
# BlueDragon™ 6.1

## CFML Compatibility and Reference Guide

NEW ATLANTA COMMUNICATIONS, LLC

# BlueDragon™ 6.1 CFML Compatibility and Reference Guide

June 17, 2004
Version 6.1



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

# Contents

# BlueDragon 6.1
# CFML Compatibility and Reference Guide

## 1 Introduction

New Atlanta BlueDragon is family of server-based products for the deployment of ColdFusion® Markup Language (CFML) applications for dynamic web publishing—featuring 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 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 6.1 CFML Compatibility Guide* presents information about the compatibility of the implementation of CFML in BlueDragon compared to what developers may expect when using Macromedia ColdFusion MX 6.1. Developers currently working with ColdFusion 4.5 or 5.0 should be aware that there are differences between those releases and ColdFusion MX 6.1 which are not generally documented in this manual. See the Macromedia documentation for information on those differences.

### *1.2 BlueDragon Product Configurations*

BlueDragon is currently available in three product configurations. Details about these configurations—BlueDragon Server, BlueDragon Server JX, and BlueDragon for J2EE Servers—are provided in the *BlueDragon Installation Guide*. Except where explicitly noted, all references to “BlueDragon” in this document refer to all three 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

### 1.4 Other Documentation

The other manuals available in the BlueDragon documentation library are:

- *BlueDragon 6.1 Installation Guide*
- *BlueDragon 6.1 User Guide*
- *Deploying CFML on J2EE Application Servers*

Each offers useful information that may be relevant to developers, installers, and administrators, and they are available in PDF format from New Atlanta's web site:

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

## 2 ColdFusion Compatibility

### 2.1 ColdFusion Compatibility

The BlueDragon implementation of the CFML scripting language is designed to be highly compatible with the Macromedia ColdFusion MX 6.1 (CFMX) implementation. This document describes differences in syntax and semantics between the two implementations, including BlueDragon enhancements that are not supported by ColdFusion Server 5.0 (CF5) or CFMX. Except where explicitly noted, all occurrences of "ColdFusion" in this document refer to both CF5 and CFMX.

With the 6.1 release, BlueDragon now implements features introduced by Macromedia ColdFusion MX (CFMX), such as ColdFusion Components (CFCs), XML processing, and Web Services.

This document is not a complete reference to the CFML scripting language; for in-depth coverage of CFML, the following books are recommended:

**ColdFusion MX Bible**
by Adam Churvis, Hal Helms, and Charlie Arehart
http://www.wiley.com/WileyCDA/WileyTitle/productCd-0764546228.html

**Programming ColdFusion MX**
by Rob Brooks-Bilson
http://www.oreilly.com/catalog/coldfusion2/

## 2.2 Migration from ColdFusion to BlueDragon

While this document covers issues of CFML language compatibility, additional information about migration from ColdFusion to BlueDragon is provided in the *BlueDragon User Guide*, in the Section "Migration from ColdFusion". Developers are strongly encouraged to review that material, and indeed all the information presented in the *User Guide*.

## 2.3 Enhancements to CFML in BlueDragon

A significant portion of this guide is devoted to detailing information about enhancements or additions to CFML that are available in BlueDragon. These include (but are not limited to):

- new tags and functions (such as `CFIMAP`, `CFIMAGE`, `CFASSERT`, and `CFDEBUGGER`, to name a few)
- enhancements to existing tags (such as a new `CACHENAME` attribute for `CFQUERY`, and the option to leave off `VAR` on `CFDUMP`)
- changes in behavior of language-oriented features (such as page buffering behavior)

These additions or enhancements are provided in direct response to customer feedback, to provide important and needed benefits.

# 3 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.

BlueDragon follows CFMX as a model in its handling of variables containing periods in their names. Both act differently than CF5 in this respect.

## 3.2 CFML Keywords/Reserved Words

ColdFusion resources caution developers to avoid use of keywords as variable or user-defined function names. Even so, there are instances where ColdFusion will permit use of some keywords in variable names when BlueDragon will not.

For instance, both ColdFusion and BlueDragon will fail if you attempt to create a variable named "mod", which is a logical operator used to perform a modulus operation (often used in a `CFIF` statement). However, ColdFusion will let you use the variable if you define it in a scope other than the local (variables) scope, such as `URL.mod`. The following code works in ColdFusion but fails in BlueDragon:

```
<CFSET url.mod=0>
<CFOUTPUT>#url.mod#</CFOUTPUT>
```

The error occurs on the `CFOUTPUT`, rather than the `CFSET`. Similarly, ColdFusion will allow you to create and use a variable named `var` (which is a keyword used within `CFSCRIPT` and `CFSET`), but BlueDragon will not. It also fails when you try to `CFOUTPUT`.

So it's not that you can never use keywords, just that you can't always use them the same way as in ColdFusion (and there's no reliable documentation of how reserve words can or cannot be used). Again, it's best to avoid keywords as variable or function names in CFML on both BlueDragon and ColfFusion.

## *3.3 Setting Variable Names Dynamically*

In BlueDragon, it is not possible to create variable names dynamically by using a quoted string, such as on the left side of the equals sign in a `CFSET`. Consider the following:

```
<CFSET hold="FirstName">
<CFSET "#hold#"="bob">
<CFOUTPUT>#FirstName#</CFOUTPUT>
```

In ColdFusion, this creates a new variable called `FirstName` and the output of the last line is "bob". On BlueDragon, this instead assigns "bob" to the variable `hold`, and the reference to the variable `FirstName` generates a "variable-not-found" runtime error.

You can work-around this issue any of three ways (and they all apply to any variable scope, not just the local or `variables` scope). The one closest to the code above involves simply adding the "variables" prefix within the second line before the variable name, as in:

```
<CFSET hold="FirstName">
<CFSET "variables.#hold#"="bob">
<CFOUTPUT>#FirstName#</CFOUTPUT>
```

Another similar variation would be to replace the second line with:

```
<CFSET variables[hold]=”bob")>
```

Still another approach is to use the `SetVariable()` function. You could replace the second line above with:

```
<CFSET temp = SetVariable(hold,"bob")>
```

or more simply:

```
<CFSET SetVariable(hold,"bob")>
```

Alternately, you could put the entire code fragment in a `CFSCRIPT`, as in:

```
<CFSCRIPT>
hold="firstname";
setvariable(hold,"bob");
writeoutput(firstname);
</CFSCRIPT>
```

## 3.4 SERVER Variables

For BlueDragon, the variable `Server.ColdFusion.ProductName` returns the value "BlueDragon" and `Server.ColdFusion.ProductLevel` returns a value indicating the BlueDragon product installed (such as "Server JX"). For BlueDragon 6.1, the `ProductVersion` variable returns the value "6,1,0,xxx" where "xxx" is the internal build number; for example: 6,1,0,237.

BlueDragon also offers its own scope, `Server.BlueDragon`, with 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 CF5 and CFMX, these pre-defined `Server` scope variables are read-only.

## 3.5 Client and Cookie Variables After Flush

After using `CFFLUSH`, if you try to set a cookie (or a client variable stored in a cookie), ColdFusion gives no error but does not send the cookie to the client (or set the value of the client variable). BlueDragon, on the other hand, will throw an exception (since, as with CFMX, you cannot use tags that modify the HTTP header after performing a `CFFLUSH`). For more on page flush (buffering) behavior in BlueDragon, see Section 4.4.4.

## 3.6 CFTOKEN Value

Both ColdFusion and BlueDragon support cookie variables named `CFID` and `CFTOKEN`, which are used as identifiers for maintaining client and session variable state (unless "J2EE Sessions" are enabled) for each user.

Where ColdFusion uses a simple 8-digit string of numbers for `CFTOKEN`, BlueDragon uses a much longer 35-character string of random numbers. This is similar to CFMX's option of enabling `CFTOKEN` as a `UUID`. The longer string improves security by making it much harder for a client to spoof valid `CFID`/`CFTOKEN` values and access another user's session/client variables.

Note that if the `CFID` and `CFTOKEN` cookies are already set for a given client/browser from their having visited a ColdFusion site, the `CFTOKEN` values will not reflect this behavior.

New CFID/CFTOKEN values are assigned only when a client/browser executes a CFML page for the first time without those cookies already set.

## 3.7 Client Variable Processing

There are some differences between BlueDragon and ColdFusion in the underlying implementation of client variable processing with regard to using a database to store the client variables.

First, where ColdFusion requires an administrator to configure a datasource to serve as a client variable repository, BlueDragon does not. If a datasource is specified in the CFAPPLICATION ClientStorage attribute (or indeed in the administration console when setting a default client storage datasource, if desired), BlueDragon will store client variables there without need to prepare the datasource to hold client variables.

Note as well that BlueDragon will not create the table needed within the datasource (BDCLIENTDATA) until a page is processed that references client variables.

BlueDragon does not currently support an option to expire client variables due to inactivity. This is planned for a future update.

Additionally, in ColdFusion, when client variables are stored in a datasource they are stored in a table named CDATA that uses two columns as the primary key: CFID (which is a combination of CFID:CFTOKEN) and APP (which is the application name as specified in CFAPPLICATION). Therefore, client variables are stored per-application so that they are only retrieved and visible within the named application.

BlueDragon currently stores all client data in a single table named BDCLIENTDATA, which has a primary key of only CFID (the CFID:CFTOKEN combination). This means that client variables are not stored per-application (client variables created for a user in one application are visible to that user in another application—but each client's variables are unique to them as individual visitors).

# 4 CFML Tags

## *4.1 ColdFusion Components*

There are a few differences in CFC (ColdFusion Component) processing in BlueDragon, including both enhancements and limitations, and with respect to their use as web services. The following sections discuss these.

### 4.1.1 CFC Enhancements in BlueDragon

BlueDragon adds the following enhancements when working with CFCs:

- CFC instances can be duplicated using the `Duplicate()` function.
- CFC instances can be serialized (useful for J2EE servers 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 adds `CFCONSTRUCTOR` for explicit definition of constructors (in addition to implicit construction). See Section 4.5.3 for more information.

### 4.1.2 CFC Limitations in BlueDragon

BlueDragon has the following limitations when working with CFCs:

- A component file (.cfc) must contain opening and closing `CFCOMPONENT` tags.
- It's not possible to define component functions inside `CFSCRIPT` blocks. Component functions can only be defined using the `CFFUNCTION` tag.
- The file `component.cfc` must exist, must not be empty, and must contain a `CFCOMPONENT` tag pair (even if the tag pair is empty). In BlueDragon Server and Server JX, it's stored by default in their `[bluedragon]\config` directory. In the J2EE edition, it's stored by default in the `WEB-INF\bluedragon` directory.
- If you modify the source of a CFC that is inherited by another CFC, but don't modify the inheriting CFC, the modified inherited class will not be reloaded when the inheriting class is accessed. To reload the inherited class you must access it directly, or manually flush the File Cache using the BlueDragon admin console

For additional limitations related to CFCs, see Section 4.3.10, discussing the `CFFUNCTION` tag, and Section 6.5, discussing declaration of CFC methods within a `CFINCLUDE` tag.

#### *4.1.2.1 Case-sensitive Searching for CFCs on Unix-based Systems*

On UNIX-based systems (including Linux and Mac OS X), when invoking a CFC, the search for the CFC will be case-sensitive in the following cases:

- when searching the current template directory for CFCs whose names are specified without using dotted notation;
- when using mapped directories;
- when specifying a CFC name using dotted notation in the `EXTENDS` attribute of `CFCOMPONENT`;
- when specifying a CFC name using dotted notation in the `TYPE` attribute of `CFARGUMENT`; and,
- when specifying a CFC name using dotted notation in the `RETURNTYPE` attribute of `CFFUNCTION`.

### 4.1.3 Web Services

Web services method names in BlueDragon are case-sensitive.

For web services interoperability with CFMX, web services that use CFCs as parameters/return types must define identical CFCs in both BlueDragon and CFMX. CFC names and properties are case sensitive.

Query result datatypes used in web services are only preserved as such when using BlueDragon as both web services client and server.

Complex object web services parameters will only be recieved as CFC instances if a CFC is defined that matches the complex object structure and is found in or with a directory or name that corresponds to the complex object namespace or name. Received complex objects that do not have a corresponding CFC (e.g., that are unknown to BlueDragon) will be deserialized into an XML object.

## *4.2 Unsupported Tags*

The following CF5 tags are not supported by BlueDragon, and will generate run-time errors when processed by BlueDragon:

| Unsupported CF5 Tags | | |
|---|---|---|
| Client-side Java | Extensibility | Web Applications |
| `CFAPPLET` | `CFREPORT` | `CFAUTHENTICATE`* |
| `CFGRID` | | `CFIMPERSONATE`* |
| `CFGRIDCOLUMN` | | |
| `CFGRIDROW` | | |
| `CFGRIDUPDATE` | | |

*obsolete in Macromedia CFMX

The following tags added in CFMX are not supported by BlueDragon, and will generate run-time errors when processed by BlueDragon:

| Unsupported CFMX Tags | | |
|---|---|---|
| CFCHART | CFLOGIN | CFTRACE |
| CFCHARTDATA | CFLOGINUSER | |
| CFCHARTSERIES | CFLOGOUT | |

## *4.3 Supported with Limitations*

The following CFML tags are supported by BlueDragon with differences and limitations relative to the CFMX implementation as noted; tags are listed alphabetically.

### 4.3.1 CFAPPLICATION

BlueDragon does not support the `LoginStorage` attribute, introduced in CFMX 6.1.

### 4.3.2 CFARGUMENT

When a `TYPE` of `boolean` value is specified in `CFARGUMENT`, any value passed into the `CFFUNCTION` is converted internally to a CFML boolean variable type, if the value passed in is `YES/NO`, `1/0`, or `true/false`. This can cause unexpected results if you test for the original value passed in, such as `1` or `0`. Therefore, you should always treat `boolean` arguments as such when doing conditional processing (such as `CFIF` or `CFCASE`) within the function, and not attempt to treat them as numbers or strings.

### 4.3.3 CFCACHE

BlueDragon does not support the attributes `TimeSpan` and `CachedDirectory`, which were added in CFMX. They are simply ignored.

### 4.3.4 CFCOLLECTION

BlueDragon does not support the `Language` attribute. The default is always `English`. Additionally, BlueDragon does not support the `MAP`, `OPTIMIZE`, or `REPAIR` values for the `ACTION` attribute, as these are not required due to differences in the underlying search engine technology.

See additional compatibility information under `CFINDEX` and `CFSEARCH`.

### 4.3.5 CFCOMPONENT

The `OUTPUT` attribute of `CFCOMPONENT` is currently not supported; it is not possible to produce HTML output within a component pseudo-constructor

See Section 4.1 for additional discussion of ColdFusion components.

### 4.3.6 CFCONTENT

BlueDragon does not support using a pair of CFCONTENT tags, as in: <CFCONTENT…>*some data*</CFCONTENT>. In BlueDragon, only the opening CFCONTENT tag is supported.

### 4.3.7 CFDUMP

BlueDragon currently ignores the EXPAND and LABEL attributes available in ColdFusion.

### 4.3.8 CFERROR

In ColdFusion, when a CFERROR uses TYPE="Request", the page can list variables in the Error scope without need of a CFOUTPUT. BlueDragon does not recognize or output those variables; instead, you must wrap them in CFOUTPUT tags. The use of CFOUTPUT tag is compatible with ColdFusion, as the CFOUTPUT tags will simply be sent to the browser by ColdFusion, where they are ignored.

### 4.3.9 CFEXECUTE

BlueDragon does not support the Variable attribute, which was added in CFMX 6.1.

### 4.3.10 CFFUNCTION

BlueDragon does not support the Roles attribute.

BlueDragon does not support the OUTPUT="yes" in the same manner as ColdFusion. With this feature, meaningful only in the context of a CFC method, ColdFusion acts as if a CFOUTPUT is implied within the method, and free-standing expressions are evaluated and displayed even though they're not surrounded by CFOUTPUT tags. BlueDragon does not offer this behavior and therefore if OUTPUT="yes", which is the default, you must add CFOUTPUT tags in the method around free-standing expressions to cause them to be evaluated and displayed.

### 4.3.11 CFGRAPH

BlueDragon only supports creating graphs in JPEG format; it does not support GIF or Flash formats. Therefore, the only valid value for the fileFormat attribute is "JPG"; if any other value is assigned to this attribute an error will be generated by BlueDragon.

### 4.3.12 CFHTTP

BlueDragon also does not support the specification of https as the protocol in CFHTTP's url attribute.

Also, BlueDragon does not support the following new attributes added in CFMX: firstRowAsHeaders and charset, nor those added in CFMX 6.1: multipart, getasbinary, proxyuser, and proxypassword attributes, nor does it support the new values for method: head, put, delete, options, and trace. It also does not support the new variables in the CFHTTP structure (which is provided after a CFHTTP operation): charset, errorDetail, and text.

### 4.3.13 CFHTTPPARAM

BlueDragon does not support the following new `type` attribute values introduced in CFMX 6.1: `header` and `body`.

### 4.3.14 CFIMPORT

While BlueDragon supports use of `CFIMPORT` as an alternative approach to executing CFML custom tags (as introduced in CFMX), it does not support the ability to import or execute JSP custom tag libraries.

### 4.3.15 CFINDEX

In BlueDragon, the search capability (`CFSEARCH`, `CFINDEX`, and `CFCOLLECTION` tags) is based on the Jakarta Lucene project open source search engine, which results in differences between the BlueDragon and ColdFusion implementations.

For `CFINDEX`, BlueDragon does not support the `Language` attribute; the default language is always `English`. Also, BlueDragon is currently limited to searching HTML- and text-based documents.

See additional compatibility information under `CFSEARCH` and `CFCOLLECTION`.

### 4.3.16 CFINSERT

Like CFMX, BlueDragon does not support “DSN-less connections”; therefore, it always requires the `dataSource` attribute and does not support the following optional attributes:

```
connectString
dbName
dbServer
dbType
provider
providerDSN
```

See the discussion of `CFQUERY` attribute support in Section 4.3.22 for more information.

### 4.3.17 CFLDAP

BlueDragon does not support the `secure` attribute.

### 4.3.18 CFMAIL

BlueDragon does not support the `spoolEnable` attribute, introduced in CFMX, nor the new attributes introduced in CFMX 6.1: `failto`, `replyto`, `username`, `password`, and `wraptext`.

### 4.3.19 CFMAILPARAM

BlueDragon does not support the new `type` attribute introduced in CFMX 6.1.

### 4.3.20 CF_ Custom Tags

When invoking CFML custom tags using "CF_" notation on UNIX-based systems (including Linux and Mac OS X), the filename match is case-sensitive when searching the

current template directory. The filename match is not case-sensitive when searching custom tag directories.

### 4.3.21 CFOBJECT

BlueDragon supports "`java`", "`component`", and "`webservice`" values for the `type` attribute; it does not support the values "`com`" or "`corba`".

### 4.3.22 CFQUERY

#### *4.3.22.1 Unsupported Tag Attributes*

CF5 added a feature often referred to as "DSN-less" connections (where the database connection string is specified via the `connectString` attribute, rather than using a `dataSource` attribute). Like CFMX, BlueDragon does not support this feature or the following attributes.

```
connectString
dbName
dbServer
provider
providerDSN
```

These attributes are all deprecated in CFMX.

Additionally, BlueDragon does not support the following optional `CFQUERY` attributes:

```
blockfactor
timeout
```

#### *4.3.22.2 Query of Query processing*

BlueDragon supports query of query processing (specified by `dbType="query"`). While CFMX added support for new SQL language statements such as `lower`, `upper`, BlueDragon does not yet support all those statements.

### 4.3.23 CFREGISTRY

BlueDragon simulates the Windows registry on all operating systems, including Windows. Therefore, it is not possible to read, write, or delete "registry" entries other than those created using the `CFREGISTRY` tag.

In other words, on Windows systems, you cannot access the real Registry. There are available Java utilities that you can try in order to gain access to the real registry, via `CFOBJECT` calls to the utility's methods. At the time of this writing, URLs for possible candidates you may consider include:

http://www.trustice.com/java/jnireg/

http://www.bayequities.com/tech/Products/jreg_key.shtml

### 4.3.24 CFSCRIPT

BlueDragon does not support `try/catch` statements within `CFSCRIPT`.

### 4.3.25 CFSEARCH

In BlueDragon, the search capability (`CFSEARCH`, `CFINDEX`, and `CFCOLLECTION` tags) is based on the Jakarta Lucene project open source search engine, which results in differences between the BlueDragon and ColdFusion implementations.

Text searches work similarly; while BlueDragon does not identically support all the ColdFusion search language keywords such as `NEAR`, `STEM`, `WILDCARD`, `CONTAINS`, and others, which you might specify with a `CFSEARCH Type="explicit"`, BlueDragon does contains its own rich set of search language keywords. Many of them are the same or very similar to ColdFusion's keywords and operators.

Syntax and simple examples are offered in the brief but quite complete "Query Syntax" document available on the Lucene web site:

http://jakarta.apache.org/lucene/docs/queryparsersyntax.html

As for the `CFSEARCH` tag itself, BlueDragon does not support the `Language` attribute. The default is always `English`.

See additional compatibility information under `CFINDEX` and `CFCOLLECTION`.

### 4.3.26 CFSETTING

BlueDragon does not support the `RequestTimeOut` attribute which was introduced in CFMX, nor the `catchExceptionsByPattern` attribute, which was obsoleted in CFMX.

### 4.3.27 CFSTOREDPROC

Like CFMX, BlueDragon does not support "DSN-less connections"; therefore, it always requires the `dataSource` attribute and does not support the following optional attributes:

```
connectString
dbName
dbServer
dbType
provider
providerDSN
```

See the discussion of `CFQUERY` attribute support in Section 4.3.22 for more information.

#### *4.3.27.1 Oracle Stored Procedures and Reference Cursors*

There are some aspects of processing stored procedures on Oracle, especially with respect to reference cursors, that differ in BlueDragon compared to CF5 and CFMX (note that the method for calling stored procedure differs between CF5 and CFMX).

Oracle returns result sets from stored procedures as `OUT` parameters of type `REF CURSOR`. In BlueDragon 6.1, you simply use the `CFPROCPARAM` tag and specify the variable name to

hold the result set. The following code running on BlueDragon 6.1 returns the result set in a query variable named "myResults":

```
<cfstoredproc proc="myProc" datasource="dsn">
   <cfprocparam type="IN" value="#inValue#">
   <cfprocparam type="OUT"
                cfsqltype="CF_SQL_REFCURSOR"
                variable="myResults">
</cfstoredproc>
```

In CF5, you're required to specify a "dummy" OUT parameter and then use CFPROCRESULT to create a variable to hold the actual result set. The following code running on CF5 returns the result set in a query variable named "myResults":

```
<cfstoredproc proc="myProc" datasource="dsn">
   <cfprocparam type="IN" value="#inValue#">
   <cfprocparam type="OUT"
                cfsqltype="CF_SQL_REFCURSOR"
                variable="dummy">
   <cfprocresult name="myResults" resultset="1">
</cfstoredproc>
```

CFMX differs from CF5 in that you don't use the CFPROCPARAM tag to specify a dummy OUT variable, but only specify the CFPROCRESULT tag to access the results. The following code running on CFMX returns the result set in a query variable named "myResults":

```
<cfstoredproc proc="myProc" datasource="dsn">
   <cfprocparam type="IN" value="#inValue#">
   <cfprocresult name="myResults" resultset="1">
</cfstoredproc>
```

### 4.3.28 CFTEXTINPUT

ColdFusion allows the BGCOLOR and TEXTCOLOR attributes to be specified as hex values in the form “#nnnnnn”, although the documentation indicates the pound sign should be escaped, as in “##nnnnnn”. BlueDragon follows the documented approach and requires that the pound sign be escaped. As a work-around, if you used the undocumented single pound sign, changing it to use two will still be compatible with ColdFusion.

### 4.3.29 CFTREE

BlueDragon does not support the following optional CFTREE attributes:

```
completePath
delimiter
onValidate
```

### 4.3.30 CFTREEITEM

BlueDragon does not support the following optional CFTREEITEM attributes:

```
img
imgOpen
```

### 4.3.31 CFUPDATE

Like CFMX, BlueDragon does not support “DSN-less connections”; therefore, it always requires the `dataSource` attribute and does not support the following optional attributes:

```
connectString
dbName
dbServer
dbType
provider
providerDSN
```

See the discussion of `CFQUERY` attribute support in Section 4.3.22 for more information.

### 4.3.32 CFWDDX

The following limitations exist in the BlueDragon implementation of the `CFWDDX` tag relative to the ColdFusion implementation:

1. BlueDragon cannot deserialize binary data from WDDX to CFML. BlueDragon can serialize all other kinds of CFML data, including query resultsets, arrays, and structures to name a few.
2. The `useTimeZoneInfo` attribute is not supported by BlueDragon (defaults to “Yes”).
3. The `validate` attribute is not supported by BlueDragon.
4. BlueDragon uses a different notation than ColdFusion when JavaScript objects are created:

| | |
|---|---|
| ColdFusion notation: | MyStock[“price”] = “66.25”; |
| BlueDragon notation: | MyStock.price = “66.25”; |

## *4.4 Enhanced CFML Tags*

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

### 4.4.1 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 admin 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.

See additional information on the optional `WAIT` attribute, discussed under `CFINDEX` in Section 4.4.6.

### 4.4.2 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">
```

### 4.4.3 CFDUMP

The CFDUMP tag VAR attribute is required in ColdFusion, but 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.

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.

BlueDragon's dump also 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.

### 4.4.4 CFFLUSH

BlueDragon 6.1 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 now defaults to buffering the entire page, like ColdFusion. See the *BlueDragon 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, 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.

BlueDragon also supports the simple use of `CFFLUSH` to flush all of the page content generated to that point.

### 4.4.5 CFINCLUDE

BlueDragon allows you to include the output of Java servlets or JavaServer Pages (JSP) in your CFML pages via the new `page` attribute to the `CFINCLUDE` tag. The `page` attribute specifies the URI for the page to include. 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.jsp”>
```

`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.

BlueDragon also supports the concept of mappings defined in the administration console. See the *BlueDragon 6.1 User Guide* Section “Migration from ColdFusion” for more information on creating administration console mappings.

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.

### 4.4.6 CFINDEX

#### *4.4.6.1 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://.

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/html files formats (.cfm, .htm, .jsp, .asp, 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.

#### *4.4.6.2 Wait Processing*

Index creation (spidering a web site or indexing a file collection) 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.

### 4.4.7 CFLOCK

BlueDragon supports the full syntax and semantics of `CFLOCK`, but like CFMX does not always require the use of `CFLOCK` when accessing variables in the Session, Application, and Server scopes. 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.

### 4.4.8 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"
        ...>
```

### 4.4.9 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 one the request is processing.

### 4.4.10 CFPROCESSINGDIRECTIVE SuppressWhiteSpace Attribute

The CFPROCESSINGDIRECTIVE tag in CFML was introduced in CF5 with an available SuppressWhiteSpace attribute to control the creation of extra whitespace characters. It is designed to be used as a tag pair, affecting all code executed within the tag pair. BlueDragon supports this tag with the following extended capabilities.

There is also an available Administrator setting to turn on whitespace suppression by default for all pages, and this is available in BlueDragon as within ColdFusion.

BlueDragon's whitespace suppression is more thorough than ColdFusion's, which is generally desirable as it reduces the total size of the HTML response sent to the client. It may, however, eliminate whitespace where it's undesirable, such as within JavaScript or within HTML tags such as PRE and TEXTAREA. You can vary the whitespace suppression in such cases by surrounding the tags or text with a <CFPROCESSINGDIRECTIVE SUPPRESSWHITESPACE="No"> pair. You can also nest CFPROCESSINGDIRECTIVE tag pairs.

Also, while in CF5 and CFMX, CFPROCESSINGDIRECTIVE 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. Again, this is generally an enhancement and a desirable feature. It can cause problems with applications that are not expecting it.

This problem arises with Fusebox applications. Since the core files intentionally turn on whitespace suppression, in BlueDragon that behavior trickles down to all pages included from the core files (which in Fusebox is all files in the applications). This unexpected behavior can cause the challenges described above with respect to JavaScript and tags such as PRE and TEXTAREA.

Again, 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 behavior for that template.

Also, just as CFMX introduced the ability to specify the attribute value ("yes" or "no") as a variable, BlueDragon also supports that approach.

### 4.4.11 CFQUERY

#### *4.4.11.1 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 a further enhancement, if you would like all variables within a query to automatically preserve any single quotes, BlueDragon 6.1 adds 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>
```

### *4.4.11.2 New CacheName Attribute*

BlueDragon offers improved caching for `CFQUERY` tags via the new `cacheName` and `action` attributes. The optional `cacheName` attribute can be used to assign a unique name for cached `CFQUERY` results:

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

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.

The `cachedWithin` and `cachedAfter` attributes as implemented by ColdFusion can be used in conjunction with `CACHENAME`.

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 4.4.9 for more information.

### 4.4.12 CFSEARCH

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

### 4.4.13 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">
```

### 4.4.14 CFXML

BlueDragon offers additional functionality with respect to case sensitivity, node processing, and array handling. See Section 6.3 for more information.

### 4.4.15 URIDirectory Attribute (FILE Attribute Modifier)

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 in the above example that the first tag is not portable to a Linux running Apache, but the second one is.

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

## *4.5 New CFML Tags*

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

### 4.5.1 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 5.4.1) takes an expression that is expected to evaluate to a Boolean result (true or false). It throws an exception if the result is false but does nothing if the result is true. They are also ignored if assertions have not been enabled in the BlueDragon admin 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.

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 this and using a `CFIF` is that the `CFIF` is intended to control the flow of the logic, executing code depending on a condition. An assertion test is intended to simply throw an error is the expected condition is not true. The `CFIF` test handles expected conditions, while the assertion flags unexpected conditions.

It 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.)

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.

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.

### 4.5.2 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 Servers* for a detailed discussion of CFBASE.

### 4.5.3 CFCONSTRUCTOR

BlueDragon has added a feature to make creation of "constructor" logic in CFCs more explicit with CFCONSTRUCTOR. An example follows:

```
<cfcomponent>
      <cfconstructor>
            <cfset variables.foo = 1>
      </cfconstructor>
      <cffunction name=″somefunction″>
            <cfreturn "test">
      </cffunction>
</cfcomponent>
```

The semantics are similar to how a Java object works in that you use init() to explicitly call the constructor. If you don't call it explicitly, it is implicitly called the first time you use a method. As with Java objects, defining a constructor and a method named init is not supported. (A method named init is permissible in a CFC, to remain backward compatible with CFMX, unless you use CFCONSTRUCTOR.)

If a component uses CFCONSTRUCTOR, its code will be executed (constructed) either implicitly upon first access or explicitly by invoking an init method when invoking the CFC. CFCONSTRUCTOR creates a specially flagged init method. Components that simply have init methods without using CFCONSTRUCTOR do not get implicit construction.

Also, BlueDragon supports the implicit constructor notion offered by CFMX, where any code inside a component that's not within a CFFUNCTION is also executed the first time any method in a component is called.

For additional information on CFC method processing, see Section 6.5.

### 4.5.4 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="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) with 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.

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

http://www.sys-con.com/coldfusion/article.cfm?id=679

### 4.5.5 CFFORWARD

CFFORWARD is a tag introduced by BlueDragon that allows you to do a “server-side redirect” to another CFML page, a Java servlet, or a JavaServer Page (JSP). 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 URL, FORM, and REQUEST scopes are available to the target of the CFFORWARD tag.

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">
```

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

### 4.5.6 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>
```

### 4.5.7 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.

#### *4.5.7.1 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

#### *4.5.7.2 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.

#### *4.5.7.3 Listing Mailbox Folders*

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

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

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.

### *4.5.7.4 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.

### *4.5.7.5 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.

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

#### *4.5.7.6 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.

#### *4.5.7.7 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.

#### *4.5.7.8 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.

#### *4.5.7.9 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.

#### *4.5.7.10 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.

#### *4.5.7.11 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.

### 4.5.8 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">
```

# 5 CFML Functions

## 5.1 Unsupported Functions

The following CFML functions are not supported by BlueDragon:

| Unsupported CF5 Functions | | |
|---|---|---|
| AuthenticatedContext** | LSIsNumeric | getK2ServerDocCount* |
| AuthenticatedUser** | LSNumberFormat | getK2ServerDocCountLimit* |
| IsAuthenticated** | LSParseDateTime | isK2ServerABroker* |
| IsAuthorized** | LSParseEuroCurrency | isK2ServerDocCountExceeded* |
| IsProtected** | LSParseNumber | isK2ServerOnLine* |
| | | |
| GetException | IsNumericDate | |
| GetMetricData | | |

*deprecated in Macromedia CFMX 6.1
**obsoleted in Macromedia CFMX 6.1

| Unsupported CFMX Functions | | |
|---|---|---|
| GetAuthUser | GetProfileSections | IsUserInrole |
| GetEncoding | GetServiceSettings | ReleaseCOMObject† |

† new in CFMX 6.1

## 5.2 Supported with Limitations

### 5.2.1 CreateObject

BlueDragon supports "java", "component", and "webservice" as values for the first argument to CreateObject(); it does not support COM or CORBA.

### 5.2.2 DateAdd, DatePart, LSTimeFormat, and TimeFormat

BlueDragon does not support the new "L" (or "l") value for use in DateParts in these functions.

### 5.2.3 DateFormat/TimeFormat

BlueDragon does not support timeformat masks in DateFormat() as completely as ColdFusion does. For instance, in BlueDragon, the HH mask returns 12-hour rather than 24-hour time.

Additionally, in both functions, the `t` mask returns a value of `am` or `pm`, where it should return just an `a` or `p`.

### 5.2.4 Decrypt/Encrypt

BlueDragon does not support decryption of text that was encrypted by ColdFusion pages using their implementation of this function (such as data stored in a file or database after encryption). Conversely, text encrypted on BlueDragon cannot be decrypted in ColdFusion. As a work-around, simply re-run the process to encrypt the text.

### 5.2.5 InputBaseN

In BlueDragon, if the first argument to `InputBaseN()` has a `0x` prefacing the number, it will fail. An example is:

```
<CFOUTPUT>#InputBaseN("0xefcdab 8 9",16)#</CFOUTPUT>
```

A work-around is to simply remove the `0x` from the argument string.

### 5.2.6 StructKeyArray and StructKeyList

The functions `StructKeyList()` and `StructKeyArray()` return keys in all uppercase.

### 5.2.7 StructSort

BlueDragon does not support case sensitive sorting in `StructSort()`. If the `sortType` is not `numeric` then it will sort as `textnocase`. The reason is that BlueDragon stores the struct keys internally as all lowercase.

## *5.3 Enhanced CFML Functions*

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

### 5.3.1 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 3rd 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.

### 5.3.2 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.

### 5.3.3 XMLSearch

`XmlSearch()` uses XPath expressions to extract data from 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.

### 5.3.4 XMLParse

BlueDragon offers additional functionality with respect to case sensitivity, node processing, and array handling. See Section 6.3 for more information.

### 5.3.5 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.

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).

## *5.4 New CFML Functions*

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

### 5.4.1 Assert

BlueDragon has added an `Assert()` function to CFML. See the discussion of `CFASSERT` for more information.

# 6 Miscellaneous

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.

## *6.1 Integrating JSP/Servlets Alongside CFML Templates*

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

## *6.2 CFLOG File Placement*

When using `CFLOG` in BlueDragon Server and Server JX, logs are written to the `work\cflog` directory of the BlueDragon installation directory. For BlueDragon/J2EE, logs are written to the `WEB-INF\bluedragon\work\cflog\` directory of the J2EE web application.

## *6.3 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.

### 6.3.1 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.

### 6.3.2 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.

### 6.3.3 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.

## *6.4 Search Process for Application.cfm*

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.

## *6.5 CFC Method Declaration via CFINCLUDE*

ColdFusion MX permits use of `CFINCLUDE` inside a `CFCOMPONENT` to specify one or more `CFFUNCTION` declarations. BlueDragon does as well, but with the following limitations.

BlueDragon will not recurse through `CFINCLUDE`s during this processing to find additional `CFFUNCTION` declarations. In other words, while it will analyze a `CFINCLUDE` found within the `CFCOMPONENT` to locate any `CFFUNCTION` declarations, it will not analyze `CFINCLUDE`s found within that included file. (The `CFINCLUDE` will be processed as expected at run time, but the process of determining available methods for the CFC is the matter that’s handled differently).

Similarly, when performing its analysis to find `CFFUNCTION` declarations, BlueDragon will not be able to process a `CFINCLUDE` whose `TEMPLATE` attribute names a file using a

variable or other expression, since again the process of finding CFFUNCTION declarations does not happen at run time but instead in a previous step of declaration analysis.

For similar reasons, BlueDragon will also not analyze a CFINCLUDE that appears inside a CFIF or indeed any other nested tag. A CFINCLUDE holding a CFFUNCTION declaration must not occur inside any nested tags.

In summary, the rules for using CFINCLUDE within a CFCOMPONENT to declare component functions are:

1. The CFINCLUDE tags can't be nested within other tags (such as CFIF).

The filename specified in the TEMPLATE attribute of CFINCLUDE tags must not contain expressions.

2. A CFINCLUDE cannot include another file containing additional CFFUNCTION declarations.

## 6.6 Case Sensitivity in Java Method Calls

When accessing methods of Java objects (whether accessed by way of CFOBJECT/CreateObject() or when referring to variable scopes that may have Java objects within them), CFML (in both ColdFusion and BlueDragon) is case insensitive. BlueDragon does, however, try to do a case-sensitive match first (to see if the case you specified matches the case of an existing method in the Java object).

On the rare occasion where you need to call a method where there are 2 methods of the same name differing only by case then case becomes important. BlueDragon will attempt to resolve the ambiguity by performing a case sensitive match first. If that fails then an exception is thrown reporting that the ambiguity could not be resolved.

For example, consider attempting to use the isRequestedSessionidFromURL in the Java servlet page context, as in:

```
<cfset rqObj = getPageContext().getRequest()>
<cfoutput>#rqObj.isRequestedSessionidFromURL()#</cfoutput>
```

There are actually two methods of that name in that object. One is spelled with URL in caps while the other has url in lowercase. Which one should BlueDragon select? It doesn’t really matter which case the method is named with, since CFML is not case sensitive. You could even request it with:

```
<cfoutput>#rqObj.isrequestedsessionidfromurl()#</cfoutput>
```

Regardless of how you spell it, BlueDragon can’t determine which method you mean since there are two of the same name but with different case. The message you’ll see is:

```
    Method isRequestedSessionidFromURL is ambiguous as there is
more than one method that could correspond to the provided argu-
```

```
ment types. If possible, use 'javacast()' to resolve this ambigu-
ity.
```

Unfortunately, even use of `JavaCast()` won't help here. The message also refers to other situations of ambiguity (as when multiple methods exist accepting different arguments), and in those cases `JavaCast()` can help. In the case of two methods of the same name (and same arguments) with only case differentiating them, there's simply no way for BlueDragon to determine which to select. (ColdFusion instead chooses one of the two, with no apparent logic in which it chooses).