New Feature Notes
ADOBE® COLDFUSION® 9 Update 1

Adobe® ColdFusion® 9 Update 1

Adobe Systems Incorporated, 345 Park Avenue, San Jose, California 95110, USA.

Last updated 7/12/2010

Contents

ColdFusion 9 Update 1 New Feature Notes

Language enhancements

Support for for-in construct (for arrays)

You can loop over arrays in CFScript using for-in construct.

Example

```
public String foo(array a)
{
    for(var item in a)
    {
      writedump(item);
    }
}
```

var declaration within for loop

You can use `var` inline with for-in construct to bind variable to the local scope for both structs and arrays.

Example

```
public String foo(struct s)
{
    for(var item in s)
    {
      writedump(item & ": " & s[item]);
    }
    writedump(local);
}
```

For arrays example, see "Support for for-in construct (for arrays)" on page 1.

Support for creating custom metadata

You can specify custom metadata for function arguments in script syntax in either of the following ways:

- With arguments, as space-separated list of key-value pairs.
- In annotations, using `@arg1.custommetadata "custom value"`.

Example

custom.cfm

```
cfscript>
                writeoutput(new custom().foo(10));
</cfscript>
```

custom.cfc

```
/**
 * custom metadata for a cfc defined using annotation as well as key-value pairs
 * @cfcMetadata1 "cfc metadata1"
 */
component cfcMetadata2 = "cfc metadata2"
{
                /**
                * custom metadata for a property defined using annotation as well as key-value
pairs
                 * @propMetadata1 "property metadata1"
                 */
                property type="numeric" name="age" default="10" propMetadata2="property
metadata2";

                /**
            * custom metadata for a function/argument using both annotation and key-value pairs
                 * @arg1.argmdata1 "arg metadata1"
                 * output true
                 * @fnMetadata1 "function metadata1"
                 */
               public string function foo(required numeric arg1=20 argmdata2="arg metadata2")
fnMetadata2="function metadata2"
                {
                                writedump(getmetadata(this));
                                return arg1;
                }
}
```

CFScript support for cffile action="upload" and cffile action="uploadall"

CFScript support has been extended to the tags `cffile action="upload"` and `cffile action="uploadall"`.

The following are the upload functions:

FileUpload

Description
Uploads file to a directory on the server.

Returns
Struct that contains the result (or status) of file upload.

For details of what the struct contains, see the Usage section of `cffile action="upload"` in *ColdFusion 9 CFML Reference*.

Function syntax

```
FileUpload(destination)FileUpload(destination, filefield)FileUpload(destination, filefield,
accept)FileUpload(destination, filefield, accept, nameconflict)
```

Parameters

Value	Description
`destination`	Path of directory in which to upload the file. If not an absolute path (starting with a drive letter and a colon, or a forward or backward slash), it is relative to the ColdFusion temporary directory returned by the function `GetTempDirectory`. If the destination you specify does not exist, ColdFusion creates a file with the specified destination name. For example, if you specify the destination C:\XYZ, ColdFusion creates a file XYZ in the C: drive.
`accept`	Limits the MIME types to accept. Comma-delimited list. For example, the following code permits JPEG and Microsoft Word file uploads: `"image/jpg,application/msword".` The browser uses the filename extension to determine file type.
`nameConflict`	Action to take if file has the same name of a file in the directory. • `Error`: File is not saved. ColdFusion stops processing the page and returns an error. • `Skip`: File is not saved. This option permits custom behavior based on file properties. • `Overwrite`: Replaces file. • `MakeUnique`: Forms a unique filename for the upload; name is stored in the file object variable `serverFile`.
`fileField`	Name of form field used to select the file. Do not use number signs (#) to specify the field name.

See also

FileUploadAll

Usage

For details, see `cffile action="upload"` in *ColdFusion 9 CFML Reference*.

Example

```
<cfif isdefined("form.fileData")>
      <cfscript>
            hello = FileUpload("<path>","<mime type>","unique");
      </cfscript>
      <cfdump var="#hello#">
<cfelse>
      <cfform name="myUpload" enctype="multipart/form-data">
            <cfinput type="file" name="fileData"><br>
            <cfinput type="submit" name="submit">
      </cfform>
</cfif>
```

FileUploadAll

Description

Uploads all files sent to the page in an HTTP request to a directory on the server.

Returns

An array of struct that provides the file upload status.

For details of what the struct contains, see the Usage Section of `cffile action = "uploadAll"` in *ColdFusion 9 CFML Reference*.

Function Syntax

```
FileUploadAll(destination)

FileUploadAll(destination, accept)

FileUploadAll(destination, accept, nameConflict)
```

Parameter

Value	Description
destination	Path of directory in which to upload the file. If not an absolute path (starting with a drive letter and a colon, or a forward or backward slash), it is relative to the ColdFusion temporary directory, returned by the function `GetTempDirectory`. If the destination you specify does not exist, ColdFusion creates a file with the specified destination name. For example, if you specify the destination C:\XYZ, ColdFusion creates a file XYZ in the C: drive.
accept	Limits the MIME types to accept. Comma-delimited list. For example, the following code permits JPEG and Microsoft Word file uploads: `"image/jpg, application/msword"` The browser uses the filename extension to determine file type.
nameConflict	Action to take if file has the same name of a file in the directory. • `Error`: File is not saved. ColdFusion stops processing the page and returns an error. • `Skip`: File is not saved. Permits custom behavior based on file properties. • `Overwrite`: Replaces file. • `MakeUnique`: Forms a unique filename for the upload; name is stored in the file object variable `serverFile`.

See also

FileUpload

Usage

See `cffile action = "uploadAll"` in *ColdFusion 9 CFML Reference*.

Script Functions Implemented as CFCs

Function summary

The following table lists the script functions and the equivalent ColdFusion tag.

Function	Equivalent ColdFusion Tag
dbinfo	cfdbinfo
imap	cfimap
pop	cfpop
ldap	cfldap
feed	cffeed

dbinfo

Description

Used in CFScript to retrieve information about a data source such as database details, tables, queries, procedures, foreign keys, indexes, and version information about the database, driver, and JDBC.

Syntax

Mode	Syntax

Creating the service	`new dbinfo();`
	or
	`createObject("component", "dbinfo");`

Executing the service action	`dbinfoService.action_method(attribute-value_pair);`
Initializing the attributes	See "Initializing the attributes" on page 15.
Getting the CFC properties	See "Getting the CFC Properties" on page 15.
Working with the data returned	`data=dbinfoService.action_method(attribute-value_pair);` `writedump(data);`

Properties

`datasource`	`dbname`	`name`	`password`
`pattern`	table	`username`	

All attributes supported by the tag `cfdbinfo` can be used as attribute-value pairs. For example,

```
<cfdbinfo userName="myUserName">
```

can be used as

```
dbinfoService.setUserName("myUserName");
```

For details, see the Attributes section for the `cfdbinfo` tag in *ColdFusion 9 CFML Reference*.

See also

"Function summary" on page 4

History

ColdFusion 9 Update 1: Added this function.

Methods

The following dbinfo types are available as methods. All methods have similar arguments and syntax.

```
dbnames          tables          columns          version
procedures       foriegnkeys     index
```

Description	All methods correspond to the type of information supported by the tag `cfdbinfo`. For details of each method, see the relevant section for the tag `cfdbinfo` in *ColdFusion 9 CFML Reference*.
Returns	All methods return a query object.
Syntax	`dbinfoService.`***`methodName`***`(`***`attribute-value pair`***`);`
Arguments	All attributes supported by the tag `cfdbinfo`.

- `setAttributes`, `getAttributes`, `clearAttributes`, `clear`, `setProperties`, `getProperties`, `clearProperties`, and `clearProperties`. For details, see "Methods common to all functions" on page 14.

Usage

This function corresponds to the tag `cfdbinfo`. For details, see the Usage section for the tag `cfdbinfo` in *ColdFusion 9 CFML Reference*.

Example

```
<cfscript>

    d = new dbinfo(datasource=" cfartgallery ").dbnames(datasource="ajax");
    writedump(d);
    d = new dbinfo(datasource=" ajax").dbnames();
    writedump(d);
</cfscript>
```

imap

Description

Used in CFScript to query an IMAP server to retrieve and manage mails within multiple folders.

Syntax

Mode	Syntax

Creating the service	`new imap();`
	or
	`createObject("component", "imap");`

Initializing the attributes	See "Initializing the attributes" on page 15.
Executing the service action	`imapService.methodName(`***`attribute-value_pair`***`)`
Getting the CFC properties	See "Getting the CFC Properties" on page 15.
Working with returned data	`imapResult=imapService.`***`action_method`***`(`***`attribute-value_pair`***`);`

Properties

```
attachmentpath          connection          folder          generateuniquefilenames
```

maxrows	messagenumber	name	newfolder
password	port	recurse	secure
server	startrow	stoponerror	timeout
uid	username		

All attributes supported by the tag `cfimap` can be used as attribute-value pairs. For example,

```
<cfimap action="open" connection = "myconnection">
```

can be used as

```
imapService = new
imap(server="myimapserver",username="myusername",password="mypassword",port="myport",secure=
"yes");
imapService.open();
```

Note: *If connection properties such as server, username, password, port, and secure are specified either during initialization or when open method is called, a connection is created implicitly. Therefore, you need not specify the properties for further actions. If sandbox security is turned on, the directory referred to by the property* `attachmentPath` *must be given the required permission. By default, the temp directory is used.*

For details of the attributes, see the Attributes section for the tag `cfimap` in the *ColdFusion 9 CFML Reference*.

See also

"Function summary" on page 4

History

ColdFusion 9 Update 1: Added this function.

Methods

The following imap actions are available as methods. All methods have similar arguments and syntax.

getAll	delete	open	close
markRead	createFolder	deleteFolder	renameFolder
listAllFolders	moveMail	getHeaderOnly	

Description	All methods correspond to the type of information supported by the tag `cfimap`. For details of each method, see the relevant section of `cfimap` in the *ColdFusion 9 CFML Reference*.
Returns	A query object for methods `getAll`, `getHeaderOnly`, and `listAllFolders`. Else, nothing.
Syntax	`imapService.`***`methodName`***`(`***`attribute-value pair`***`);`
Arguments	All attributes supported by the tag `cfimap`.

- `setAttributes`, `getAttributes`, `clearAttributes`, `clear`, `setProperties`, `getProperties`, and `clearProperties`. For details, see "Methods common to all functions" on page 14.

Usage

This function corresponds to the tag `cfimap`. See the Usage section for `cfimap` in the *ColdFusion 9 CFML Reference* for details.

Example

```
<cfscript>

    m = new imap();
    m.setAttributes(server="#REQUEST.server#",username="#REQUEST.username#",
                    password="#REQUEST.password#",secure="#REQUEST.secure#",

connection="#REQUEST.connectionname#",stoponerror="#REQUEST.stoponerror#");
    m.open();
    master = m.getAll(connection = "#REQUEST.connectionname#",name = "queryname", stoponerror
= "#REQUEST.stoponerror#" );
writedump(master);

</cfscript>
```

pop

Description

Used in CFScript to retrieve or delete e-mail messages from a POP mail server.

Syntax

Mode	Syntax

Creating the service	`new pop();`
	or
	`createObject("component", "pop");`

Initializing the attributes	See "Initializing the attributes" on page 15.
Executing the service action	`popService.`***`action_method`***`(`***`attribute-value_pair`***`);`
Getting the CFC properties	See "Getting the CFC Properties" on page 15.
Working with data returned	`popresult = popService.action_method (`***`attribute-value`*** `pair);` where `popresult` is a query object if the `action_method` is `getAll` or `getHeaderOnly`. For any other method, nothing is returned.

Properties

`server`		`attachmentPath`	`debug`
`generateUniqueFilenames`	`maxRows`	`messageNumber`	`name`
`password`	`port`	`startRow`	`timeout`
`uid`	`username`		

All attributes supported by the tag `cfpop` can be used as attribute-value pairs. For example,

```
<cfpop server = "#form.popserver# " action = "getHeaderOnly" name = "GetHeaders">
```

can be used as

```
popHeaders = popService.getHeaderOnly(server="#form.popserver#");
```

Note: *`name` is a required attribute in `cfpop`, but not in CFScript.*

See also

"Function summary" on page 4

History
ColdFusion 9 Update 1: Added this function.

Methods
The following pop actions are available as methods. All methods have similar arguments and syntax.

`getHeaderOnly` `getAll` `delete`

Description	All methods correspond to the type of information supported by the tag `cfpop`. For details of each method, see the relevant section of `cfpop` in the *ColdFusion 9 CFML Reference*.
Returns	All methods except `delete` returns a query object.
Syntax	`popService.`***`methodName`***`(`***`attribute-value pair`***`)`
Arguments	All attributes supported by the tag `cfpop`.

- `setAttributes`, `getAttributes`, `clearAttributes`, `clear`, `setProperties`, `getProperties`, and `clearProperties`. For details, see "Methods common to all functions" on page 14.

Usage
This function corresponds to the tag `cfpop`. For usage details, see the Usage section for `cfpop` in the *ColdFusion 9 CFML Reference*.

Example
```
<cfscript>

    p = createObject("component","pop");
    p.setAttributes(server="#popServer#",username="failoveruser",password="#popPassword#");
    r = p.GetAll(name="results",maxRows = "2");

    writeoutput("getAll Passed<br>");

    r = p.GetAll(messageNumber = "2");
    writeoutput(#r.FROM# & "<br>");


    r= p.GETHEADERONLY(messageNumber = "1");
    writeoutput(#r.subject# & "<br>");


</cfscript>
```

ldap

Description
Used in CFScript to provide an interface to a Lightweight Directory Access Protocol (LDAP) directory server, such as the Netscape Directory Server.

Syntax

Mode	Syntax

Creating the service	`new ldap();`
	or
	`createObject("component", "ldap");`

Initializing the attributes	See "Initializing the attributes" on page 15.
Executing the service action	`ldapService.`***`action_method`***`(`***`attribute-value pair`***`);`
Getting the CFC properties	See "Getting the CFC Properties" on page 15.
Working with data	`ldapresult = ldapService.query(`***`attribute-value pair`***`).` For other methods, nothing is returned.

Properties

	`server`	`attributes`	`delimiter`
`dn`	`filter`	`maxRows`	`modifyType`
`name`	`password`	`port`	`rebind`
`referral`	`returnAsBinary`	`scope`	`secure`
`separator`	`sort`	`sortcontrol`	`start`
`startRow`	`timeout`	`userName`	

All attributes supported by the tag `cfldap` can be used as attribute-value pairs. For example,

```
<cfldap action="add" server="ldap.uconn.edu">
```

can be used as

```
ldapService.add(server="ldap.uconn.edu");
```

For details, see the Attributes section for the tag `cfldap` in the *ColdFusion 9 CFML Reference*.

Methods

The following ldap actions are available as methods. All methods have similar arguments and syntax.

`query`	`add`	`modify`	`modifyDn`
`delete`			

Description	All methods correspond to the actions supported by the tag `cfldap`. For details of each method, see the relevant section of `cfldap` in the *ColdFusion 9 CFML Reference*.
Returns	If method is `query`, returns a query object. Else, none.
Syntax	`ldapService.`***`methodName`***`(`***`attribute-value pair`***`)`
Arguments	All attributes supported by the tag `cfldap`.

- `setAttributes`. For details, see "Methods common to all functions" on page 14
- `getAttributes`, `clearAttributes`, `clear`, `setProperties`, `getProperties`, and `clearProperties`. For details, see "Methods common to all functions" on page 14.
- `setLdapAttributes`

Description	Sets the property `attributes`.
Returns	Nothing

Syntax	`ldapService.setLdapAttributes(attribute-value);`
Arguments	A string that contains the value of the property `attributes`.

- `getLdapAttributes`

Description	Gets the property `attributes`.
Returns	A string that contains the value of the property `attributes`.
Syntax	`myattributes = ldapService.getLdapAttributes();`

See also

"Function summary" on page 4

History

ColdFusion 9 Update 1: Added this function.

Usage

This function corresponds to the tag `cfldap`. For usage details, see the Usage section for `cfldap`.

Example

```
<cfscript>

    l = new ldap();
l.setLdapAttributes("objectclass=top, person, organizationalPerson, inetOrgPerson;cn=Joe
Smith; sn=Smith; mail=spenella@allaire.com; telephonenumber=(617) 761 - 2128");
l.setUsername("uid=admin,ou=system");
l.setPassword("administrator");
l.setPort(port);
l.setServer(ldapserver);

    l.setdn("ou=People+o=aribus.com,dc=example,dc=com");

    l.add();-

l.clearAttributes();result = l.query(name="apache",
            attributes="dn,cn,o,ou,c,mail,telephonenumber",
            start="dc=example,dc=com",
            scope="SUBTREE",
            filter="(&(cn=Joe Smith)(ou=people))";

    writeoutput("<b>Adding and Querying a LDAP entry : </b>" & "CN = " & result.CN & " DN = "
& result.DN & "<br> ");
    l.clearAttributes();
    l.delete(
                DN="ou=People+o=aribus.com,dc=example,dc=com",
                );


</cfscript>
```

feed

Description

Used in CFScript to read or create an RSS or Atom syndication feed. This service reads RSS versions 0.90, 0.91, 0.92, 0.93, 0.94, 1.0, and 2.0, and Atom 0.3 or 1.0. It can create RSS 2.0 or Atom 1.0 feeds.

Syntax

Mode	Syntax

Creating the service	`new feed()`
	or
	`createObject("component" "feed")`

Initializing the attributes	See "Initializing the attributes" on page 15.
Executing the service action	`feedService.`***`action_method`***`(`***`attribute-value_pair`***`)`
Getting the CFC properties	See "Getting the CFC Properties" on page 15.
Working with the data returned	• `feedresult = feedService.read(`***`attribute-value_pair`***`)` where `feedresult` is a struct with the keys `name`, `query`, `properties`, and `xmlvar`. • `feedresult = feedService.create(attribute-value_pair)` where `feedresult` is a string that contains the `xmlvar`.

Properties

`columnMap`	`enclosureDir`	`escapeChar`	`ignoreEnclosureError`
`name` (optional in CFScript)	`outputFile`	`overwrite`	`overwriteEnclosure`
`properties` (optional in CFScript)	`proxyPassword`	`proxyPort`	`proxyServer`
`proxyUser`	`query` (optional in CFScript)	`source`	`timeout`
`useragent`	`xmlvar` (optional in CFScript)		

All attributes supported by the tag `cffeed` can be used as attribute-value pairs. For example,

```
<cffeed action="read" source="http://googleblog.blogspot.com/atom.xml"
query="feedQuery" properties="feedMetadata" >
```

can be used as

```
feedservice.read(source="http://googleblog.blogspot.com/atom.xml",
query="feedQuery", properties="feedMetadata");
```

See also

"Function summary" on page 4

History

ColdFusion 9 Update 1: Added this function.

Methods

- `create`

Description	Creates an RSS 2.0 or Atom 1.0 feed XML document and saves it in a variable, writes it to a file, or both.

Returns	String representing the xmlvar
Syntax	`feedService.create (attribute-value pair);`
Arguments	All attributes supported by the tag `cffeed`.

- `read`

Description	Parses an RSS or Atom feed from a URL or an XML file and saves it in a structure or query. You can also get feed metadata in a separate structure.
Returns	Struct with the following keys: • name • query • properties • xmlvar
Syntax	`feedService.read (attribute-value pair);`
Arguments	All attributes supported by the tag `cffeed`.

- `setAttributes`, `getAttributes`, `clearAttributes`, `clear`, `setProperties`, `getProperties`, and `clearProperties`. For details, see "Methods common to all functions" on page 14.
- `getFeedProperties`

Description	Returns the value of the property `properties`.
Returns	Struct or error (if property is not set)
Syntax	`feedService.getFeedProeprties()`
Arguments	None

- `setFeedProperties`

Description	Sets the value of the property properties.
Returns	Nothing
Syntax	`feedService.setFeedProperties()`
Arguments	properties struct

Usage

This service corresponds to the tag `cffeed`. For usage, see Usage section for `cffeed` in the *ColdFusion 9 CFML Reference*.

Example

```
<cfscript>
        f = new feed();
    r = f.read(source=feedpath);

        writeoutput("Name : " & r.name.title & "<br>");
        writeoutput("Properties : " & r.properties.version & "<br>");
        writeoutput("Query : " & r.query.recordcount & "<br>");
        writeoutput("XMLVar : " & r.xmlvar.length() & "<br>");
</cfscript>
```

Methods common to all functions

The following methods are common to all script functions:

- `setAttributes`

Description	Sets attributes for the function.
Returns	Nothing
Syntax	***service_name*.*setAttributes*** (***attribute-value pair***);
Arguments	All attributes supported by the equivalent tag.

- `getAttributes`

Description	Gets the attributes set for the function.
Returns	Returns a struct with all or some attribute values.
Syntax	***service_name*.get*Attributes*** (***attributelist***);
Arguments	A comma-separated list of attributes. If no list is specified, all defined attributes are returned.

- `clearAttributes`

Description	Removes all attributes added for the function.
Returns	Nothing
Syntax	***service_name*.clearAttributes(*attribute_list*);**
Arguments	A comma-separated list of attributes.

- `clear`

Description	Removes all attributes added for the function.
Returns	Nothing
Syntax	***service_name*.clear();**
Arguments	None

- `clearProperties`

Description	Removes all properties added for the function.

Returns	Nothing
Syntax	***service_name***.clearProperties(***attribute_list***);
Arguments	If nothing is specified, all properties are cleared.

- setProperties

Description	Sets properties for the function.
Returns	Nothing
Syntax	***service_name.setproperties*** (***attribute-value pair***);
Arguments	All attributes supported by the equivalent tag.

- getProperties

Description	Gets the properties set for the function.
Returns	Returns a struct with all or some attribute values.
Syntax	***service_name***.getproperties (***attributelist***);
Arguments	A comma-separated list of attributes. If no list is specified, all defined attributes are returned.

Initializing the attributes

You can initialize the attributes using one of the following ways:

- *service_name*=new dbinfo(*attribute-value pair*)
- *service_name*=new dbinfo().init(attribute-value pair)
- *service_name*.setAttributes(*attribute-value pair*)
- *service_name*.setA*ttributeName(attribute_value)*
- *service_name.action_method*(*attribute-value_pair*)
- *service_name*.setProperties (*attribute_value*)

Getting the CFC Properties

Get the CFC properties using one of the following ways:

- *service_name*.getAttributeName(*attributelist*)
- *service_name*.getProperties (*attributelist*)
- *service_name*.getAttributes(*attributelist*)

Caching enhancements

New function cacheGetSession

Description

Lets you retrieve the underlying cache object to access additional cache functionality that is not implemented in the tag `cfcache`.

Returns

The underlying cache object.

Syntax

```
cacheGetSession()
```

Parameters

Parameter	Description
`objectType`	Any of the following values: • object • template • name of the user-defined cache
`isKey`	Set to `true` if `objectType` is user-defined cache. The default value is `false`.

History

ColdFusion 9 Update 1: Added this function

Example 1

The following example shows how to create a user-defined cache by adding an entry in ehCache.xml:

```
<cache
name="customcache"
maxElementsInMemory="1000"
eternal="false"
timeToIdleSeconds="720"
timeToLiveSeconds="720"
overflowToDisk="true"
diskSpoolBufferSizeMB="10"
maxElementsOnDisk="100000"
diskPersistent="true"
diskExpiryThreadIntervalSeconds="3600"
memoryStoreEvictionPolicy="LRU"/>
```

After you specify the details in the ehCache.xml, you can use the user-defined cache as shown here:

```
<!--- put an object into user-defined object cache --->
      <cfset cachePut("cache1","hello",15,15,customCache)>

      <!--- get underlying user-defined object cache --->
      <cfset objectCache = cachegetsession(customCache,true)>

      <!--- get/print user-defined object cache properties --->
      <cfset config = objectCache.getCacheConfiguration()>
      <cfoutput>
            getMaxElementsInMemory() :: #config.getMaxElementsInMemory()#<br>
            isEternal() ::    #config.isEternal()#<br>
            getTimeToIdleSeconds() :: #config.getTimeToIdleSeconds()#<br>
            getTimeToLiveSeconds() :: #config.getTimeToLiveSeconds()#<br>
            isOverflowToDisk() :: #config.isOverflowToDisk()#<br>
            getDiskSpoolBufferSizeMB() :: #config.getDiskSpoolBufferSizeMB()#<br>
            getMaxElementsOnDisk() :: #config.getMaxElementsOnDisk()#<br>
            isDiskPersistent() :: #config.isDiskPersistent()#<br>
            getDiskExpiryThreadIntervalSeconds() ::
#config.getDiskExpiryThreadIntervalSeconds()#<br>
            getMemoryStoreEvictionPolicy() :: #config.getMemoryStoreEvictionPolicy()#<br>
            isClearOnFlush() :: #config.isClearOnFlush()#<br>
      </cfoutput>
```

Example 2

The following example shows how to use the function `cacheGetSession` to operate on default caches:

```
<!--- put an object into user-defined object cache --->
      <cfset cachePut("cache1","hello",15,15)>

      <!--- get underlying user-defined object cache --->
      <cfset objectCache = cachegetsession("object",true)>

      <!--- get/print user-defined object cache properties --->
      <cfset config = objectCache.getCacheConfiguration()>
```

Other enhancements

EhCache Upgrade

EhCache has been upgraded to version 2.0.

New parameter for cacheGetMetadata

A new parameter `template` has been added to the function `cacheGetMetadata`. Use this parameter to get metadata for template caches.

Support for new configuration properties

ehCache.xml now includes the following properties:

- `diskSpoolBufferSizeMB`: Size to allocate the DiskStore for a spool buffer.

  The default size is 30 MB. Each spool buffer is used only by its cache.

  Turning on trace-level logging shows if backup for cache created/updated using `action="put"` occurs in the diskstore.

- `clearOnFlush`: Determines if the MemoryStore must be cleared when the cache is flushed. By default, the MemoryStore is cleared.
- `diskExpiryThreadIntervalSeconds`: The number of seconds between runs of the disk expiry thread. The default value is `120` seconds.

Note: *The functions `cacheGetProperties` and `cacheSetProperties` can be used to get/set these properties.*

Support for user-defined caches

Except in `cacheSetProperties` and `cacheGetProperties`, user-defined caches are supported in all caching functions.

Edit ehCache.xml (*cfroot*/lib)to set the properties for user-defined caches as shown in the following example:

```
<!--- item to put in user-defined cache --->
<cfset currentTime = Now()>
<!--- put item in user-defined cache --->
<cfset timeToLive=createtimespan(0,0,0,30)>
<cfset timeToIdle=createtimespan(0,0,0,30)>
<cfset customCache = "usercache">
<cfset id = "cache1">
<cfset cachePut(id,currentTime,timeToLive,timeToIdle,customCache)>
<!-- list items in the cache --->
List Items in cache:
<cfset cacheIds = cacheGetAllIds(customCache)>
<cfdump var="#cacheIds#"><br>
<!--- print cache data --->
<cfset cachedData = cacheGet(id,customCache)>
<cfoutput>#cachedData#</cfoutput>
<!--- print cache metadata --->
Cache metadata:
<cfset mdata = cacheGetMetadata(id,"object",customCache)>
<cfdump var="#mdata#">
<!--- clear user-defined cache --->
<cfset cacheRemove(ArrayToList(cacheIds),true,customCache)>
```

Support for IIS 7

IIS 7 configuration for ColdFusion has no dependency on IIS 6 Metabase compatibility.

The following configurations are required for using IIS 7 with ColdFusion. Follow the step that is common to all scenarios and then perform the configuration specific to your scenario.

Note: *Unlike in the previous releases where a common configuration file is used for all websites, IIS 7 configuration settings are specific to websites. Therefore, each website has a web.config file. This web.config file is present in the Content Directory (Document Root for that particular site). Ensure that you do not delete this file.*

Note: *Assume that you have all your sites configured (using the wsconfig.exe by selecting All). If you add a new site in IIS, the configuration details are not inherited. To configure the new site, invoke the wsconfig.exe, and then click Add. In the Add Web Server Configuration dialog box, select All in the IIS Web Site drop-down list and then click OK. By doing this, configurations are added only to the new sites and not all sites again. When you do this, you need not unconfigure all sites that you have already configured.*

Setup common to all scenarios:

❖ On Windows, go to the section Application Development Features (Start > Control Panel > Programs and Features > Turn Windows features on or off > Internet Information Services > World Wide Web Services) and then select the following:

- .Net Extensibility
- ASP.NET
- ISAPI Extensions and ISAPI Filters

Scenario 1: While installing, you configured ColdFusion 9 with native IIS 7

When you installed ColdFusion 9, you configured native IIS 7 by disabling IIS 6 Management Compatibility options. But native IIS 7 is not supported on ColdFusion 9. Therefore, for this scenario to be effective, you must update ColdFusion 9.

❖ Follow the installation instructions and update ColdFusion.

Scenario 2: While installing ColdFusion 9, you did not configure IIS

You did not configure IIS while installing ColdFusion 9. Now you are running the updater and want to configure IIS 7. In this case:

❖ Ensure that you have not selected any options under IIS 6 Management Compatibility (Start > Control Panel > Programs and Features > Turn Windows features on or off >Internet Information Services > Web Management Tools).

Scenario 3: IIS is configured for ColdFusion 9

You are using IIS 6 Metabase compatibility and you want to use IIS 7 support.

In this case,

1. Unconfigure the (already configured) websites.
2. Turn off all options under IIS 6 Management Compatibility (Start > Control Panel > Programs and Features > Turn Windows features on or off >Internet Information Services > Web Management Tools)
3. Reconfigure the websites.

Scenario 4: Continue using IIS 6 Metabase compatibility

You are using IIS 6 Metabase compatibility and you want to continue with it.

In this case,

❖ Do not do anything.

When you run applications in IIS 7 mode on remote machines (with IIS 6 Metabase compatibility turned off), if you encounter an exception, the detailed error message is not displayed. Instead, you see an error code.

To see the error details,

1. In the Internet Information Services (IIS) Manager Home, select Error Pages in the IIS section.
2. Double-click Error Pages, and then in the Actions pane, select Edit Feature Settings.
3. In the Edit Error Pages Settings dialog box, select the option Detailed errors.
4. Click OK.

ColdFusion Ajax

Support for CFCs outside webroot

Components outside the webroot can be accessed in bind expressions. This implies that tags such as `cfajaxproxy` or Ajax components such as grid, map, or progress bar can be used in more effective ways.

Note: *In the previous releases, the CFCs had to be web-accessible for Ajax applications to function.*

In addition to accessing CFCs using relative or absolute path, you can also use any of the following methods to access CFCs:

- logical mappings (defined in the ColdFusion Administrator)
- per-app mappings (defined in Application.cfc)
- imports (using `cfimport`/import)

Usage

The following code shows the usage of this enhancement using per-map mappings:

Application.cfc

```
THIS.mappings["/mycfc"] = "C:\www\shared\components";
```

Test.cfm

```
<cfajaxproxy cfc="mycfc.utils" jsclassname='jsobjname' />
```

Example

In this example, a per-app mapping named `mycfcs` has been created in Application.cfc pointing to "c:\components". For the sample code to work, create a folder named components in your system root (in this example, c:\) and copy the Employee.cfc to that folder.

Application.cfc

```
<cfcomponent>
    <cfset this.name = "cfcoutsidewebroot">
    <cfset this.sessionmanagement = true>
    <Cfset mappingname = "/mycfcs">
    <Cfset mappingpath = "c:\components\">
    <cfset this.mappings[mappingname] = mappingpath>
</cfcomponent>
```

Employee.cfc

```
<cfcomponent>
    <cfscript>
        remote any function
getEmployees(page,pageSize,gridsortcolumn="EMP_ID",gridsortdirection="ASC"){
            var startRow = (page-1)*pageSize;
            var endRow = page*pageSize;

            if(!isdefined("arguments.gridsortcolumn") or
isdefined("arguments.gridsortcolumn") and trim(arguments.gridsortcolumn) eq "")
                gridsortcolumn = "EMP_ID";
            if(!isdefined("arguments.gridsortdirection") or
isdefined("arguments.gridsortdirection") and arguments.gridsortdirection eq "")
                gridsortdirection = "ASC";
            var mysql = "SELECT Emp_ID, FirstName,  EMail, Department FROM Employees";
            if(isdefined("arguments.gridsortcolumn") and arguments.gridsortcolumn neq "")
                mysql = mysql & " ORDER BY " & gridsortcolumn;
            if(isdefined("arguments.gridsortdirection") and arguments.gridsortdirection neq
"")
                mysql = mysql & " " & gridsortdirection ;
            rs1 = new query(name="team", datasource="cfdocexamples", sql=mysql).execute();
                return QueryConvertForGrid(rs1.getResult(), page, pageSize);
        }


        remote any function editEmployees(gridaction,gridrow,gridchanged){
                writelog("edit employee info");
        }

    </cfscript>
</cfcomponent>
```

Employee.cfm

```
<cfform>
    <cfgrid
            format="html"
            name="grid01"
            pagesize=10
            title="Employee database"

bind="cfc:mycfcs.employee.getEmployees({cfgridpage},{cfgridpagesize},{cfgridsortcolumn},{cfg
ridsortdirection})"
            onChange="cfc:mycfcs.
employee.editEmployees({cfgridaction},{cfgridrow},{cfgridchanged})">
            <cfgridcolumn name="Emp_ID" display=false header="ID" />
            <cfgridcolumn name="FirstName" display=true header="First Name"/>
            <cfgridcolumn name="Email" display=true header="Email"/>
            <cfgridcolumn name="Department" display=true header="Department" />
    </cfgrid>
</cfform>
```

Modifications to cffileupload

- ColdFusion throws an error if the value of the attribute `maxuploadsize` exceeds the throttle/ post data size settings specified in the ColdFusion Administrator.
- `ColdFusion.FileUpload.getselectedfiles` returns the status of upload operation.

  The attribute `uploadStatus` has the following values:

  - `yes`: To indicate successful upload
  - `no`: To indicate failed upload
  - `error`: To indicate that an exception has occurred during the upload operation.
- `url` attribute is optional and it defaults to cgi.script_name.
- The fileupload control now passes the session information implicitly to the target page if session management is turned on either in Application.cfc or Application.cfm.

Serialization of Numeric Values

In the previous releases (including ColdFusion 9), serializing an integer using serializeJSON, converted the number to a double. For example, `SerializeJSON (123)` returns `123.0`.

Now, ColdFusion retains the integer in its original format. That is, `SerializeJSON (123)` returns `123`. This is applicable only to positive integers.

Note: *To retain previous behavior, a jvm argument `json.numberasdouble` has been provided.*

The following table gives more examples:

Input	Serialized JSON
10	10
012	"012" ***Note:*** *A number with leading zero is converted to a string.*
10.25	10.25
10.25E5	1025000.0
10.25E-5	1.025E-4
-10	-10.0
-10.25	-10.25

cfmap

The following are the `cfmap`-related changes:

- The valid values for `latitude/centerlatitude`: -90 to + 90
- The valid values for `longitude/centerlongitude`: (-180to +180)
- A new attribute `initshow=true|false;` default is `true`.

  Use this attribute to show/hide a map when the page loads. This is useful if you have collapsible divs or collapsible spry regions, where the user has to show the map on click of a link or button.

See also
`cfmap` in *ColdFusion 9 CFML Reference*

Grid enhancements

cfgrid

Added an attribute, `multirowselect=true|false` to allow selection of multiple rows. This is particularly useful in the cases where batch processing is required, for example, moving multiple records at a time.

By default, `multirowselect` is set to `false`. If `true`, a check box appears as the first column of the grid, enabling selection of multiple records. A Select All/Deselect All option also appears.

Note: *If `multirowselect=true`, then row data is sent as an array of structs as opposed to a struct if `mutirowselect=false`. Also, if the grid data is manipulated by the user, for example, using JavaScript, to move records when a button is clicked, set the method to POST. This is required as a GET method imposes restrictions on the amount of data that can be sent.*

cfgridcolumn

headerMenu
Added an attribute `headerMenu`.

```
Headermenu=true|false
```

The default is `false`.

The attribute allows you to turn on/off the header menu of the grid column. Header menu is the drop-down list that appears on grid header columns on mouse hover. The attribute is helpful when you have images for grid headers.

autoExpand
Added an attribute `autoExpand`.

```
autoExpand=true|false
```

On a particular column, it lets you expand the specified column.

Setting `autoExpand=true` for multiple columns results in error.

By default, `autoExpand` is set to `true` for the first column and `false` for the remaining columns.

Note: *If the attribute display is `false`, then `autoExpand` cannot be `true`; else, it results in error.*

Datehandling when the attribute mask is used
If the attribute `mask` is applied to a datefield column in an HTML grid, ColdFusion converts the date to an intermediate format as shown here:

```
MMMMM, dd yyyy HH:mms
```

for example,

```
January, 19 2005 07:35:42
```

This is required for proper date conversion and is applicable both when data is sent to the server (for example, when using an onChange grid event) and when data is received from the server (for example, populating a date field in a grid). Therefore, in some cases, users might have to format the date if they are updating a date column in the database.

Note: *Date values which are NULL are sent as empty strings when the form is submitted. In such cases, set the value to NULL explicitly while updating the date column in the database.*

dataAlign support for HTML grid

The attribute `dataAlign` is supported for HTML grids, in addition to Flash.

mask support for HTML grid

In addition to flash format grids, the attribute `mask` is also supported for HTML grids.

The default format is `m/d/y`, for example `05/06/75`.

where `m` is month with leading zeros, `d` is day with leading zeros, and `y` is two-digit representation of year.

For further details, go to the following URL:

http://www.extjs.com/deploy/dev/docs/output/Date.html

Modification to dynamic grid

In ColdFusion 9, data for the first row is available on form submission in a form with dynamic grid. In this release, the data is not available.

Other changes

If the type is `Boolean` and `selectmode` is `browse`, or `select=false`, the column is shown as a check box where click does not take effect.

Ajax plumbing

ORM CFCs support the attribute `remotingfetch` for a property.

By default, `remotingfetch` is set to `false`.

While serializing ORM CFCs, ColdFusion introspects the `remotingfetch` property and if it is `false`, does not return any relationship information.

If it is set to `true`, relationship information is shown. If circular reference is detected, only one level of relationship is shown.

Other enhancements

Assume that fileupload control is defined without a URL attribute. In this case, if the user chooses to upload data using the upload button, the control comes back to the same page. Users can check for `form.fieldnames` to perform the upload as shown in the following example:

Upload.cfm

```
<cfif isdefined("form.FIELDNAMES")>
      <cffile action = "upload" destination = "#ExpandPath('.')#" nameconflict="makeunique">
</cfif>
<cffileupload name="myuploader">
```

In this case, `url` defaults to CGI.script_name.

To maintain sessions between the fileupload control and the URL, users must turn on session management. You can do this by setting the `this.sessionmanagement=true` in Application.cfc. The setting ensures that CFID and CFtoken are passed as part of the URL if Enable J2EE Session Variables (ColdFusion Administrator > Server Settings > Memory Variables) is not selected. If it is selected, then JsessionID is passed as part of the URL.

JavaScript Functions

The following are the Ajax JavaScript functions added in this release:

ColdFusion.Autosuggest.getAutosuggestObject

Description

Lets you access underlying YUI AutoComplete object thereby providing fine-grained control over the object, for example attaching an event.

Returns

The underlying AutoComplete object.

Function syntax

```
ColdFusion.Autosuggest.getAutosuggestObject (Id)
```

Parameters

- `Id`: Name of the auto-suggest object.

Example

```
<html>
      <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
      <head>
            <cfajaximport tags="cfinput-autosuggest">
            <script>
                  var init = function()
                  {
                      autosuggestobj = ColdFusion.Autosuggest.getAutosuggestObject('state');
                        autosuggestobj.itemSelectEvent.subscribe(foo);
                  }
                  var foo = function(event,args)
                  {
                        var msg  = "";
                        msg = msg + "Event: " + event + "\n\n";
                        msg = msg + "Selected Item: " + args[2] + "\n\n";
                        msg = msg + "Index: " + args[1]._nItemIndex + "\n\n";
                        alert(msg);
                  }
                  var getStates = function(){
                        return
["California","Connecticut","Colorado","Illinois","Alabama","Iowa","Utah","Alaska"];
                  }
            </script>
      </head>
      <body>
            <h3>Attaching an event handler to the autosuggest object</h3>
            <cfform name="mycfform" method="post" >
                  State:<BR>
                  <cfinput
                        type="text"
                        name="state"
                        autosuggest="javascript:getStates({cfautosuggestvalue})"
                        autosuggestMinLength=1
                        autosuggestBindDelay=1>
                  <cfset ajaxOnLoad("init")>
            </cfform>
      </body>
</html>
```

ColdFusion.Layout.disableSourceBind

Description

Disables the source bind.

Function syntax

```
ColdFusion.Layout.disableSourceBind(Id)
```

Parameters

- `Id`: Name of the layout area.

Usage

Assume that you are using `Coldfusion.navigate` to populate content into tab or accordion panels. You can have instances where content comes from the source bind call if the `source` attribute is defined for `cflayoutarea` (and is not from `ColdFusion.navigate`).

In such instances, you might disable the source bind to get content using `Coldfusion.navigate`.

Example

layout.cfm uses the templates Tab1_Src.cfm, Tab2_Src.cfm, and Tab3_Src.cfm. If you run layout.cfm, you notice that clicking

- navigate populates content of tab2_src.cfm instead of navigate.cfm
- Disable Source bind ensures that the content of navigate.cfm is populated in tab2_src
- Enable Source Bind and then clicking tab2_src would again populate the content of tab2_src

Tab1_Src.cfm

```
<br><cfdump var="#CGI#" keys="15" label="[CGI scope]"><br>
```

Tab2_Src.cfm

```
<br><cfdump var="#server#" label="[Server scope]"><br>
```

Tab3_Src.cfm

```
<br><cfdump var="#server.coldfusion#" label="[Showing key coldfusion in server scope]"><br>
```

Tab4_Src.cfm

```
<br><cfdump var="#server.os#" label="[Showing key OS in server scope]"><br>
```

layout.cfm

```
<script>
      var navigateToTab = function(layoutId,tabId){
            alert("Navigating to " + tabId);
            ColdFusion.Layout.selectTab(layoutId,tabId);
            ColdFusion.navigate('navigate.cfm',tabId);
      }
      var disableBind = function(tabId){
            alert("Disabling binding on source for " + tabId);
            ColdFusion.Layout.disableSourceBind(tabId);
      }
      var enableBind = function(tabId){
            alert("Enabling binding on source for " + tabId);
            ColdFusion.Layout.enableSourceBind(tabId);
      }

</script>
<cflayout type="tab" name="layout1">
      <cflayoutarea
            name = "tab1"
            overflow = "auto"
            refreshonactivate = "yes"
            title = "Tab 1"
            source = "tab1_src.cfm"/>
      <cflayoutarea
            name = "tab2"
            overflow = "auto"
```

```
            refreshonactivate = "false"
            title = "Tab 2"
            source = "tab2_src.cfm"
            bindonload=false
            />
      <cflayoutarea
            name = "tab3"
            overflow = "auto"
            refreshonactivate = "yes"
            title = "Tab 3"
            source = "tab3_src.cfm"
            />
</cflayout>
<cfform name="myform">
      <cfinput type="button" name="disable" value="Disable Source Bind"
onClick="javascript:disableBind('tab2')">
      <cfinput type="button" name="b" value="Navigate"
onClick="javascript:navigateToTab('layout1','tab2')">
      <cfinput type="button" name="disable" value="Enable Source Bind"
onClick="javascript:enableBind('tab2')">
</cfform>
```

ColdFusion.Layout.enableSourceBind

Description

If disabled, enables the source bind.

Function syntax

`ColdFusion.Layout.enableSourceBind(Id)`

Parameters

- `Id`: Name of the layout area.

Usage

See usage in ColdFusion.Layout.disableSourceBind.

Example

See example in ColdFusion.Layout.disableSourceBind.

ColdFusion.FileUpload.getSelectedFiles

Description

Returns an array of objects containing the filename and size of the files selected for upload. The file size is returned in bytes.

The function also returns file upload status as `YES|NO|Error`.

Function syntax

`ColdFusion.FileUpload.getSelectedFiles(Id)`

Parameters

- `Id`: Name of the `cffileupload` control.

Usage

In a real life scenario, you normally use the uploader with other controls. For example, a form with three fields: name, email, and uploader. Assume that you upload the files, but forget to click Submit or you select the files, submit the form, but forget to click Upload.

You can use this function to inform the user that there are files that have been selected for upload and provide the following details:

- `FILENAME`: Name of the file selected for upload.
- `SIZE`: Size of the file in bytes.
- `UPLOADSTATUS`: `YES|NO|Error`

Example

The following example illustrates a scenario where the user clicks Submit and is informed about the files selected for upload:

```
<html>
<head>
    <script language="javascript">
        var formatNumber = function(num){
            if(num < 1024) return num + "  bytes";
            if(num < (1024 * 1024)) return (num/1024).toFixed(2) + " KB";
            if(num < (1024 * 1024  *1024)) return (num/(1024 * 1024)).toFixed(2) + " MB";
            return (num/(1024 * 1024 * 1024)).toFixed(2) + " GB";
        }
        var getSelectedList = function(id){
            var files = ColdFusion.FileUpload.getSelectedFiles(id);
            var fileslist = "";
            if(files.length)
                fileslist = "You have selected The following files for upload: \n\n";
            for(var i=0;i < files.length; i++){
                fileslist = fileslist + files[i].FILENAME + "  (" + formatNumber(files[i].SIZE)
+ ")"
                if(i != files.length-1)
                    fileslist = fileslist + "\r\n";
            }
            if(files.length)
            {
                alert(fileslist);
```

```
            }
        }
    </script>
</head>
<body>
<br>
<cfform name="frmUpload" method="POST">
    First Name: <cfinput type="text" name="fname" value=""><br>
    Last Name: <cfinput type="text" name="lname" value=""><br><br>
    <cffileupload
        url="uploadAll.cfm"
        name="myuploader1"
        hideUploadButton=false
        onUploadComplete="foo"
        /><br><br>
<cfinput type="button" name="submitForm2" value="Submit"
onClick="getSelectedList('myuploader1')">
</cfform>
</body>
</html>
```

Coldfusion.fileUpload.setUrl

Description

Used to set URL for the fileupload control dynamically.

Returns

Nothing

Function syntax

```
ColdFusion.fileUpload.setUrl(id, url)
```

Parameters

- `Id`: Name of upload control.
- `Url`: URL can be an absolute URL, relative URL, or fully qualified URL.

Example

```
<script language="javascript">
    var uploadDone = function(result){
        alert("File uploaded");
    }

    var setUploadUrl = function(id)
    {
        var selectedFiles = ColdFusion.FileUpload.getSelectedFiles(id);
        var uploadUrl = "/manual/ajaxui/cffileupload/setUrl/includes/_uploadall.cfm";
        alert("Upload URL : " + uploadUrl);
        if(selectedFiles.length){
            ColdFusion.FileUpload.setURL(id,uploadUrl);
            ColdFusion.FileUpload.startUpload(id);
        }
    }
    var callbackhandler = function(obj)
    {
        var fileName = obj["FILENAME"];
        var status = obj["STATUS"];
        var message = obj["MESSAGE"];
        var msg = "In callbackhandler()" + "\n\n" +
                  "FILENAME: " + fileName + "\n\n" +
                  "STATUS: " + status + "\n\n" +
                  "MESSAGE: " + message
        alert(msg);
    }
    var errorhandler = function()
    {
        alert("In errorhandler()");
    }
    var uploadcompleted = function()
    {
        alert("All files have been uploaded successfully");
    }
</script>
<cfform name="frmUpload">
    <br>
    <cffileupload name="uploader" hideuploadbutton="true" onComplete="uploadDone"
onError="errorhandler" onUploadComplete="uploadcompleted">
    <br>
    <cfinput type="button" name="submit" value="Click to set URL and Upload Files"
onClick="setUploadUrl('uploader')">
</cfform>
```

ColdFusion.grid.getSelectedRows

Description

Used to fetch data for the selected rows in the grid.

Returns

An array of objects that contains row data.

Function syntax

```
ColdFusion.grid.getSelectedRows(id)
```

Parameters

- `Id`: Name of the grid defined using `cfgrid`.

See also

FileUpload

Usage

See the example in ColdFusion.grid.clearSelectedRows.

Example

See the example in ColdFusion.grid.clearSelectedRows.

ColdFusion.grid.clearSelectedRows

Description

Used to clear the selected rows in the grid.

Returns

Nothing

Function syntax

```
ColdFusion.grid.clearSelectedRows(id)
```

Parameters

- `Id`: Name of the grid defined using `cfgrid`.

Usage

See the following example.

Example

Employee.cfm

```
<html>
<head>
    <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
    <cfajaxproxy cfc="emp" jsclassname="emputils">
    <script language="javascript">
        var emp = new emputils();
        var deleteAllSelectedRows = function(grid)
        {
            emp.setHTTPMethod("POST");
            emp.deleteEmployees(getAllSelectedRows(grid,false));
            ColdFusion.Grid.refresh(grid);
        }
        var getAllSelectedRows = function(grid,showalert)
        {
            obj = ColdFusion.Grid.getSelectedRows(grid);
            jsonbj = ColdFusion.JSON.encode(obj);
            if(showalert)
                alert(jsonbj);
            return obj;
        }
        var clearAllSelectedRows = function(grid)
        {
            ColdFusion.Grid.clearSelectedRows(grid);
        }
    </script>
</head>
<body>
<cfform>
    <cfgrid
        format="html"
        name="empListing"
        selectmode="edit"

bind="cfc:emp.getEmployees({cfgridpage},{cfgridpagesize},{cfgridsortcolumn},{cfgridsortdirec
tion})"
        onchange="cfc:emp.editEmployees({cfgridaction},{cfgridrow},{cfgridchanged})"
        autowidth="true"
        multirowselect=true
        delete="true"
```

```
        insert="true"
        title="Employee database"
        pagesize="25"
        >
        <cfgridcolumn name="EMP_ID" header="EMP_ID" select="false" display="false">
        <cfgridcolumn name="FIRSTNAME" header="First Name" select="true" />
        <cfgridcolumn name="LASTNAME" header="Last Name" select="true" />
        <cfgridcolumn name="DEPARTMENT" header="Department" select="true" />
        <cfgridcolumn name="EMAIL" header="Email" select="true" />
    </cfgrid>
    <br>
    <cfinput type="button" onClick="javascript:getAllSelectedRows('empListing',true)"
name="getRows" value="Get Selected Rows">
    <cfinput type="button" onClick="javascript:clearAllSelectedRows('empListing')"
name="clearRows" value="Clear Selected Rows">
    <cfinput type="button" onClick="javascript:deleteAllSelectedRows('empListing')"
name="deleteRows" value="Delete Selected Rows">
</cfform>
</body>
</html>
```

Employee.cfc

```
<cfcomponent>
    <cfscript>
        empQuery = new query(name="emps", datasource="cfdocexamples");
        remote any function
getEmployees(page,pageSize,gridsortcolumn="EMP_ID",gridsortdirection="ASC",empName)
        {
            var orderBy = "EMP_ID";
            var mysql = "SELECT Emp_ID, FirstName,  LastName, EMail, Department, Email FROM
Employees";
            if(isdefined("arguments.empName") and trim(arguments.empName) neq ""){
                mysql = mysql & " WHERE " & "firstname = '#arguments.empName#'";
            }
            if(arguments.gridsortcolumn eq ""){
                mysql = mysql & " ORDER BY " & orderBy;
            }
            mysql = mysql & " " & gridsortdirection;
            return QueryConvertForGrid(empQuery.execute(sql=mysql).getResult(), page,
pageSize);
        }
        remote void function editEmployees(gridaction,gridrow,gridchanged)
        {
            switch(gridaction)
            {
                case "I":
                {
                    var eFName = gridrow["FIRSTNAME"];
                    var eLNAme = gridrow["LASTNAME"];
                    var eDept = gridrow["DEPARTMENT"];
                    var eEmail = gridrow["EMAIL"];
                    var insertSql = "insert into
Employees(FirstName,LastName,Department,Email) values (" & "'" & eFName & "', '" & eLName &
"', '" & eDept & "', '" & eEmail & "')";
                    empQuery.execute(sql=insertSql);
                    break;
```

```
                }
                case "U":
                {
                    var empId = gridrow["EMP_ID"];
                    var changedCol = structkeylist(gridchanged);
                    var updateSql =  "UPDATE Employees SET " & changedCol & "='" &
gridchanged[changedCol] & "' WHERE emp_id=" & empId;
                    empQuery.execute(sql=updateSql);
                    break;
                }
                case "D":
                {
                    deleteEmployees(gridrow);
                }
            }
        }
        remote void function deleteEmployees(empdata)
        {
            var i = 1;
            var emp = {};
            if(isArray(empdata) and not ArrayIsEmpty(empdata)){
                for(emp in empdata){
                    if(isStruct(emp) and structkeyexists(emp,"emp_id")){
                        empid = emp["emp_id"];
                        writelog("deleting " & empid);
                        //var deleteSql =  "delete from Employees where emp_id=" & empid;
                        //empQuery.execute(sql=deleteSql);
                    }
                }
            }
        }

    </cfscript>
</cfcomponent>
```

In this example, setting `multirowselect=true` enables performing of batch operations on grid data, such as deleting multiple records.

In the `deleteemployees` functions, two lines have been commented out to prevent accidental deletion of data (since it is a batch operation). To see deletion, uncomment the code.

The form has a `deleteAllSelectedRows` button that illustrates how records can be deleted externally. That is, without using the delete button built in to the grid. The same approach can be used to perform other batch operations such as moving multiple files to another folder or batch updates.

Note: *Set the `httpMethod` to POST on the Proxy object carefully to avoid "request URI too large" errors as shown in the `deleteAllSelectedRows` method in Employee.cfm.*

ColdFusion.Map.show

Description

Shows the map if it is hidden.

Function syntax

```
ColdFusion.Map.show(Id)
```

Parameters

- `Id`: Name of the map.

Example

```
<script>
    function showMap(mapId)
      {
                  ColdFusion.Map.show(mapId);
    }

    function hideMap(mapId)
      {
                  ColdFusion.Map.hide(mapId);
    }
</script>
<a href="##" id="a1" onclick="return showMap('mainMap')">Show Map</a> | <a href="##" id="a1"
onclick="return hideMap('mainMap')">Hide Map</a>
<cfmap
      zoomlevel = "12"
    name = "mainMap"
    showcentermarker= "true"
    centeraddress = "The Key Learning centre, Oxford, UK"
      title="Venue Address"
      hideborder=false
      collapsible=true
      initShow=false/>
```

ColdFusion.Map.hide

Description

If displayed, hides the map.

Function syntax

```
ColdFusion.Map.hide(Id)
```

Parameters

- `Id`: Name of the map.

Example

See example in ColdFusion.Map.show

ColdFusion.Map.refresh

Description

Reloads the map.

Function syntax

```
ColdFusion.Map.refresh (Id)
```

Parameters

- `Id`: Name of the map.

Usage

If the map is embedded within spry collapsible panels or divs that are hidden on display, that is the map container is displayed while the actual map is hidden, use this function to force the map to display.

Example

```
<script type="text/javascript"
src="/CFIDE/scripts/ajax/spry/includes_minified/SpryCollapsiblePanel.js" ></script>
<link type="text/css"
href="/CFIDE/scripts/ajax/spry/widgets/collapsiblepanel/SpryCollapsiblePanel.css"
rel="stylesheet">
<div id="cp" class="CollapsiblePanel" style="width:500px;">
      <div class="CollapsiblePanelTab" tabindex="0">SHOW MAP</div>
    <div class="CollapsiblePanelContent">
    <cfmap
            width="500"
            height="200"
            zoomlevel="12"
            name="mainMap"
            markercolor="333444"
            showscale="false"
            typecontrol="none"
            showcentermarker="true"
            centeraddress="The Key Learning centre, Oxford, UK"
            >
      </cfmap>
      </div>
</div>
<script type="text/javascript">
      var myTabClick = function()
      {
            !cpanel.isOpen() ? cpanel.open() : cpanel.close();
            cpanel.focus();
            ColdFusion.Map.refresh('mainMap');
      }
      var cpanel = new Spry.Widget.CollapsiblePanel("cp", {contentIsOpen:false});
      cpanel.onTabClick = myTabClick;
</script>
```

ColdFusion.Grid.getTopToolbar

Description

Gets the top toolbar that can be used to add a control, for example icon or button.

Function syntax

```
ColdFusion.getTopToolbar(Id)
```

Parameters

- `Id`: Name of the grid.

Example
See example in ColdFusion.Grid.refreshBottomToolbar.

ColdFusion.Grid.getBottomToolbar

Description
Gets bottom toolbar that can be used to add a control, for example icon or button.

Function syntax
```
ColdFusion.Grid.getBottomToolbar(Id)
```

Parameters
- `Id`: Name of the grid.

Example
See example in ColdFusion.Grid.refreshBottomToolbar.

ColdFusion.Grid.showTopToolbar

Description
Displays the top toolbar that can be used to add a control, for example icon or button.

Function syntax
```
ColdFusion.Grid.showTopToolbar(Id)
```

Parameters
- `Id`: Name of the grid.

Example
See example in ColdFusion.Grid.refreshBottomToolbar.

ColdFusion.Grid.hideTopToolbar

Description
Hides the top toolbar that can be used to add a control, for example icon or button.

Function syntax
```
ColdFusion.Grid.hideTopToolbar(Id)
```

Parameters
- `Id`: Name of the grid.

Example
See example in ColdFusion.Grid.refreshBottomToolbar.

ColdFusion.Grid.showBottomToolbar

Description

Shows bottom toolbar that can be used to add a control, for example icon or button.

Function syntax

```
ColdFusion.Grid.showBottomToolbar(Id)
```

Parameters

- `Id`: Name of the grid.

Example

See example in ColdFusion.Grid.refreshBottomToolbar.

ColdFusion.Grid.hideBottomToolbar

Description

Hides the bottom toolbar that can be used to add a control, for example icon or button.

Function syntax

```
ColdFusion.Grid.hideBottomToolbar(Id)
```

Parameters

- `Id`: Name of the grid.

Example

See example in ColdFusion.Grid.refreshBottomToolbar.

ColdFusion.Grid.refreshTopToolbar

Description

Refreshes the top toolbar that can be used to add a control, for example icon or button. This function internally calls the JavaScript function ColdFusion.Grid.showTopToolbar.

Function syntax

```
ColdFusion.Grid.refreshTopToolbar(Id)
```

Parameters

- `Id`: Name of the grid.

Example

See example in ColdFusion.Grid.refreshBottomToolbar.

ColdFusion.Grid.refreshBottomToolbar

Description

Refreshes the bottom toolbar that can be used to add a control, for example icon or button. This function internally calls the JavaScript function ColdFusion.Grid.showBottomToolbar.

Function syntax

```
ColdFusion.Grid.refresheBottomToolbar(Id)
```

Parameters

- `Id`: Name of the grid control.

Example

grid.cfc

```
<cfcomponent>
      <cfscript>
            remote any function
getEmployees(page,pageSize,gridsortcolumn="EMP_ID",gridsortdirection="ASC"){
                  var startRow = (page-1)*pageSize;
                  var endRow = page*pageSize;

                  if(!isdefined("arguments.gridsortcolumn") or
isdefined("arguments.gridsortcolumn") and trim(arguments.gridsortcolumn) eq "")
                        gridsortcolumn = "EMP_ID";
                  if(!isdefined("arguments.gridsortdirection") or
isdefined("arguments.gridsortdirection") and arguments.gridsortdirection eq "")
                        gridsortdirection = "ASC";
                  var mysql = "SELECT Emp_ID, FirstName,  EMail, Department FROM Employees";
                if(isdefined("arguments.gridsortcolumn") and arguments.gridsortcolumn neq "")
                        mysql = mysql & " ORDER BY " & gridsortcolumn;
                  if(isdefined("arguments.gridsortdirection") and arguments.gridsortdirection
neq "")
                        mysql = mysql & " " & gridsortdirection ;
               rs1 = new query(name="team", datasource="cfdocexamples", sql=mysql).execute();
                        return QueryConvertForGrid(rs1.getResult(), page, pageSize);
            }


            remote any function editEmployees(gridaction,gridrow,gridchanged){
                        writelog("edit employee info");
            }

      </cfscript>
</cfcomponent>
```

grid.cfm

```
<script>
      var refreshToolbar = function(id,type){
            type == "top" ? ColdFusion.Grid.refreshTopToolbar(id) :
ColdFusion.Grid.refreshBottomToolbar(id);
      }

      var hideToolbar = function(id,type){
            type == "top" ? ColdFusion.Grid.hideTopToolbar(id) :
ColdFusion.Grid.hideBottomToolbar(id);
      }

      var showToolbar = function(id,type){
            (type == "top") ? ColdFusion.Grid.showTopToolbar(id) :
ColdFusion.Grid.showBottomToolbar(id);
      }

      var handleToolbar = function(id,type){
            if(type == "top"){
                  tbar = ColdFusion.Grid.getTopToolbar(id);
                  tbar.addButton({
                        text: "Add User Account",
                        tooltip: "Add a user account",
                        handler: addUserAccount
                  });
            }
            else{
                  bbar = ColdFusion.Grid.getBottomToolbar(id);
                  bbar.add(new Ext.Toolbar.Separator());
                  bbar.addButton({
                        text: "Delete User Account",
                        tooltip: "Delete a user account",
                        handler: deleteUserAccount
                  });
            }
      }

      var GetUserInfo = function(){
            alert("Retrieving user account");
      }

      var addUserAccount = function(){
            alert("Adding new user account")
      }
      var deleteUserAccount = function(){
            alert("Deleting user account")
      }
</script>
<cfform>
      <br>
      <cfinput type="button" onClick="showToolbar('empGrid','top')" name="btn1" value="Show
Top Toolbar">
      <cfinput type="button" onClick="handleToolbar('empGrid','top')" name="btn2" value="Add
button to Top Toolbar">
      <cfinput type="button" onClick="refreshToolbar('empGrid','top')" name="btn3"
value="Refresh Top Toolbar">
      <cfinput type="button" onClick="hideToolbar('empGrid','top')" name="btn4" value="Hide
Top Toolbar">
```

```
      <br><br>

      <cfgrid
                  format="html"
                  name="empGrid"
                  width="800"
                  pagesize=5
                  sort=true
                  title="Employee database"
                  collapsible="true"
                  insert="yes"
                  delete="yes"

bind="cfc:grid.getEmployees({cfgridpage},{cfgridpagesize},{cfgridsortcolumn},{cfgridsortdire
ction})"
               onChange="cfc:grid.editEmployees({cfgridaction},{cfgridrow},{cfgridchanged})"
                  selectMode="edit"
                  >
                  <cfgridcolumn name="Emp_ID" display=false header="ID" />
                  <cfgridcolumn name="FirstName" display=true header="First Name"/>
                  <cfgridcolumn name="Email" display=true header="Email"/>
                  <cfgridcolumn name="Department" display=true header="Department" />
      </cfgrid>

      <br><br>
     <cfinput type="button" onClick="hideToolbar('empGrid','bottom')" name="btn5" value="Hide
Bottom Toolbar">
     <cfinput type="button" onClick="showToolbar('empGrid','bottom')" name="btn6" value="Show
Bottom Toolbar">
    <cfinput type="button" onClick="handleToolbar('empGrid','bottom')" name="btn7" value="Add
button to Bottom Toolbar">
      <cfinput type="button" onClick="refreshToolbar('empGrid','bottom')" name="btn8"
value="Refresh Bottom Toolbar">
</cfform>
```

ORM enhancements

Support for multiple data sources for ORM

Introduction

You can use multiple data sources for ORM in ColdFusion applications. A multiple data source setup is useful in scenarios where your application has multiple modules that interact with each other.

Hibernate inherently supports single data source for a Hibernate configuration. To support multiple data sources, ColdFusion builds and manages multiple Hibernate configurations and SessionFactory objects, one for each data source in the application.

Usage scenario

Consider an application with the following three modules:

- HR

- Finance
- Sales

Assume that all these modules have their own databases (and therefore separate data sources). But at the application-level, all the three modules have to interact with each other. A single data source makes it impossible to build the entire application using ORM. Building three separate applications is not advisable as the interaction between the applications is possible only using web services.

If you use a multiple data source setup for ORM, all the three modules can be built in ORM. They can be part of the same application and the modules can interact with each other.

Configuring the application to use multiple data sources

Configure Persistent CFCs with the attribute `datasource` pointing to the appropriate data source. You can specify the attribute data source on the CFC using the tag `cfcomponent` or by specifying the annotation on Component in the CFC definition. If you do not specify a data source, the default data source is used for that CFC.

Since a Hibernate configuration uses a single data source, all related CFCs (using ORM relationships) must have the same data source.

Example

Art.cfc

```
<cfcomponent persistent="true" datasource="artgallery" table="Art">
  ...
</cfcomponent>
```

Author.cfc

```
<cfcomponent persistent="true" datasource="bookclub" table="author">
  …
</cfcomponent>
```

ORM settings

The following are the data source-specific ORM settings for which you can specify string or struct values in the Application.cfc:

- `schema`
- `catalog`
- `dialect`
- `dbcreate`
- `sqlscript`

For multiple data sources, a struct can be specified with data source name as the key and the appropriate setting as the value. If a string value is specified, it applies to the default data source of ORM.

Example 1

```
<cfset this.ormsettings.dbcreate={artgallery="dropcreate", bookclub="none"}>
```

Example 2

```
<cfset this.ormsettings.dbcreate="dropcreate">
```

If multiple data sources are used for ORM, these settings apply to the default ORM data source.

Mapping using Hibernate mapping files

In multiple data source scenarios, the data source information must be provided in the CFC (and not in .hbmxml file).

Also, all CFCs used in one .hbmxml file must have the same data source.

Example

The following example illustrates two different entities using two different data sources. In this example, art.cfc and artist.cfc are related and therefore use the same data source.

art.cfc

```
<cfcomponent persistent="true" table="art" datasource="cfartgallery">
<cfproperty name="ArtID" fieldtype="id" generator="native">
<cfproperty name="ArtName">
<cfproperty name="IsSold">
</cfcomponent>
```

artists.cfc

```
<cfcomponent persistent="true" table="artists" datasource="cfartgallery">
<cfproperty name="ArtistID" fieldtype="id">
<cfproperty name="FirstName">
<cfproperty name="LastName">
<cfproperty name="art" fieldtype="one-to-many" cfc="art" fkcolumn="ArtistID">
</cfcomponent>
```

authors.cfc

```
<cfcomponent persistent="true" table=authors datasource="cfbookclub">
<cfproperty name="AuthorID" fieldtype="id">
<cfproperty name="LastName">
<cfproperty name="FirstName">
</cfcomponent>
```

index.cfm

```
<cfoutput>Original Data<br></cfoutput>
<cfset artistObj = EntityLoad("artists", 1, true)>
<cfoutput>#artistObj.getArtistID()# | #artistObj.getFirstName()# |
#artistObj.getLastName()#<br></cfoutput>
<cfset artObj = artistObj.getart()>
<cfoutput>#artObj[1].getartname()# <br></cfoutput>
<cfset authorObj = EntityLoad("authors", 1, true)>
<cfoutput>#authorObj.getFirstName()#</cfoutput>
<cfoutput>#authorObj.getLastName()#</cfoutput>
```

Session Management

ColdFusion ORM uses session-per-request pattern.

In multiple data source scenarios, there are multiple sessions (one for each data source) in the same request. For all entity functions, the appropriate sessions are used transparently. ORM session-related functions also take optional data source argument. If you do not specify a data source, the default data source specified for ORM is used.

For a list of functions modified for multiple data source scenarios, see the section ORM Function Enhancements.

ORM Function enhancements

Multiple data source support impacts the following ORM functions:

ORMGetSession

Description
Returns the Hibernate session associated with the data source in the request. If ORM is not configured for this data source, it results in an exception. If data source is not specified, the Hibernate session of the default data source is returned.

Use this session object to call the APIs, which, otherwise, ColdFusion does not expose.

For information on session APIs, see:

http://docs.jboss.org/hibernate/core/3.3/api/org/hibernate/Session.html

Function syntax
```
ormgetsession([datasource])
```

ORMCloseSession

Description
Closes the Hibernate session associated with the data source in the request. If you do not specify a data source, the Hibernate session associated with the default data source is closed.

Function syntax
```
ormclosesession([datasource])
```

ORMCloseAllSessions

Description
Closes all Hibernate sessions in the request.

Function Syntax
```
ormcloseallsessions()
```

History
ColdFusion 9 Update 1: Added this function

ORMFlush

Description
Flushes the Hibernate session associated with the data source in the request. ORMFlush flushes all pending CRUD operations in the request. Any changes made in the objects, in the current ORM session, are saved to the database.

If you do not specify the data source, the Hibernate session associated with the default data source is flushed.

Function syntax
```
ormflush([datasource])
```

ORMFlushall

Description
Flushes all the current Hibernate sessions in the request.

Function syntax
```
ormflushall()
```

History
ColdFusion 9 Update 1: Added this function

ORMClearSession

Description
Clears the Hibernate session associated with the given data source.

The function clears the first level cache and removes the objects that are not yet saved to the database.

If you do not specify the data source, the Hibernate session associated with the default data source is cleared.

Function syntax
```
Ormclearsession([datasource])
```

ORMGetSessionFactory

Description
Returns the Hibernate Session Factory object associated with the data source. Results in an error if ORM is not configured for this data source. If you do not specify the data source, the Hibernate session factory object associated with the default data source is returned.

For information on Session API, go to the following URL:

http://docs.jboss.org/hibernate/core/3.3/api/org/hibernate/SessionFactory.html

Function syntax
```
Ormgetsessionfactory([datasource])
```

ORMEvictQueries

Description
This method is used to evict the data of all the queries from the default query cache of the specified data source. If cache name is specified, then the data of all queries belonging to the cache region with the given cache name are evicted.

If no data source is specified, the default query cache of the default data source is evicted.

Syntax
```
ORMEvictQueries([cachename])
ORMEvictQueries([cachename], datasource)
```

Parameter	Description
`cachename`	Name of the cache region that you want to evict.
`datasource`	Name of the data source whose cache you want to evict. If you do not specify the cache, the default query cache is evicted.

ORMExecuteQuery

Description

Executes a Hibernate Query Language (HQL) query.

By default, this function works on ORM's default data source. To use this function for another data source, specify the data source key-value pair within the `queryoptions`.

Syntax

```
ORMExecuteQuery(hql, [params] [,unique])
ORMExecuteQuery(hql, [,unique] [, queryoptions])
ORMExecuteQuery(hql, params [,unique] [,queryOptions])
```

Parameters

Parameter	Description
Hql	The HQL query that has to be executed.
Params	Object parameter for the entity.
Unique	Specifies if the object parameter is unique.
Queryoptions	Key-value pair of options for the query.

Example

```
<cfset artistArr = ORMExecuteQuery("from Artists where artistid=1", true,
{datasource="cfartgallery"})>
<cfset countArray = ORMExecuteQuery("select count(*) from Authors", [], false,
{datasource="cfbookclub"})>
```

Transaction Management

With ColdFusion ORM, you can manage transactions in the following two ways:

- **Using Hibernate transaction:** User has full control and ColdFusion does not intervene. The application has to flush/close the session and commit/rollback the transaction.

  For more information on transactions, go to the following URL:

  http://community.jboss.org/wiki/sessionsandtransactions

- **Using CFTransaction:** ColdFusion manages the transaction. Since a transaction cannot be distributed (across different data sources), application must ensure that the changes made in the transaction affect only one Hibernate session. That is, only one data source.

  ColdFusion allows reading of data from other sessions (data source) in a transaction but changes must be made in only one session. Multiple dirty sessions at any time in the transaction can result in exceptions and the transaction is rolled back. Before transaction begins, all existing sessions in the request are flushed. The previous session (if any) is reused.

When the transaction is committed, the dirty session is automatically flushed (before committing the transaction). When the transaction is rolled back, the changed session cannot be used any longer because it can cause rolled back data to get committed later. Therefore, the session participating in the transaction is cleared when transaction is rolled back.

Change of behavior in ColdFusion 9 Update 1

When..	Behavior in ColdFusion 9	Changed behavior
When the transaction starts	The existing session closes and a new session starts.	The existing session is flushed and is reused.
When the transaction is committed	The existing session is flushed and closed.	The existing session is flushed.
When the transaction is rolled back	The existing session is closed without flushing.	The existing session is cleared.

Other ORM enhancements

What's New	Where	Syntax	Description
Flag `skipCFCWithError`	`ormsettings` struct in the THIS scope of Application.cfc.	`skipCFCWithError=true|false` The default is `false`.	Lets you specify if ColdFusion must skip the CFCs that have errors. If set to `true`, ColdFusion ignores the CFCs that have errors.
Attribute `mappedSuperClass`	`cfcomponent`	`mappedSuperClass=true|false` The default is `false`.	If set to `true` on a non-persistent CFC, child CFCs can inherit its properties. For example, you can define a base CFC with common properties such as `ID`, `version`, or `createdOn` which all other persistent CFCs would extend and thus get one common behavior. The attribute `MappedSuperClass` cannot be set to `true` on a persistent CFC.
`automanageSession`	`ormsettings` struct in the THIS scope of Application.cfc.	`automanageSession=true|false` The default is `true`.	Lets you specify if ColdFusion must manage Hibernate session automatically. • If enabled: ColdFusion manages the session completely. That is, it decides when to flush the session, when to clear the session, and when to close the session. • If disabled: The application is responsible for managing flushing, clearing, or closing of the session. The only exception is (in the case of transaction), when the transaction commits, the application flushes the session. ColdFusion closes the ORM session at the end of request irrespective of this flag being enabled or disabled.
Changes to `missingrowignored`	`cfproperty`		`missingrowignored` is now supported for one-to-one relationships as well.
Attribute `ormoptions`	`cfquery`		A struct that takes orm options for executing HQL. Applies only if `dbtype` is set to `hql`.

EntityNew

Description
Creates an instance of the persistent CFC with the entity name that you provide.

Returns
Object

Syntax
```
entityNew(entityName [ ,properties])
```

Parameters

Parameter	Description
`entityName`	Entity name of the persistent CFC.
`properties`	Key-value pair (ColdFusion struct) of property names and values.

Usage
The enhancement allows the application to initialize the object that is being created. `properties` takes a struct with key being the property name. When the object is created, all the properties are populated with the passed struct.

Example
```
cfset artistObj = entityNew("Artists",{FirstName="Tom",LastName="Ron"})>
```

Using Amazon S3 storage

ColdFusion customers can now store data in Amazon S3. The support is extended across almost all tags and functions that take file or directory as input or output.

Storing files in Amazon S3 can be performed in the same manner as storing files on disk. Use a prefix `s3://` to indicate that the files reside on Amazon S3. For example, `s3://testbucket/sample.txt`.

Amazon S3

For using Amazon S3, ColdFusion user must have an S3 account with Amazon.

For concepts and details related to Amazon S3, see the AmazonS3 Documentation.

Accessing Amazon S3

Use either of the following URL formats to access Amazon S3 from ColdFusion:

- `s3://bucket/x/y/sample.txt`

  Here, `bucket` is the name of the bucket and the remaining portion of the URL is the key name of the Amazon S3 object.

  In this case, specify the following authentication information in the Application.cfc:

```
<cfscript>
        this.name ="Object Operations";
        this.s3.accessKeyId = "key_ID";
        this.s3.awsSecretKey = "secret_key";
        this.s3.defaultLocation="location";
        </cfscript>
```

For example,

```
<cffile action="write" output="S3 Specification" file="s3://testbucket/sample.txt"/>
```

- `s3://accessKeyId:awsSecretKey@bucket/x/y/sample.txt`
  - This format has the `accessKeyID` and `awsSecretKey` specified in it.
  - `@` acts as the token to indicate the end of authentication information.

  Note: *If you have specified the `accessKeyID` and `awsSecretKey` in both the URL and Application.cfc, then value specified in the URL takes precedence.*

Example

```
<cffile action="write" output="S3 Specifications"
file="s3://accessKeyID:awsSecretKey@bucket/x/y/sample.txt"/>
```

Supported operations

The following are the supported operations on Amazon S3:

- Create and delete bucket
- Get bucket location
- Read, write, copy, and delete object
- List keys in bucket
- Get and set metadata for object or bucket
- Get and set ACL for object or bucket

Bucket operations

Use the `cfdirectory` tag or the `directory` functions to perform the bucket operation (create, delete, or list).

Operation	Tag used	Function	Example
Create	`cfdirectory action="create"` The `directory` attribute can only take the path of the bucket. Any additional path results in an error. All other attributes are ignored. While creating S3 bucket, the default bucket location is `US`. You can change the location using the attribute `storeLocation`. `storeLocation` is a new attribute added to the `cfdirectory` tag. You can specify ACL for the bucket while creating it using the `storeACL` attribute which takes a struct value. For details, see "Setting up access control" on page 52.	`DirectoryCreate`	`<cfdirectory action="create" directory="s3://bucket1"/>`
List keys	`cfdirectory action="list"`	`DirectoryList`	`<cfdirectory action="list" directory="s3://bucket1/X/y" />` Since Amazon S3 does not have the concept of directory, it returns the key name (that is, the full path) of objects contained in the bucket. `Directory` attribute in this case takes the path, for example, `s3://bucket1` in which objects have to be searched. The path that follows the bucket name is used as a prefix to perform the list operation and all the objects that match the prefix are returned. In this case, the following attributes are ignored: `recurse`, `type`, and `sort`.
Delete	`cfdirectory action="delete"`	`DirectoryDelete`	`<cfdirectory action="delete" directory="s3://bucket1"/>`

Note: *To verify if the bucket exists and is accessible, use the function* `directoryExists`.

Object operations

All object operations are similar to file operations (read, write, copy, and delete). Therefore, the tag `cffile` and the file functions can be used to perform the operations. The following table describes the common scenarios:

Operation	Tag used	Function	Example
Read	`cffile action="read"`	`FileRead`	`<cffile action="read" file="s3://testbucket/test.txt" variable="data"/>`
Write	`cffile action="write"`	`FileWrite`	`<cffile action="write" output="#data#" file="s3://testbucket/test.txt"/>`
Delete	`cffile action="delete"`	`FileDelete`	`<cffile action="delete" file="s3://testbucket/test.txt"/>`
Copy	`cffile action="copy"`	`FileCopy`	`<cffile action="copy" source="s3://testbucket/test.txt" destination="s3://bucket2/a/b.txt"/>`

The following are the supported functions:

- `FileIsEOF`
- `FileReadBinary`
- `Filecopy`
- `FileReadLine`
- `FileExists`
- `FileWriteln`
- `FileOpen`
- `FileClose`
- `FileRead`
- `FileDelete`

New attributes in cfdirectory action="create" tag

Attribute Added	Description	Example
`storeLocation`	Used to change the location of the created bucket. The location can either be `EU` or `US`. The default location is `US`.	`<cfdirectory action="create" directory="s3://<bucketname>" storelocation="US"> <cfdirectory action="create" directory="s3://<bucketname>" storelocation="EU">`
`storeACL`	An array of struct where each struct represents a permission or grant as discussed in "ACLObject" on page 53.	`<cfdirectory action="create" directory="s3://<bucketname>" storeACL="ACLObject">`

Setting up access control

Amazon S3 lets you set access control list (ACL) for buckets and objects. The ACLs for buckets and objects are independent. You have to manage them separately. Also, object ACLs do not inherit from the bucket ACLs.

ACL consists of multiple Grants where each grant has a grantee and a permission. S3 allows three types of grantees:

- group
- email
- canonical (ID)

The following are the possible permissions:

- read
- write
- read_acp
- write_acp
- full_control

See Amazon S3 ACL Documentation for more details.

ACLObject

`ACLObject` is an array of struct where each struct represents an ACL grant. The grantee details are as follows:

group Must have the keys `Group` (with value `all`, `authenticated`, or `log_delivery`) and `permission`.

email Must have the keys `email` and `permission`.

canonical Must have the keys `Id` and `permission`. `displayName` is optional.

Sample ACLObject

```
all_read = {group="all", permission="read"};
   owner_full = {email="xxx@yyy.com", permission="full_control"};
   aclObj = [owner_full, all_read];
```

Access control functions

storeSetACL

Description

Sets the ACL for object or bucket.

Returns

Nothing

Syntax

```
StoreSetACL(url, ACLObject)
```

Parameters

Parameter	Description
`url`	Amazon S3 URLs (content or object)
`ACLObject`	An array of struct where each struct represents a permission or grant as discussed in "ACLObject" on page 53.

History

ColdFusion 9 Update 1: Added this function

Usage

Use this function to set full permission. The function overwrites all existing permissions. Only the ones you set in the current context exist.

Example

```
<cftry>
    <cfset dir = "s3://bucket_name">
    <cfif !directoryExists(dir)>
        <cfset directorycreate(dir)>
    </cfif>

    <cfset perm = structnew()>
    <cfset perm.group = "all">
    <cfset perm.permission = "read">
    <cfset perm1 = structnew()>
    <cfset perm1.email = "email ID">
    <cfset perm1.permission = "FULL_CONTROL">
    <cfset myarrray = arrayNew(1)>
    <cfset myarrray = [perm,perm1]>
    <cfset fileWrite("#dir#/test.txt","This is to test all users permission")>

    <cfset StoreSetACL("#dir#/textl.txt","#myarrray#")>
    <cfset test = StoreGetACL ("#dirkey#/test.txt") >

    <cfdump var= "test">

    <cfcatch>
        <cfdump var="#cfcatch#">
    </cfcatch>
</cftry>
```

storeAddACL

Description

Adds ACL to existing ACL for object or bucket.

Returns

Nothing

Syntax

```
StoreAddACL(url, ACLObject)
```

Parameters

Parameter	Description
`url`	Amazon S3 URLs (content or object).
`ACLObject`	An array of struct where each struct represents a permission or grant as discussed in "ACLObject" on page 53.

History

ColdFusion 9 Update 1: Added this function

Usage

Use this function to add permissions to the existing ones.

Example

```
<cftry>
    <cfset dir = "s3://bucket_name/">
    <cfset perm = structnew()>
    <cfset perm.group = "authenticated">
    <cfset perm.permission = "READ">
    <cfset perm1 = structnew()>
    <cfset perm1.email = "email_ID">
    <cfset perm1.permission = "READ_ACP">
    <cfset myarrray = [perm,perm1]>
    <cfif NOT DirectoryExists(dir)>
        <cfset directoryCreate(dir)>
    </cfif>
    <cfset fileWrite("#dir#/Sample.txt","This is to test StoreAddACL")>
    <cfset StoreAddACL("#dir#","#myarrray#")>
    <cfset test = StoreGetACL(dirkey)>
        <cfdump var="#test#">
    <cfcatch>
        <cfdump var="#cfcatch#">
    </cfcatch>
</cftry>
```

storeGetACL

Description

Gets the ACL object or bucket.

Returns

Returns an ACLObject

Syntax

```
StoreGetACL(url, ACLObject)
```

Parameters

Parameter	Description
`url`	Amazon S3 URLs (content or object)
`ACLObject`	An array of struct where each struct represents a permission or grant as discussed in "ACLObject" on page 53.

History

ColdFusion 9 Update 1: Added this function

Example

```
<cfset dir = "s3://bucket_Name">
    <cfif NOT DirectoryExists(dir)>
         <cfset directoryCreate(dir)>
        </cfif>
    <cfset test = StoreGetACL("#dir#")>
    <cfdump var="#test#">
```

Using metadata

Amazon S3 allows you to specify metadata for both objects and buckets.

The following two functions let you get and set the metadata on objects or buckets.

StoreGetMetadata

Description

Returns the metadata related to the object or bucket.

Returns

Object metadata or bucket metadata

Syntax

```
StoreGetMetadata(url)
```

Parameters

Parameter	Description
`url`	Amazon S3 URLs (bucket or object).

History

ColdFusion 9 Update 1: Added this function

Example

```
<cfdump var = #StoreGetMetadata("bucket_Name")#>
```

StoreSetMetadata

Description

Sets the metadata on bucket or object.

Returns

Nothing

Syntax

```
StoreSetMetadata(url,Struct)
```

Parameters

Parameter	Description
`url`	Amazon S3 URLs (bucket or object).
`struct`	Represents the metadata. See "Standard keys" on page 57 for a list of standard keys in metadata. You can also have custom metadata apart from the standard ones.

History

ColdFusion 9 Update 1: Added this function

Example

```
<cfscript>
      mydate = #Now()#;
      hello = structNew();
      hello.color = "grey";
/cfscript>

<cfset dir = "s3://mycfbucket">
      <cffile action="write" file="#dir#/hello5.txt" output="Sample s3 text">

      <cfset StoreSetMetadata("#dir#/hello5.txt","#hello#")>
      <cfset test = StoreGetMetadata("#dir#/hello5.txt")>

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

Standard keys

The following are the standard keys in the metadata:

For objects

- last_modified
- date
- owner
- etag
- content_length
- content_type
- content_encoding
- content_disposition
- content_language
- content_md5
- md5_hash

For buckets

- date
- owner

Security considerations

Sandboxing is not applicable to S3 buckets or objects as Amazon S3 has its own security features that take care of it.

Supported functions

fileOpen	fileClose	fileCopy	fileDelete
fileExists	fileisEOF	fileMove	fileWrite
fileRead	fileReadBinary	fileReadLine	fileSetLastModified
getFileInfo	getDirectoryFromPath	directoryCreate	directoryDelete
directoryExists	directoryList	imageNew	imageRead
imageWrite	imageWriteBase64	isImageFile	isPDFFile

Supported tags

All `cffile` actions	All `cfdirectory` actions (except rename)	`cfdocument`	`cffeed`
`cfftp`	`cfimage`	`cfloop`	All `cfimage` actions

Limitations

- The following tags are not supported:
  - `cfpdf`
  - `cfpdfform`
- The following functions are not supported:
  - `FileSetAccessMode` that sets attributes of a file in Linux/UNIX
  - `FilesSetAttribute` that sets the attributes of a file in Windows
  - `cfzip` does not accept Amazon S3 object as source.
  - When S3 object is used as output for `outputfile` attribute of `cfexecute` tag, it results in an error `Timeout period expired without completion of <exe>`. It also results in a NullPointerException at server console.
  - To use the function `fileMove`, the source and destination objects must have the same bucket name. That is, you cannot move Amazon S3 objects across buckets or to other file systems.

SpreadSheet enhancements

A new function `SpreadsheetRemoveSheet` has been added in this release.

SpreadsheetRemoveSheet

Description
Deletes a spreadsheet.

Returns
Nothing

Category
Microsoft Office Integration

Function syntax

```
SpreadsheetRemoveSheet (spreadsheetObj, sheetname)
```

Parameters

Parameter	Description
`spreadsheetObj`	The Excel spreadsheet object from which you delete the sheet.
`sheetname`	Name of the sheet that must be removed.

Example

```
<cfset spreadsheetVar= spreadSheetNew("New")>
<cfset spreadsheetCreateSheet(spreadsheetVar,"A")>
<cfset spreadsheetCreateSheet(spreadsheetVar,"B")>
<cfspreadsheet action="write" filename="#dirname#mySpreadSheet.xls"  name="spreadsheetVar"
overwrite="true" >
<cfspreadsheet action="read" src="#dirname#mySpreadSheet.xls" name="spreadSheetVar" >
<cfset spreadsheetRemoveSheet(spreadsheetVar,"B")>
<cfspreadsheet action="write" filename="#dirname#mySpreadSheet.xls"  name="spreadsheetVar"
overwrite="true" >
```

SpreadsheetFormatCellRange

Description

Formats the cells within the given range.

Returns

Nothing

Category

Microsoft Office Integration

Function syntax

```
SpreadsheetFormatCellRange (spreadsheetObj, format, startRow, startColumn, endRow, endColumn)
```

Parameters

Parameter	Description
`spreadsheetObj`	The Excel spreadsheet object for which you want to format the cells.
`format`	A structure that contains format information.
`startRow`	The number of the first row to format.
`startColumn`	The number of the first column to format.
`endRow`	The number of the last row to format.
`endColumn`	The number of the last column to format.

Example

```
<!--- Get the spreadsheet data as a query. --->
<cfquery
    name="courses" datasource="cfdocexamples"
    cachedwithin="#CreateTimeSpan(0, 6, 0, 0)#">
    SELECT CORNUMBER,DEPT_ID,CORLEVEL,COURSE_ID,CORNAME,CORDESC,LASTUPDATE
    FROM COURSELIST
</cfquery>
<cfscript>
    ///We need an absolute path, so get the current directory path.
    theFile=GetDirectoryFromPath(GetCurrentTemplatePath()) & "courses.xls";
    //Create a new Excel spreadsheet object and add the query data.
    theSheet = SpreadsheetNew("CourseData");
    SpreadsheetAddRows(theSheet,courses);
    // Define a format for the column.
    format1=SructNew()
    format1.font="Courier";
    format1.fontsize="10";
    format1.color="dark_blue;";
    format1.italic="true";
    format1.bold="true";
    format1.alignment="left";
    SpreadsheetFormatCellRange(theSheet,format1, 3,4,30,10);
</cfscript>
<!--- Write the spreadsheet to a file, replacing any existing file. --->
<cfspreadsheet action="write" filename="#theFile#" name="theSheet"
sheet=1 sheetname="courses" overwrite=true>
```

Modifications to cfspreadsheet

A new attribute `excludeHeaderRow` has been added with `true|false` as the supported values. The default value is `false`. If set to `true`, excludes the headerrow from being included in the query results.

The attribute helps when you read Excel as a query. When you specify the `headerrow` attribute in `cfspreadsheet`, the column names are retrieved from the header row. But they are also included in the first row of the query. To not include the header row, set `true` as the attribute value.

Other enhancements

- Performance improvements in the case of formatting huge number of rows and columns using the following functions:
  - `Spreadsheetformatrows`
  - `Spreadsheetformatcolumns`
  - `Spreadsheetformatrow`
  - `Spreadsheetformatcolumn`
- Allows preformatting of a cell while you use `SpreadSheetformatcell` or `SpreadSheetformatcellrange` as shown in the following example:

```
<cfscript>
sheet= SpreadSheetNew();
Spreadsheetformatcell(sheet,{dataformat="@"},1,1);
spreadsheetSetCellValue(sheet,'000006534',1,1);
</cfscript>
```

Here, the cell is preformatted and the data is taken as it is provided.

- Support for vertical alignment in format struct using the key `verticalalignment`. The following values are supported for `verticalalignment`:
  - `VERTICAL_TOP`
  - `VERTICAL_BOTTOM`
  - `VERTICAL_CENTER`
  - `VERTICAL_JUSTIFY`

  Example:

```
<cfscript>
SpreadsheetFormatCellRange(theSheet,{verticalalignment="VERTICAL_TOP"}, 3,4,30,10);
</cfscript>
```

  Note: *The parameters `vertical_top`, `vertical_bottom`, `vertical_center`, and `vertical_justify` have been removed from the key `alignment` in the format struct used in the Spreadsheet format functions.*

AIR integration in ColdFusion

The AIR integration feature introduced in ColdFusion 9 has an ActionScript ORM for persisting entities in the SQLite database present within Adobe Integrated Runtime (AIR). This release has the following enhancements for this ActionScript ORM:

- Support for auto-generating primary keys
- Support for encrypted database (introduced in AIR 1.5).
- Cache file used by ActionScript ORM to track the operations on SQLite database is now in the applicationStoragedirectory instead of applicationDirectory. You can specify the location of the cahceDirectory in openSession API on syncmanager.
- Supports Self Join relationships for one-to-one, one-to-many, and many-to-many database relationships.
- Supports both Array and ArrayCollection for use in ActionScript Entity to represent a collection in a database relationship.
- ActionScript ORM logs all the SQL statements that ORM uses to persist entities into the SQLite database.
- New APIs `keepClientObject` and `keepAllClientObjects` to ensure that the server updates are not retained when ColdFusion server raises conflict.
- The class `SessionToken` is dynamic and therefore, data can be stored on the token returned from the ORM APIs.
- Supports autocommit mode

Auto-generating primary keys

This release supports primary key generation for the ActionScript ORM using the metadata tag `GeneratedValue`.

GeneratedValue

Description
Adding this tag on an ActionScript primary key file auto-generates primary key.

Parameters

Parameter	Description
`strategy`	UUID uses the Flash UUID API to generate the ID (used for primary key of type `string`) or INCREMENT (for primary key of type `int`).
`initialValue`	Applies only for INCREMENT strategy. Specifies the initial value of the primary key. The default value is 0.
`incrementBy`	Applies only for INCREMENT strategy. Specifies the integer with which the value must be incremented to generate the primary key.

If the ID value is not present in the object, the value is generated and is assigned the primary key value. If the key value is already present on the object instance, then the key generation is ignored.

For integer primary keys, the database table is checked for the presence of existing primary keys. If the highest key value is greater than the `initialValue`, then the key that is generated next will be an increment of the highest key value. For example, if the `initialValue` that you specify is `1`, and the database (already) has a key value `5`, then the next key is generated with the value `6` (5+1, if `incrementBy` is set to `1`).

Example

```
//Integer Primary Keys
===========
package test.apollo.CFSQLiteSupport.INCREMENTPK
{
    [Entity]
    [RemoteClass(alias="Customer")]
    public class Customer
    {
        public function Customer()
        {
        }
        [Id]
        [GeneratedValue(strategy="INCREMENT",initialValue=5,incrementBy=2) ]
        public var cid:int;
        public var name:String;

        [OneToOne(mappedBy="customer")]
        public var ord:Order;
    }
}
```

```
//String Primary Keys
===========
package test.apollo.CFSQLiteSupport.UUIDPK
{
    [Entity]
    [RemoteClass(alias="Customer")]
    public class Customer
    {
        public function Customer()
        {
        }
        [Id]
        [GeneratedValue(strategy="UUID") ]
        public var cid:String;
        public var name:String;

        [OneToOne(mappedBy="customer")]
        public var ord:Order;
    }
}
```

Encrypting the database

You can protect the database used by ActionScript ORM with an encryption key.

Use the `ByteArray` encryption key for `syncmanager.openSession` method to encrypt the database. The user-specified database file and the cache database file (used by the ActionScript ORM) are both encrypted using the encryption key you specify.

The key is optional.

Example

```
dbFile = File.userDirectory.resolvePath("customerManger.db");
    dbDir = File.applicationStorageDirectory;
    var keyGenerator:EncryptionKeyGenerator = new EncryptionKeyGenerator();
    var encryptionKey:ByteArray = keyGenerator.getEncryptionKey("UserPassword");

    var sessiontoken:SessionToken
=syncmanager.openSession(dbFile,179176,encryptionKey,dbDir);
```

For details on `EncryptionKeyGenerator`, see the section Using the EncryptionKeyGenerator class to obtain a secure encryption key in *Developing Adobe AIR 1.5 Applications with Flex.*

Specifying the cache directory

The cache directory where you store the cache file can be specified using the `cacheDirectory` (instance of `flash.filesystem.File`) for the `syncmanager.openSession` method.

The `cacheDirectory` is optional.

Note: *By default, the cache file used by the ActionScript ORM is stored in the File.applicationStorageDirectory (in ColdFusion 9, it was stored in File.applicationDirectory).*

For example, see "Encrypting the database" on page 63

Support for self joins

Database table can be related to itself through a foreign key. A typical example is an Employee table with a manager relationship containing the employee id of the managers (who manage the employee).

The manager id refers to another row in the same table. This is an example of one-to-one self join.

There can be one-to-many self join and many-to-many self joins with an intermediate join table.

ColdFusion 9 Update 1 has self join support for all the relationships in the ActionScript ORM.

The following ActionScript class definition for customer entity illustrates how all the self-join relationships are defined:

```
    package
{

    [Bindable]
    [RemoteClass(alias="AIRIntegration.customer")]
    [Entity]
    public class Customer
    {
        [Id]
        [GeneratedValue(strategy="INCREMENT",initialValue=5,incrementBy=2)]
        public var cid:int;
        public var name:String;

        [OneToOne(cascadeType='ALL',fetchType="EAGER")]
        [JoinColumn(name="add_id",referencedColumnName="aid")]
        public var address:Address;

        // Many-to-One self Join
        [ManyToOne(targetEntity="onetoone::Customer",fetchType="EAGER")]
        [JoinColumn(name="managerId",referencedColumnName="cid")]
        public var manager:Customer;

        // One-to-one Self Join
        [OneToOne(targetEntity="onetoone::Customer",fetchType="EAGER")]
        [JoinColumn(name="spouseId",referencedColumnName="cid",unique="true")]
        public var spouse:Customer;

       // Many-to-Many self Join
```

```
        [ManyToMany(targetEntity="onetoone::Customer",fetchType="EAGER")]
        [JoinTable(name="CUSTOMER_PARENTS_MAPPINGS")]
        [JoinColumn(name="CUST_ID",referencedColumnName="cid")]
        [InverseJoinColumn(name="PARENT_ID",referencedColumnName="cid")]
        public var parents:Array;


        // Many-to-Many self Join
        [ManyToMany(targetEntity="onetoone::Customer",fetchType="EAGER")]
        [JoinTable(name="CUSTOMER_CHILDREN_MAPPINGS")]
        [JoinColumn(name="CUST_ID",referencedColumnName="cid")]
        [InverseJoinColumn(name="CHILD_ID",referencedColumnName="cid")]
         public var children:Array;

[OneToMany(targetEntity="onetoone::Order",cascadeType='REMOVE',mappedBy="customer",fetchType
="EAGER")]
         public var orders:Array;

    }
}
```

ArrayCollection to hold multiple entities

In addition to Array, you can now use ArrayCollection to hold multiple entities in a database relationship. ArrayCollection can also be used in the ActionScript entities as Arrays are used to represent the related entities.

Example

```
package
{
    import mx.collections.ArrayCollection;

    [Bindable]
    [RemoteClass(alias="AIRIntegration.customer")]
    [Entity]
    public class Customer
    {
        [Id]
        [GeneratedValue(strategy="INCREMENT",initialValue=5,incrementBy=2)]
        public var cid:int;
        public var name:String;

        [OneToOne(cascadeType='ALL',fetchType="EAGER")]
        [JoinColumn(name="add_id",referencedColumnName="aid")]
        public var address:Address;

[OneToMany(targetEntity="onetoone::Order",cascadeType='REMOVE',mappedBy="customer",fetchType
="EAGER")]
        public var orders:ArrayCollection;

    }
}
```

Server-side configuration

See the section "Flash Remoting enhancements" on page 68.

Logging SQL statements

The ActionScript ORM logs all SQL statements that it executes.

The log can be configured as follows:

❖ Add a log target for the AIR applications as shown in the following example:

```
var logTarget:TraceTarget = new TraceTarget();
logTarget.filters = "*";
logTarget.level = LogEventLevel.ALL;
Log.addTarget(logTarget);
```

The log target is the TraceTarget where all the trace statements appear. The log target can be set to any other log using the Flash APIs.

ColdFusion ActionScript APIs

The following two APIs have been introduced to the session class in the `coldfusion.air` package:

keepAllClientObjects

Description

Takes an `ArrayCollection` of conflict instances and keeps the client object for every conflict instance in the ArrayCollection.

Returns

An instance of `coldfusion.air.SessionToken` (which is the token for `keepAllClientObjects` call).

Syntax

```
public function keepAllClientObjects(conflicts:ArrayCollection):SessionToken
```

Parameters

Parameter	Description
`mx.collections.ArrayCollection`	An `ArrayCollection` of conflicts raised by the server.

Example

```
private function conflictHandler(event:ConflictEvent):void
{
   // Alert.show("Server returned a Conflict !");
    var conflicts:ArrayCollection = event.result as ArrayCollection;
    // Ignore Server data and retain client Data in  SQLite DB
    var token:SessionToken = session.keepAllClientObjects(conflicts);
    token.addResponder(new mx.rpc.Responder(conflictSuccess, conflictFault));
}
```

keepClientObject

Description

Ensures that the client object is retained instead of the one from the server (despite server raising data conflict).

The API also ensures that the retained client object is not sent to the server as a new operation on sync.

Returns

An instance of `coldfusion.air.SessionToken` associated with `keepClientObject` call.

Syntax

```
public function keepClientObject(conflict:coldfusion.air.Conflict):SessionToken
```

Parameters

Parameter	Description
`coldfusion.air.Conflict`	The conflict that the server raises.

Example

See the example in the section keepAllClientObjects. For `keepClientObject`, the only difference is that you must iterate over each conflict in the conflictarray collection.

Offline AIR SQLite API enhancements

The following new parameters for `openSession`:

New Parameters	Type	Required/Optional	Description
`encryptionKey`	`ByteArray`	Optional	Used to encrypt offline SQLite database. For details, see "Encrypting the database" on page 63.
`cacehDirectory`	`File`	Optional	Used to specify a custom cache directory. For details, see "Specifying the cache directory" on page 63.

SessionToken class is dynamic

A class is dynamic in ActionScript if you can add additional key-value pairs to the instance of the class.

In this release, `sessionToken` is dynamic class. Therefore, you can add additional information that can be passed from where the API is called to the success or fault handlers.

Example

```
private function fetchData():void
{
var token:AsyncToken= syncmanager.fetch("fetch");
token.addResponder(new mx.rpc.Responder(fetchSuccess, fetchFault));

               // Test For SessionToken class to be Dynamic, so that Dynamic Properties could
be added
                 token.userdefined_key = "value";
         }
        public function fetchSuccess(event:SyncResultEvent):void
            {

              if(event.token.userdefined_key == "value")
                { .... }
             }
```

Support for AutoCommit

SyncManager supports a Boolean property `autoCommit`.

The default value is `false`.

If `true`, the changes in the local database are committed to the server when the `save`, `saveUpdate`, and `remove` methods are used as shown here:

```
private var syncmanager:SyncManager = new SyncManager();
syncmanager.autoCommit = true;
```

This functionality helps you minimize the conflicts during the synchronization with the server, particularly in the case of auto-generation of primary key on client and serverside.

New attribute for SessionResultEvent and SessionToken

The classes `SessionResultEvent` and `SessionToken` have a new attribute `autoGeneratedId` that gets populated with the auto-generated ID used by ActionScript ORM. `autoGeneratedId` is populated only when a key is generated by the ActionScript ORM in that specific call.

Example

```
private function connectSuccess(event:SessionResultEvent):void
{
  // Generate an Order Obect
    .
    .
    .
    .
  // Save the Order
  var savetoken:SessionToken = session.save(ord);
  savetoken.addResponder(new mx.rpc.Responder(savesuccess, savefault));
}
private function savesuccess(event:SessionResultEvent):void
{
   // This is how, you can access autogenrated PK
   RememberINTPK = event.autoGeneratedId.toString();
   var loadtoken:SessionToken = session.loadByPK(Order,{oid:RememberINTPK},true);
   loadtoken.addResponder(new mx.rpc.Responder(loadsuccess,loadfailure))
}
```

Note: *Assume that the server database generates primary keys and you choose to generate primary key on client SQL Lite table (as shown in the example). This scenario results in a conflict which the application developer must resolve. An option is to design your application in such a way that you minimize conflicts between client and server objects. In this case, you can set client object primary keys as null or empty string before saving data to the database server using serverside ORM EntitySave function.*

Flash Remoting enhancements

A channel-definition construct has been introduced in services-config.xml (*CF_root*/wwroot/WEB-INF/flex/) named `serialize-array-to-arraycollection`.

This construct provides more flexibility and control for users to decide whether to serialize the ColdFusion array to ActionScript Array or ArrayCollection.

To serialize, in the XML, set the value to `true` for the following:

```
<serialize-array-to-arraycollection>false</serialize-array-to-arraycollection>
```

Note: *This construct is not considered when ColdFusion Array is sent to LCDS Flex Client. In this case, ColdFusion Array always gets translated to ActionScript ArrayCollection.*

Support for BlazeDS 4 and LCDS 3

ColdFusion 9 Update 1 supports LCDS 3.

In ColdFusion 9, support was limited to LCDS 2.6.1.

Also, ColdFusion now supports BlazeDS 4.

New End Point classes for BlazeDS

Channel-definition ID	Endpoint Class
my-streaming-amf	coldfusion.flash.messaging.CFStreamingAMFEndPoint
secure-streaming-amf	coldfusion.flash.messaging.SecureCFStreamingAMFEndPoint

New End Point classes for LCDS 3

Channel-definition ID	Endpoint Class
my-nio-amf	coldfusion.flash.messaging.CFNIOAMFEndPoint secure-nio-amf
secure-nio-amf	coldfusion.flash.messaging.SecureCFNIOAMFEndPoint
secure-cf-rtmp	coldfusion.flash.messaging.SecureCFRTMPEndPoint
my-nio-amf-stream	coldfusion.flash.messaging.CFStreamingNIOAMFEndPoint
secure-nio-amf-stream	coldfusion.flash.messaging.SecureCFStreamingNIOAMFEndPoint

LCDS 3 Integration Files

The following ZIP files are available for LCDS3 integration.

- **LCDS2.6.1_for_CF901.zip**: Use this ZIP file to integrate LCDS 2.6.1. The ZIP file packaged along with ColdFusion 9 does not work with ColdFusion 9 Update 1. Therefore, users who want to use LCDS 2.6.1 with ColdFusion 9 Update 1 must use this ZIP file.
- **LCDS3.x_for_CF901.zip:** Use this ZIP file to integrate LCDS 3.

A readme.txt is available inside the ZIP files. Follow the instructions in it for assistance.

New methods in ColdFusion Messaging Gateway CFCs

The following new methods have been introduced in ColdFusion Messaging Gateway CFCs:

- `allowSend`
- `allowSubscribe`

  Both the methods take `subtopic` as the parameter.

  These methods help you place control over subscribing and sending data to a particular subtopic.

Note: *To call these methods on their gateway CFC, specify the gateway id under the messaging destination in messaging-config.xml (Web_INF/Flex). By default, the value is* `*`*.*

Solr enhancements

Apart from overall improvement in the accuracy of indexing, the following are the enhancements:

- Displays correct MIME types for all documents
- Enhanced support for indexing of metadata for binary files such as mp3 and JPEG
- Support for the attribute `previousCriteria` (in the tag `cfsearch`)
- Both the tags `cfindex` and `cfsearch` support the attribute `categoryTree`.
- A new section to enable/disable term highlighting for entire document has been added to the ColdFusion Administrator. This applies only if Solr is running on local machine.

  Note: *The following steps apply only if Solr is running on local machine. If Solr is on a remote machine, follow the steps provided in the section Term highlighting in Developing ColdFusion 9 Applications.*

  1. In the ColdFusion Administrator, go to ColdFusion Collections and then click a Solr collection.
  2. Use the Enable/Disable button to enable/disable term highlighting.
  3. Reindex the collection.

  Note: *Enabling term highlighting increases the size of the Solr collection. So ensure that you allocate adequate memory for Solr if you are enabling term highlighting.*

Upgrading Solr

Solr is upgraded as part of updater.

Therefore, to use Solr with ColdFusion 9 Update 1, upgrade Solr.

For local installation, Solr is auto-upgraded when you run the ColdFusion Updater.

For remote installation, manually upgrade Solr using the following steps:

1. Stop Solr.
2. Back up solr.xml available in *Solr_Home*/multicore.
3. Uninstall Solr.
4. Reinstall the standalone version of Solr available on Adobe download location.
5. Stop Solr (if it has started automatically).
6. Bring back the backed up copy of solr.xml to *Solr_Home*/multicore.

Note: *After you upgrade, ensure that you reindex the entire Solr collection before you use the search service.*

Logging enhancements

Log files

You can generate log files for the following services:

- http
- ftp
- web service

- Portlet
- Derby
- Feed

Enable/Disable logging

A new icon has been added in the Actions column of the Log Files page (ColdFusion Administrator > Debugging & Logging). This icon lets you stop/start logging for a particular log type.

Server monitoring enhancements

Enhancements in this release help you use Server Monitoring effectively in load conditions. In ColdFusion 9 and older releases, Server Monitoring used to be unresponsive when the server load is high. To address this issue, a new monitoring server has been introduced.

Also, the ColdFusion Administrator has the following monitoring options (ColdFusion Administrator > Server Monitoring > Monitoring Settings):

- Enable monitoring
- Enable profiling
- Enable memory tracking

Configuring the Server monitoring settings

The monitoring server can be configured in one of the following ways:

- Use ColdFusion Administrator
- Manually edit neo-monitoring.xml and jetty.xml
- Use Admin API (servermonitoring.cfc)

Using the ColdFusion Administrator

The Server Monitoring Settings Page in the ColdFusion Administrator (Server Monitoring > Monitoring Settings) lets the following configurations:

- Enable monitoring server.

  Note: *When you enable monitoring server and configure it to use SSL, include the following setting to java.args in the JVM.config file:* `Dcoldfusion.jsafe=true`

- Specify the port on which monitoring server listens. The default port is 5500

Note: *If a server monitoring application is already running, the configuration mentioned here takes effect only after you relaunch the application.*

Manually editing neo-monitoring.xml and jetty.xml

neo-monitoring.xml

Go to the following location:

cf_root\lib (in the server configuration)

or

cf_root/WEB-INF/cfusion/lib (in the J2EE configuration).

Modify the value to `true` in the following code:

```
<var name='ismonitoringserverenabled'><boolean value='false'/></var>
```

Jetty.xml

Modify Jetty.xml only if you have to change the port or if your connection uses HTTPS protocol.

Go to the following location:

cf_root\lib (in the server configuration)

or

cf_root/WEB-INF/cfusion/lib (in the J2EE configuration).

You can specify the following configurations in the XML file:

- Port
- MaxThreads
- Logging

For connections using HTTPS protocol

1. Open jetty.xml.
2. Remove or comment out the Set Connectors section:

```
<Call name="addConnector">
      <Arg>
          <New class="org.eclipse.jetty.server.nio.SelectChannelConnector">
            <Set name="host">0.0.0.0</Set>
            <Set name="port">5500</Set>
            <Set name="maxIdleTime">300000</Set>
            <Set name="Acceptors">2</Set>
            <Set name="statsOn">false</Set>
            <Set name="lowResourcesConnections">10</Set>
        <Set name="lowResourcesMaxIdleTime">5000</Set>
            </New>
      </Arg>
```

3. Uncomment the Set SSL Connector section:

```
<Call name="addConnector">
      <Arg>
        <New class="org.eclipse.jetty.server.ssl.SslSelectChannelConnector">
            <Set name="host">0.0.0.0</Set>
            <Set name="port">5500</Set>
            <Set name="maxIdleTime">300000</Set>
            <Set name="Acceptors">1</Set>
            <Set name="AcceptQueueSize">100</Set>
            <Set name="Keystore">"path to keystore"</Set>
            <Set name="Password">OBF:1vny1zlo1x8e1vnw1vn61x8g1zlu1vn4</Set>
            <Set name="KeyPassword">OBF:1u2u1wml1z7s1z7a1wnl1u2g</Set>
            <Set name="truststore">"path to keystore"</Set>
            <Set name="trustPassword">OBF:1vny1zlo1x8e1vnw1vn61x8g1zlu1vn4</Set>
        </New>
      </Arg>
    </Call>
```

4 Specify the port and the keystore-related settings.

Using Admin APIs

To programmatically configure the Server Monitoring server, use the ServerMonitoring.cfc.

The following Administrator APIs are added in this release:

API	Description
setMonitoringServerPort(port);	Sets the port information for the monitoring server.
getMonitoringServerPort();	Gets details of the port to which the monitoring server listens.
getMonitoringServerProtocol();	Gets the protocol details for the monitoring server.
enableMonitoringServer();	Enables the monitoring server and starts it if not running.
stopMonitoringServer();	Stops the monitoring server
startMonitoringServer();	Starts the monitoring server
disableMonitoringServer();	Disables the monitoring server and stops it if it is running
isMonitoringServerEnabled();	Indicates if the monitoring server is enabled
isMonitoringServerRunning();	Indicates if the monitoring server is running
configureMonitoringServer(flag, port);	Enables monitoring server and sets port information

Troubleshooting scenarios

Multi-server monitoring

For multi-server monitoring, ensure that you specify the cross domain details in the crossdomain.xml in (*CFRoot*/MonitoringServer).

Someone changes port in XML

The exception does not appear in the ColdFusion Administrator. You verify the log.

Monitoring with SSL

You might encounter an error while starting Monitoring Server in SSL mode.

To resolve this known issue, add the following in the jvm.config:

```
"-Dcoldfusion.disablejsafe=true"
```

Updating the threadpool

You can update the threadpool in the jetty.xml.

Modify the threadpool in the Server Threadpool section of the XML file:

```
<Set name="ThreadPool">
      <!-- Default queued blocking threadpool
      -->
      <New class="org.eclipse.jetty.util.thread.QueuedThreadPool">
        <Set name="minThreads">2</Set>
        <Set name="maxThreads">50</Set>
      </New>
      <!-- Optional Java 5 bounded threadpool with job queue
      <New class="org.eclipse.thread.concurrent.ThreadPool">
        <Set name="corePoolSize">50</Set>
        <Set name="maximumPoolSize">50</Set>
      </New>
      -->
    </Set>
```

Configurable seed for password encryption

In ColdFusion 9 Update 1, Administrator has option to specify a new seed value to encrypt data source passwords. Previously, ColdFusion used to assign a default seed value to encrypt data source passwords, but modification was not allowed.

To modify the default seed value assigned by ColdFusion or to change the value you specified,

1. In the ColdFusion Administrator, got to Security > Administrator and then in the Password Seed section, specify the new seed value between 8-500 characters.
2. Click Submit Changes.

Note: *When you modify the seed value, all data source connections are reset. Therefore, Adobe recommends that you perform this task when the server is idle or at the initial phase (after installation).*

Other enhancements

Data source username and passwords

Application.cfc lets you specify data source authentication details in the data source. The data source settings can now be a string or a struct. When it is a string, it is considered to be the data source name and authentication information is taken from the data source defined in the ColdFusion Administrator.

You can specify the authentication information using a struct value for data source. The following are the key names:

- `name`: data source name
- `username`: Username for the data source
- `password`: Password for the data source

Example

```
<this.datasource={name='cfartgallery', username="user", password="passwd"}>
```

or

```
<this.datasource="cfartgallery">
```

Note: *The same convention is used for ORM default data source where you can specify the data source authentication information in the ormsettings.*

Support for HQL in cfquery

The tag `cfquery` allows HQL queries. For HQL, specify `dbtype="hql"` in the `cfquery` tag.

Example

```
<cfquery dbtype="hql" name="artists" ormoptions=#{cachename=""}#>
from Artists where firstname=<cfqueryparam value="Aiden">
</cfquery>
```

New actions for cfpdf AIR Proxy

The ActionScript proxy class for PDF service coldfusion.service.mxml.pdf has the following new attributes:

- `extracttext`
- `extractimage`

Scheduling enhancement

This release supports automatic logging of scheduled tasks.

Third-party services and technologies

ColdFusion 9 Upgrade 1 supports the following services:

- Microsoft .NET Framework 4

  Note: *If you install ColdFusion 9 update, integration with .NET Service might not work. Therefore, you must reinstall .NET extension using the standalone .NET installer available in the Adobe download location.*

- Ehcache 2.0
- Hibernate 3.5.2
- ExtJS 3.1
- Solr 1.4
- DataDirect Connect for JDBC 4.1
- MySQL 5.1.11

Security-related changes

The following security-related specifications apply when you upgrade to ColdFusion 9 Upgrade 1:

- CFID, CFTOKEN, and jsessionid are marked `httpOnly`. This reduces the chance of session information being compromised on Cross Site Scripting (XSS) attack.
- Set the following system property for the session cookies to be httpOnly:
  `Dcoldfusion.sessioncookie.httponly=true`
- The support for session cookies to be `httpOnly` depends on the application server you use:
  - For Tomcat/JBoss, `httpOnly` is not supported for JSESSIONID
  - On JRun, add the system property in jvm.config file
  - For other application servers, see the relevant documentation for details on `httpOnly` support for session cookies.