLDER” will delete a folder from the mail server, including all of its contents (mail messages):

```
<CFIMAP ACTION="DELETEFOLDER"
        CONNECTION="name"
        FOLDER="fullfoldername">
```

The folder name is the complete path to the folder. This is a very powerful action and should be used with extreme care, as it can remove all messages and folders from the mail server.

4.3.8.13 Renaming a Folder

Specifying ACTION=”RENAMEFOLDER” will rename a folder on the mail server:

```
<CFIMAP ACTION="RENAMEFOLDER"
        CONNECTION="name"
        OLDFOLDER="fullfoldername"
        NEWFOLDER="fullfoldername">
```

The folder name is the complete path to the folder.

4.3.8.14 Sending Mail Messages

Sending email messages is done using the `CFMAIL` tag, not `CFIMAP`. However, BlueDragon has added two new attributes to the `CFMAIL` tag to allow you to store sent mail in an IMAP server folder. See the section on the `CFMAIL` tag for details.

4.3.9 CFMAPPING

`CFMAPPING` is a new tag introduced by BlueDragon to assist in creating mappings (for use with tags like CFINCLUDE, CFMODULE, and CFC invocation) at a page- or application-level. (BlueDragon also supports defining global mappings in the BlueDragon admin console.)

CFMAPPING requires two attribute, one of which is `LOGICALPATH`, and the other can be either `DIRECTORYPATH` or `RELATIVEPATH.`

4.3.9.1 Using DirectoryPath for Absolute Paths

`DIRECTORYPATH` must specify a full physical path. An example that corresponds to a similar kind of setting in the Admin console is:

```
<cfmapping logicalpath="/mypath" directorypath="C:\mymappedpath">
```

4.3.9.2 Using RelativePath for Relative Paths

The `RELATIVEPATH` option provides a benefit with no corresponding setting in the Admin console, in that it permits specification of a path that's relative rather than absolute. If the `RELATIVEPATH` value starts with "/", it's interpreted as being relative to the application root directory. For example:

```
<cfmapping logicalpath="/map1" relativepath="/WEB-INF/map1">
```

For BlueDragon Server JX, the application root is the web server document root; for BlueDragon/J2EE, this the J2EE web application root; for BlueDragon.NET, this is the ASP.NET application root.

If the `RELATIVEPATH` value does not start with "/", it's interpreted as being relative from the current document directory. For example:

```
<cfmapping logicalpath="/map2" relativepath="../../map2">
```

4.3.9.3 Other Information

All three attributes, `DIRECTORYPATH`, `LOGICALPATH`, and `RELATIVEPATH` accept variable expressions as well as string constants.

When the `CFMAPPPING` tag is rendered BlueDragon will verify that the specified `DIRECTORYPATH` actually exists, is a directory (and not a file), and that BlueDragon can read from that directory. If any of these checks fail, a CFML runtime exception will be thrown.

Mappings specified by CFMAPPING will exist for the duration of the request and survive across CFINCLUDEs, custom tag calls, CFC method calls, etc. You can put a CFMAPPING tag in any page, including Application.cfm.

The CFMAPPING tag will override mappings configured via the admin console.

4.3.10 CFPAUSE

CFPAUSE is a new tag introduced by BlueDragon to assist in debugging CFML pages. The CFPAUSE tag allows you to pause the execution of a page for a specified number of seconds. The interval attribute is required and must specify an integer value:

```
<CFPAUSE INTERVAL="seconds to pause">
```

4.3.11 CFTHROTTLE

CFTHROTTLE is a new tag introduced by BlueDragon to help respond to requests that are coming in too quickly from a given host/client. Rogue spiders, bad software etc can cripple a server with repeated requests. This tag is designed to track such requests in a given a window of time and give you the opportunity to deny or redirect their requests.

```
<CFTHROTTLE
        ACTION = "action"
        TOKEN = "name of item to track"
        HITTHRESHOLD = "number of hits"
        HITTIMEPERIOD = "time period to count hits"
        MINHITTIME = "time between each request">
... some operation
</CFTHROTTLE>
```

The following table lists the CFTHROTTLE tag attributes.

Attribute	Description
Action	Required. Available values are: THROTTLE – Default value. Enable throttle processing FLUSH - Flushes entire historical table of throttle processing STATUS - Returns a CFML variable, CFTHROTTLE, which is an array of structures detailing each item that is currently being tracked by CFTHROTTLE. Allows creation of a status page. SET - Sets the size of the window for tracking. Defaults to 100, for the last 100 items to be tracked; can be overriden by passing a new value using the optional HISTORY attribute.
History	Optional. Used with ACTION=SET. Sets the size of the window for tracking. Defaults to 100, for the last 100 items being tracked.
Token	Optional. Used with ACTION=THROTTLE. The string used to track repeated requests. In most instances you will track the client ip address, and this is the default if not specified. You may choose to use the CGI.HTTP_USER_AGENT variable to track, for instance, when requests from certain search engines are visiting your site too often in a given period of time.
HitThreshold	Optional. Used with ACTION=THROTTLE. The maximum number of times a unique TOKEN value can be used by requests being monitored in the time period specified by HITTIMEPERIOD. If the number is exceeded, then the request is flagged as excessive and a candidate to be throttled (see discussion of resulting THROTTLE scope below). Defaults to 5.
HitTimePeriod	Optional. Used with ACTION=THROTTLE. The time period, expressed in milliseconds, where if a successive number of requests (determined by HITTHRESHHOLD) are received sharing

	the given TOKEN, the request will be deemed excessive. Defaults to 10000 (10 seconds).
MinHitTime	Optional. Used with ACTION=THROTTLE. The time period, expressed in milliseconds, where if successive requests (regardless of the TOKEN) are received, the request will be deemed excessive. Defaults to 500 (one half second).

This tag does not require an end tag. `CFTHROTTLE` would typically be used in an `Application.cfm` file to detect and enable handling of a request when it has been deemed excessive.

After using `ACTION=THROTTLE` a special structure, `CFTHROTTLE`, is returned containing a number of variables to aid in tracking a potential rogue user. If the boolean `CFTHROTTLE.THROTTLE` is `TRUE`, then the server has detected an excessive request that is a candidate for throttling.

The `CFTHROTTLE` tag will not throttle the connection itself. Instead, it helps detect such requests, based on the attributes described above. Code within the CFTHROTTLE tag is then processed to handle the excessive request.

The following example illustrates a use of this tag:

```
<CFTHROTTLE>
<CFIF cfthrottle.throttle EQ true>
   <CFHEADER STATUSCODE="503"
      STATUSTEXT="Try backing off the time between requests">
   <H1>503 Server is very busy.  Try later</H1>
   <CFABORT>
</CFIF>
<!--- continue processing --->
```

In this case, since all the defaults were taken for `CFTHROTTLE`'s attributes, it would detect if a request came from the same IP address more than 5 times in a 10 second period, or if requests came in more often than every half second.

Other variables available in the `CFTHROTTLE` result structure include

- `HITCOUNT` - number of times the detected token has been found in requests in given the time period
- `TOTALHITS` - total number of times the token has been detected throughout the server's life time (or since the history has been flushed)
- `LASTHIT` - number of milliseconds since the token was last detected
- `AGE` - the total time since this token has been tracked

4.3.12 CFXMLRPC

`CFXMLRPC` is a new tag introduced by BlueDragon to easily and quickly invoke remote XML-RPC services:

```
<CFXMLRPC
```

```
SERVER = "url of server"
METHOD = "method name"
PARAMS = "array of params">
```

Note that if you're intending to invoke remote Web Services, you should use the CFINVOKE tag (or CFOBJECT/createObject) instead. More information on XMLRPC can be found in the following resources:

http://www.xml-rpc.com/

http://weblog.masukomi.org/writings/xml-rpc_vs_soap.htm

http://www.oreilly.com/catalog/progxmlrpc/chapter/ch03.html

The following table lists the CFXMRPC tag attributes.

Attribute	Description
Server	Required. The full URL to the XML-RPC server
Method	Required. The method you wish to invoke on at the given SERVER
Params	Required. A CFML Array containing all the parameters for the given METHOD. All complex types will be converted to the types laid out in the XML-RPC specification.

This tag does not require an end tag.

Upon completion, CFXMLRPC will create a variable, XMLRPC, as a structure with at least 1 key, SUCCESS. If the operation was successful, then SUCCESS will be set to TRUE and the RESULT key will contain a CFML structure of the data that was returned. If the operation was unsuccessful, SUCCESS will be set to FALSE, with ERROR reporting the error message.

The following example illustrates calling an XML-RPC server:

```
<CFXMLRPC SERVER="http://servername/filename.ext"
METHOD="methodname" PARAMS="#myarray#">
```

CFXMLRPC relies on the underlying (and provided) Apache XML-RPC library.

4.3.13 CFZIP and CFZIPPARAM

CFZIP is a new tag introduced by BlueDragon to assist in creating, extracting, and listing the contents of compressed (zip) files. The optional CFZIPPARAM can be used as described in the next section.

CFZIP ACTION="create" will create a zip file comprised either of the file/files named in the SOURCE attribute of CFZIP, or those named in the SOURCE attribute of the optional CFZIPPARAM (multiple CFZIPPARAMs are permitted).

CFZIP ACTION="extract" will extract the zip file contents to the named DESTINATION directory.

CFZIP ACTION="list" will return a VARIABLE with a query resultset representing the zipfile contents (similar to that returned by CFDIRECTORY).

The following table lists the CFZIP tag attributes.

Attribute	Description
ZipFile	Required. The name of the zipfile to create, extract, or list.
Source	Required if ACTION=CREATE. The path to a file, or directory in which to find files, to be added to the zipfile. (To name multiple directories or files, use CFZIPPARAM.)
Action	Optional. The action to be taken by the CFZIP tag. The value CREATE creates a zip file from the file(s) specified by the source attribute. The value of EXTRACT extracts a file(s) from a zipfile to the named DESTINATION. The value of LIST creates a query resultset describing the contents of the zip file. Defaults to CREATE.
Recurse	Optional. Used with ACTION=CREATE. Indicates whether the subdirectories in the SOURCE directory be included. Defaults to YES.
Filter	Optional. Used with ACTION=CREATE. Specify a filter for the given SOURCE directory e.g. *.gif (similar to the cfdirectory filter attribute)
Timeout	Optional. Used with ACTION=CREATE or EXTRACT. An exception will be thrown if the time taken to create/extract the zipfile exceeds this. Defaults to 60 seconds.
CompressionLevel	Optional. Used with ACTION=CREATE. A numeric in the range 0-9 (inclusive) that specifies the level of compression with 0 meaning no compression and 9 being the maximum. Defaults to 8.
NewPath	Optional. Used with ACTION=CREATE. If the item specified in SOURCE is a file then this attribute can be used to specify a replacement path. If the SOURCE item is a directory then this is ignored.
Prefix	Optional. Used with ACTION=CREATE. A prefix that will be prepended to the path of files in the created zipfile
Variable	Required if ACTION=LIST. Ignored for other actions. The name of the variable where the result of the zipfile contents listing will appear (as a query result set)
Destination	Required if ACTION=EXTRACT. Ignored for other actions. The directory to which the zip file will be extracted.
Flatten	Optional. Used with ACTION=EXTRACT. Whether the directory structure of the zip file will be maintained in the directory to which the zipfile contents are extracted. Defaults to YES.

An example of listing the contents of a zip file is:

```
<cfzip action="list" zipfile="sourcepath\somefile.zip"
variable="somevar">

<cfdump var="#x#">
```

An example of extracting a zip file is:

```
<cfzip action="extract" zipfile="soourcepath\somefile.zip"
destination="destpath">
```

The CFZIPPARAM tag can be used to embed references to multiple files/directories during the process of creating a zip file. It is to be used only with CFZIP ACTION="create". The following table lists the CFZIPPARAM tag attributes.

Attribute	Description
Source	Required. The path to a file, or directory in which to find files, to be added to the zipfile
Recurse	Optional. Indicates whether the subdirectories in the SOURCE directory be included. Defaults to YES.
Filter	Optional. Used with ACTION=CREATE. Specify a filter for the given SOURCE directory e.g. *.gif (similar to the cfdirectory filter attribute)
NewPath	Optional. If the item specified in SOURCE is a file then this attribute can be used to specify a replacement path. If the SOURCE item is a directory then this is ignored.
Prefix	Optional. A prefix that will be prepended to the path of files in the created zipfile

CFML Functions

4.4 Enhanced CFML Functions

This section lists CFML function enhancements that are unique to BlueDragon.

4.4.1 CreateObject

See the discussion under CFOBJECT in section 4.2.15, for information on a new `type` value of `.net`, which is supported only on *BlueDragon for the Microsoft .NET Framework.*

4.4.2 ListToArray

BlueDragon adds a new third argument to `ListToArray()`, a boolean value, which determines whether to include empty list elements in the resulting array. The default is `no`, which causes it to operate consistently with ColdFusion.

Consider the following:

```
<cfset list = "1,2,,3">
<cfdump var="#listToArray(list,",")#">
```

Both ColdFusion and BlueDragon would return an array of 3 elements, even though there are 4 items in the list, the third of which is empty. Use the newly available third argument to change this behavior::

```
<cfset list = "1,2,,3">
<cfdump var="#listToArray(list,",","yes")#">
```

This creates instead an array of 4 elements, with the third being empty.

4.4.3 ParagraphFormat

From the CFML Reference for CF5:

> "Returns *string* with converted single newline characters (`CR/LF` sequences) into spaces and double newline characters into HTML paragraph markers (`<p>`)."

BlueDragon varies from this behavior in that it converts single newline characters into HTML break tags (`<br>`) instead of spaces. Double newline characters are converted into HTML paragraph markers (`<p>`) by both BlueDragon and CF5.

4.4.4 StructNew

BlueDragon has enhanced the `StructNew()` function to accept an optional argument, a boolean, indicating whether to create a structure allowing case-sensitive keynames. The default is false, in which case BlueDragon internally normalizes structure keys to lowercase.

Generally, the case of structure keynames is not important and shouldn't be relied upon. Indeed, if you have code that relies upon the current default behavior of normalizing

lower-case, you should use caution when implementing this enhancement as it may break that code.

There are situations, however, where the case of keynames is important. For instance, see the discussion of the enhancement to `XMLTransform()`, in section 4.4.7.

Finally, note that there are some cases where structures created by internal processes are already case-sensitive. For instance, in XML tag and function processing, the keynames created from XML elements are case sensitive.

4.4.5 XMLSearch

`XmlSearch()` uses XPath expressions to extract data from an XML document. In CFMX, the result is an array of XML document objects containing the elements that meet the expression criteria. BlueDragon additionally supports execution of any other type of XPath statement that may return a boolean, string, or number type as a result.

4.4.6 XMLParse

BlueDragon offers additional functionality with respect to case sensitivity, node processing, and array handling. See section 5.4 for more information.

4.4.7 XMLTransform

BlueDragon adds the ability to pass arguments to an XML transformation by way of a new optional third argument to `XMLTransform()`. The value of the argument is a structure, whose keys are used to substitute values in any XSLT `param` elements that may be found in the XSLT specified in the second argument. An example of these `param` elements is `<xsl:param name="keyname"></xsl:param>`. For more information on using these substitutable parameters, consult an XSLT reference.

Be aware that by default, BlueDragon normalizes structure keynames to lower case, which could compromise the ability to match XSLT `param` elements. To address this issue, an enhancement has been made to the `StructNew` function. See the documentation in section 4.4.4.

Additionally, the key values in the structure that's passed to the transformation can be any valid java datatype or object. Normally they'll be strings, but if you want to use XALAN extensions and need to pass a real object, we permit that and do not convert it to a string automatically (the XALAN engine, which is the underlying XML/XSLT engine, does this where appropriate).

4.5 New CFML Functions

This section lists new CFML functions that are unique to BlueDragon.

4.5.1 Assert

BlueDragon has added an `Assert()` function to CFML. See the discussion of `CFASSERT` for more information. The `Assert()` function takes as its only argument the expression to be tested, as in:

```
<CFSCRIPT>
      Assert(somevar is somevalue);
</CFSCRIPT>
```

4.5.2 GetHttpContext

For BlueDragon.NET only, returns the ASP.NET `System.Web.HttpContext` object associated with the currently executing request. For instance, to view the machine name for the current server, run the following code:

```
<cfset svr = gethttpcontext().get_server()>
<cfoutput>#svr.get_machinename()#</cfoutput>
```

For more information on the `HttpContext` object, including its many properties including `Server`, `Request`, `Response`, and more, see:

http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cpref/html/frlrfsystemwebhttpcontextclassservertopic.asp

For more information on calling .NET objects in general (including the use of the get_ syntax used in the example), see the manual, *Integrating CFML with ASP.NET and the Microsoft .NET Framework.*

4.5.3 ListRemoveDuplicates

BlueDragon has added a `ListRemoveDuplicates()` function to CFML. It removes any duplicate elements in a given list. There is no return value. The syntax is as follows:

```
ListRemoveDuplicates( query [, separator])
```

The function accepts an optional second argument describing the list separator, which defaults to a comma(,).

An example of usage is:

```
<cfset list = "1,2,3,3">
<cfdump var="#listRemoveDuplicates(list)#">
```

4.5.4 QueryDeleteRow

BlueDragon has added a `QueryDeleteRow()` function to CFML. It deletes a given row from a query resultset. There is no return value. The syntax is as follows:

```
QueryDeleteRow( query, rowNumberToDelete )
```

The second argument refers to the row number within the query result set, not any internal database record id. An example of usage is:

```
<CFSET Query = QueryNew("id,name,age")>
<CFLOOP INDEX="X" FROM=1 TO=8>
   <CFSET QueryAddRow(Query,1)>
   <CFSET QuerySetCell(Query,"ID",X,X)>
   <CFSET QuerySetCell(Query,"Name","Name #X#",X)>
   <CFSET QuerySetCell(Query,"Age",X+15,X)>
```

```
</CFLOOP>

Before deleting:
<CFDUMP VAR="#Query#">

<cfset QueryDeleteRow( Query, 8)>
After deleting:
<CFDUMP VAR="#Query#">
```

4.5.5 QuerySort

BlueDragon has added a QuerySort() function to CFML. It sorts a given query resultset according to provided sort arguments. There is no return value. The syntax is as follows:

```
QuerySort( query, column, sorttype, direction )
```

The following table lists the `QuerySort` arguments.

Argument	Description
Query	Required. The CFML query resultset to be sorted (could be from a CFQUERY or tags like CFDIRECTORY, CFPOP, CFZIP, etc.)
Column	Required. The query column to be used for sorting the query result set.
SortType	Required. The type of sort to perform. That values can be TEXT, NUMERIC, or TEXTNOCASE.
Direction	Optional. Indicates the sort order, which can be ASC or DESC. Defaults to ASC.

An example of usage is the following:

```
<CFDIRECTORY ACTION="LIST" DIRECTORY="c:\" NAME="tests">
<CFSET QuerySort(tests,"name","TEXT","DESC")>
<CFDUMP VAR="#tests#">
```

4.5.6 Render

BlueDragon has added a Render function, which will dynamically render (execute) the CFML within any variable. This solves a long-standing problem where developers have wished to store CFML in a database column, for example, and then later process it.

While it may seem like a CFINCLUDE, it's much more powerful and by designing it as a function it's more flexible, in that the results can more easily be processed (or ignored).

Examples include the following:

```
<cfoutput>#Render( someQuery.cfmlContent )#</cfoutput>

<cfscript>
writeOutput( Render( someQuery.cfmlContent ) );
</cfscript>

<cfset render(somecfmlcontent)>
```

As with a CFINCLUDE, any CFML in the variable is processed just as if it was running in the template that called it. Any variables set inside this CFML will be available to the calling template, and path names for custom-tags and CFINCLUDE's will be relative to the calling template.

5 Miscellaneous Enhancements

There are various other aspects of working with ColdFusion and CFML that may be slightly different in BlueDragon, but don't fit neatly into a discussion of tags or functions.

5.1 Option to Support Relative Paths in Tags Requiring Absolute

There are a number of CFML tags that manipulate the file system via the `file` attribute. In ColdFusion, you must specify a full file system path for the `file` attribute for these tags:

```
CFCACHE
CFCONTENT
CFDIRECTORY
CFEXECUTE
CFFILE
CFFTP
CFHTTP
CFIMAGE
CFLOG
CFPOP
CFSCHEDULE
```

BlueDragon adds an optional `URIDirectory` attribute to these tags to indicate whether the `file` attribute specifies a full file system path or a URI path that is relative to the web server's document root directory. For example, the following tags would produce the same result on Microsoft IIS:

```
<CFFILE ACTION="delete" FILE="C:\Inetpub\wwwroot\images\a.jpg">
<CFFILE ACTION="delete" FILE="/images/a.jpg" URIDIRECTORY="Yes">
```

Specifying `file` attributes as relative URI paths improves the portability of CFML pages by eliminating web server and operating system specific physical path specifications. Note that if the code above was moved to a Linux running Apache, the first tag is not portable, but the second one is.

The optional `URIDirectory` attribute accepts the values "Yes" and "No"; the default value is "No".

5.2 Integrating JSP/Servlets Alongside CFML Templates

BlueDragon Server JX and BlueDragon/J2EE both allow you to execute JSPs and servlets alongside your CFML templates, as well as integrate your CFML with those and other Java components. ColdFusion MX requires the Enterprise edition for the same capability. For more information on CFML/J2EE integration, see the *BlueDragon 6.2.1 User Guide*.

5.3 Integrating ASP.NET Alongside CFML Templates

With BlueDragon for the Microsoft .NET Framework, you can run ASP.NET pages alongside your CFML templates (because the .NET framework knows how to process

them) , as well as integrate your CFML with those and other Java components. In the .NET edition, BlueDragon adds even more integration features than where ever possible in the Java editions of BlueDragon and ColdFusion. For more information on ASP.NET integration, see *Integrating CFML with ASP.NET and the Microsoft .NET Framework.*

5.4 XML Handling

There are a few ways in which BlueDragon supports XML in enhanced ways over ColdFusion. Rather than point these out with respect to particular tags or functions, this section introduces these enhancements.

5.4.1 Case Sensitivity

XML case sensitivity is an optional parameter that can be passed to `<cfxml>` and `XMLparse()`. The created XML object then requires case sensitive treatment when accessing nodes.

CFMX won't allow you to access an XML object using dot notation when you create it using the case sensitive option, even if you use proper case. The error returned indicates that CFMX is uppercasing the dot notated name, complaining that it cannot find the uppercased value in the XML object. It won't find it when comparing on a case sensitive basis. This operation is contrary to the ColdFusion documentation.

More specifically, in CFMX, using case sensitive XML objects forces you to use `myDoc["Root"]["FirstNode"]` notation. CFMX uppercases all their nodes so you cannot use normal dot notation when case sensitivity is turned on. In BlueDragon, we support both bracket and dot notation with case sensitive and case insensitive XML objects.

5.4.2 Assignment of New Nodes

CFMX does not allow adding nodes via assignment unless both the LHS (left hand side) node name and RHS node name are identical. BlueDragon does. In the event of a mismatch, BlueDragon lets the RHS node name be the name of the appended node.

For example, the following works in BlueDragon but fails in CFMX because the node names don't match up.

```
myDoc.Root.SubNode = XmlElemNew(myDoc, "WrongNode")
```

BlueDragon allows the RHS node name to take precedence.

In addition, the following fails in CFMX when there is only 1 `SubNode` element child of `Root`.

```
myDoc.Root.SubNode[2] = XmlElemNew(myDoc, "SubNode")
```

This is allowed in BlueDragon.

5.4.3 XML Array Processing

There are some instances in CFMX where an XML node cannot be treated as an array in array processing functions. For example, the following works in CFMX:

```
ArrayClear(myDoc.Root.SubNode)
```

But the following does not:

```
ArrayInsertAt(myDoc.Root.SubNode,1,XmlElemNew(myDoc,"SubNode"))
```

In BlueDragon, a node with even one element can be processed by the array functions.

5.5 Application.cfm Processing Enhancements

5.5.1 Application.cfm Processed Even When Requested Template Does Not Exist

BlueDragon offers an enhancement whereby if a URL requests a file that does not exist, Application.cfm is still processed before rejecting the request as a file not found, so that processing can take place that redirects based on the requested URL. This has an important advantage over CFMX, especially in the way of creating search-engine safe URLs or otherwise hiding the technology behind your site.

As an example, the sites Blog-City.com and LinuxWorld.com both use URLs such as the following to request CFML-driven pages. Notice that it's a request for an HTM file:

```
http://alan.blog-city.com/read/789800.htm
```

In this case, the file 789800.htm doesn't physically exist. The only file in the `read` directory is `Application.cfm`. When that request is processed, BlueDragon runs the `Application.cfm which in their code then` parses the `cgi.script-name` looking for the filename ('`789800`') and then makes a decision on that (such as pulling a record from a database or such), and then they render the template. At the end of Application.cfm processing, they call `<CFABORT>` to prevent the user getting a "file not found" error.

Using this technique, they can create very clean URLs without having to resort to complicated REWRITE rules in their web server. Also, using this technique allows them to code to a Model-View-Controller (MVC) paradigm more effectively, with the `Application.cfm` file being the controller.

5.5.2 Search Process for Application.cfm Stops at Docroot

In both ColdFusion and BlueDragon, if an `Application.cfm` file is not located in the same directory as a page being requested, each ancestor directory (parent, grandparent, etc.) will be searched until an `Application.cfm` is found. In BlueDragon, the search will stop at the web server document root directory or J2EE web application root directory, whereas ColdFusion will search beyond that to the drive root.

5.6 WhiteSpace Compression

BlueDragon's whitespace suppression is more thorough than ColdFusion's, which can reduce the bandwidth required to send pages to clients. See the discussion in section 4.2.17 as well as in the *BlueDragon 6.2.1 CFML Compatibility Guide*.

5.7 Error handling enhancements

BlueDragon offers various enhancements with regard to error handling and logging, as discussed in the section above on `CFERROR` as well as the section "Resolving CFML Compatibility Errors" in the *BlueDragon 6.2.1 CFML Compatibility Guide.*