ADOBE® COLDFUSION (2016 release)
## Contents

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>New and changed functions/tags in Adobe ColdFusion (2016 release)</td>
<td>1</td>
</tr>
<tr>
<td>Command Line Interface (CLI)</td>
<td>3</td>
</tr>
<tr>
<td>Generating Swagger documents</td>
<td>5</td>
</tr>
<tr>
<td>External Session Storage</td>
<td>16</td>
</tr>
<tr>
<td>NTLM support</td>
<td>19</td>
</tr>
<tr>
<td>Security enhancements</td>
<td>22</td>
</tr>
</tbody>
</table>
New and changed functions/tags in Adobe ColdFusion (2016 release)

This document lists new and changed functions/tags in Adobe ColdFusion (2016 release).

New functions
The following is a list of new functions in Adobe ColdFusion (2016 release):

ArrayContainsNoCase
ArrayDeleteNoCase
BooleanFormat
Floor
IsPDFArchive
QuerySort
QueryEach
QueryFilter
QueryKeyExists
QueryMap
QueryReduce
ReplaceListNoCase
SpreadsheetGetColumnCount
ValueArray

SpreadsheetAddRows
There is a new parameter, includeColumnNames, in this function. For more information, see SpreadsheetAddRows function description and examples.

ReplaceList
There is a new parameter, includeEmptyFields, in this function. For more information, see ReplaceList function description and examples.
**cfapplication**
There are two new attributes in the `<cfapplication>` tag:

1. `passArrayByReference`
2. `searchImplicitScopes`

For more information, see `cfapplication`.

**cfsearch**
Theverity types, simple, explicit, internet, internet_basic, and natural no longer exist. There are two new verity types, Standard and DisMax, in the tag. For more information, see `cfsearch`.

**cfloop**
There is a new attribute, `item`, in `<cfloop>`. The attribute, `index`, is now optional.

For more information, see `cfloop`.

**cfmailparam**
There is a new, optional attribute, `filename`, in `cfmailparam`.

For more information, see `cfmailparam`.

**cfoutput**
There is a new, optional attribute, `encodefor` in this tag.

For more information, see `cfoutput`.

**cfpdf**
In Adobe ColdFusion (2016 release), you can use two additional 256-bit encryption algorithms when protecting a PDF document. The algorithms are:

1. AES_256R5
2. AES_256R6

For more information, see `cfpdf`.

**CacheRemove**
The `CacheRemove` function has an updated syntax:

`CacheRemove(Object id, boolean throwOnError, String key, boolean exact)`

For more information, see `CacheRemove`.

**TimeFormat**
The `TimeFormat` function has new masks to represent time zones in different formats. For more information, see `TimeFormat` function description and examples.
**DateFormat**
The DateFormat function has new masks to represent time zones in different formats. For more information, see `DateFormat` function description and examples.

**Replace**
The Replace function has an updated description and syntax. This function can take either string or callback function as an argument instead of the argument to replace the string. For more information, see `Replace` function description.

**StructNew**
The function has a new parameter, structType, that represents the type of struct to be created.
For more information, see `StructNew`.

**WriteOutput**
The function, WriteOutput, has a new parameter, encodefor. encodefor applies encoding on the input string.
For more information, see `WriteOutput`.

**Other changes**
All services in the services folder work as expected only if you create a virtual directory. Create a virtual directory that points to the path of the services folder (/CFIDE/services/).
When accessing the ColdFusion services via the connector port, replace “/CFIDE/services/” in the WSDL url with the name of the virtual directory. For example, http://localhost:82/cfideserv/image.cfc?wsdl, where cfideserv is the name of the virtual directory.

**Command Line Interface (CLI)**

**Overview**
In Adobe ColdFusion (2016 release), there is a new Command Line Interface (CLI) for developers to run cfm scripts without starting the ColdFusion server.
You can use cf.bat to run cfm scripts from CLI. The cfm files can either be in wwwroot or a different location.
ColdFusion developers can write scripts with the following:

- File operations for reporting, logging, archiving, and so on.
- Database operations for monitoring, debugging, and so on.
- Network operations like mailing an error log or thread dump to a system admin.

**Execution**
Developers can also pass parameters from command line to the cfm script that is being executed.
Path
The path to the CFM can be either absolute or relative. Giving an absolute path sets the directory of the cfm as wwwroot. Giving a relative path sets the current working directory as wwwroot.

Arguments
You can pass positional and named arguments to the executing cfm from command line.

CLI has the following methods to read the arguments.

- cli.getArgs() - gets all the arguments
- cli.getNamedArgs() - gets all the named arguments.
- cli.getUnnamedArgs() - returns all the unnamed args as a CFML array or empty array if none are specified.
- cli.getArg(int index) - gets the argument at the index location.
- cli.getNamedArg(String argName) - gets the value of the named argument with name argName

Example

cf.bat test.cfm 10 20 foo=bar
cli.getArg(1) returns 10
cli.getArg(2) returns 20 cli.getNamedArg("foo") returns bar

Custom directories
In CLI, you can set outputdir and logdir while executing the cfm. By default, both classes and logs go to temp folder.

Usage:

cf.bat cliscript.cfm –outputdir=c:\cfclasses –logdir=c:\logs

Application.cfc
Lookup for Application.cfc is up to the wwwroot, which is set according to the path of cfm (absolute or relative). In Application.cfc, only onApplicationStart(), onApplicationStop(), and onError() methods are supported. There is no support for session and request methods in CLI.

Scopes
CLI supports the following scopes:

- application
- argument
- request
- this

Reading/Writing in Command Line
Also, CLI supports three more methods to read and write to stdin and stdout/stderr respectively:

- cli.read() - reads one line from stdin
- cli.writeln(message) - writes the message string to stdout
- cli.writeError(errorMessage) - writes the error message string to stderr
Example
readwrite.cfm
<cfset CLI.writeError("This is an error message from CLI writeError!")>
<cfset CLI.writeln("This is CLI write method!")>

Usage
cf.bat readwrite.cfm >> c:\logFile.txt 2>> c:\errFile.txt

Other supported features
CLI supports the following features:
• Mail
• Webservice
• Application datasources

Other functions
• cli.exit(exitCode) - Exit takes a status code and exits to the command prompt with the specified exit code.

Unsupported features
The following features are not supported with CLI:
• Charting
• Scheduled Tasks
• PDF features
• Document
• REST
• Solr
• Flex Integration
• DotNet Integration
• WebSocket
• Image Functions
• API Manager

Generating Swagger documents

Overview
Swagger is a project specification that is used to describe and document RESTful APIs. In ColdFusion 12, you can create swagger doc automatically from REST CFC after it is implemented and registered in server. The Swagger version that is supported in ColdFusion is 1.2.
For more information on Swagger project overview, see the Swagger documentation. Swagger specification 1.2 is available here.

**Document generation process**

The Swagger doc generation feature is a part of ColdFusion Server. ColdFusion server generates the Swagger doc automatically once you register REST CFC application.

**Creating a REST CFC file**

You can create REST CFC application file of your choice and place this file in the root folder (wwwroot) of ColdFusion server. A sample CFC file content structure is shown in the following studentservice.cfc file.

```cfc
<cfcomponent rest="true" restpath="studentService">
  <cffunction name="addStudent" access="remote" returntype="void" httpmethod="PUT" description="add student">
    <cfargument name="name" type="string" required="yes" restargsource="Form"/>
    <cfargument name="age" type="numeric" required="yes" restargsource="Form"/>
    <!--- Adding the student to data base. --->
  </cffunction>
  <cffunction name="addStudents" access="remote" returntype="void" httpmethod="POST" description="add students">
    <cfargument name="name" type="student[]" required="yes" restargsource="body"/>
    <!--- Adding the student to data base. --->
  </cffunction>
  <cffunction name="deleteStudent1" access="remote" returntype="void" httpmethod="DELETE" description="delete students">
    <cfargument name="students" type="student[]" required="yes" restargsource="Body"/>
    <!--- Adding the student to data base. --->
  </cffunction>
  <cffunction name="updateStudentAddress" access="remote" returntype="address" httpmethod="POST" restpath="{studentId}" description="modify student address" hint="modify the address for given studentId">
    <cfargument name="studentId" type="numeric" required="yes" restargsource="PATH"/>
    <cfargument name="address" type="address" required="yes" restargsource="Body"/>
    <!--- Adding the student to data base. --->
  </cffunction>
  <cffunction name="getStudent" access="remote" returntype="Student" restpath="{name}-{age}" httpmethod="GET" description="retrieve student" produces="application/json" responseMessages="404:Not found,200:Successfull:student" >
    <cfargument name="name" type="string" required="yes" restargsource="Path"/>
    <cfargument name="age" type="string" required="yes" restargsource="Path"/>
    <!--- Create a student object and return the object. This object will handle the request now. --->
    <cfset myobj = CreateObject("component", "Student")>
    <cfset myobj.name = name>
    <cfset myobj.age = age>
    <cfreturn myobj>
  </cffunction>
</cfcomponent>
```

**Using application.cfc file**

If you do not want the ColdFusion server to generate swagger doc automatically, set the following code to false in application.cfc file. Refresh the registered application in CF administrator.

```cfc
<cfset this.restsettings.generateRestDoc="false">
```
A sample application.cfc file is shown below for your reference.

```cfcomponent>
  <cfset this.name="info" />
  <cfset this.sessionmanagement = true />
  <cfset this.restsettings.generateRestDoc="true">
  <cfset this.restsettings.restDocInfo.title="this is title">
  <cfset this.restsettings.restDocInfo.apiVersion="2.0">
  <cfset this.restsettings.restDocInfo.description="this is description">
  <cfset this.restsettings.restDocInfo.termOfServiceUrl="url is here">
  <cfset this.restsettings.restDocInfo.contact="xyz@adobe.com">
  <cfset this.restsettings.restDocInfo.license="adobe 1.0">
</cfcomponent>

**Using new responseMessages attribute**

A new attribute called responseMessages has been introduced in ColdFusion 12 release. You can use this attribute in REST CFC file as shown in the sample file below.

```cfcomponent rest="true" restPath="/cookieService" produces="text/plain" >
  <!--- Test with various produces --->
  <cffunction name="sayPlainHelloUser" responseMessages="404:Not Found,200:successful,10:notdefine" access="remote" returnType="String" httpMethod="GET" produces="text/plain">
    <cfargument name="nAme" type="string" restargsource="cOOkie" required="false" default="CF">
    <cfset res="Hello " &amp; name>
    <cfreturn res>
  </cffunction>
</cfcomponent>
```

The swagger API document generated from this sample responseMessages code appears as shown below.

```
"responseMessages": [
  {
    "code":404,
    "message":"Not Found"
  },
  {
    "code":200,
    "message":"successful"
  }
]
```

**Registering your CFC application**

Start ColdFusion Administrator. Click Data & Services > REST Services on the left pane and add configuration values according to the instructions in the following dialog.
1 Enter the root path where REST CFCs are available in your system. Alternatively, you can click **Browse Server** and choose the path where the CFC application resides.

2 Enter the host name for the REST service. For example, localhost:8500.

3 Enter the **Service Mapping** string name. For example, http://localhost/rest/{service mapping}/test.

4 Select the check-box if you want to set the application as default while calling the web service.

### Accessing swagger api-docs

The Swagger representation of the API contains two file types:

1 **The Resource Listing** - This is the root document that contains general API information and lists the resources. Each resource has its own URL that defines the API operations on it.

2 **The API Declaration** - This document describes a resource, including its API calls and models.

You can verify the ColdFusion generated swagger APIs document by using the ColdFusion server path as follows:

```
<ColdFusion server URL path:port number>/<Service Mapping name>/api-docs/<resourcePath name>
```

You can access resource listing by using the same path as above without the resourcePath name. (``<ColdFusion server URL path:port number>/Service Mapping name>/api-docs``)

Service Mapping name is the name that you specify while registering your REST application in ColdFusion server.

For example, localhost:8500/test/api-docs/studentService

The swagger API document generated from the sample studentservice.cfc REST CFC file appears as shown in the following api document:
{
  "swaggerVersion":"1.2",
  "apiVersion":"1.0",
  "basePath":"localhost:8500/rest/test",
  "resourcePath":"/studentService",
  "apis": [
    { 
      "path":"/studentService/",
      "description": "",
      "operations": [ 
        { 
          "nickname": "addStudents",
          "method": "POST",
          "summary": "add students",
          "type": "void",
          "parameters": [ 
            { 
              "name": "name",
              "paramType": "body",
              "allowMultiple": false,
              "required": true,
              "type": "array",
              "items": {
                "$ref": "student"
              }
            }
          ]
        },
        { 
          "nickname": "deleteStudent1",
          "method": "DELETE",
          "summary": "delete students",
          "type": "void",
          "parameters": [ 
            { 
              "name": "students",
              "paramType": "body",
              "allowMultiple": false,
              "required": true,
              "type": "array",
              "items": {
                "$ref": "student"
              }
            }
          ]
        },
        { 
          "nickname": "addStudent",
          "method": "PUT",
          "summary": "add student",
          "type": "void",
          "parameters": [ 
            { 
              "name": "name",
              "paramType": "form",
              "allowMultiple": false,
              "required": true,
              "type": "string",
              "items": { }}: $$ref": "student"
            }
          ]
        }
      ]
    }
  ]
}
"type":"string"
},

"name":"age",
"paramType":"form",
"allowMultiple":false,
"required":true,
"type":"number"
}]
]

"path":"/studentService/{name}-{age}",
"description":",
"operations":[
{
"nickname":"getStudent",
"method":"GET",
"summary":"retrieve student",
"type":"student",
"produces":[
  "application/json"
],
"parameters":[
{
  "name":"name",
  "paramType":"path",
  "allowMultiple":false,
  "required":true,
  "type":"string"
},

{
  "name":"age",
  "paramType":"path",
  "allowMultiple":false,
  "required":true,
  "type":"string"
}
],
"responseMessages":[
{
  "code":404,
  "message":"Not found"
},
{
  "code":200,
  "message":"Successfull",
  "responseModel":"student"
}
]
}
},

"path":"/studentService/{studentId}"
"description": "", "operations": [ { "nickname": "updateStudentAddress", "method": "POST", "summary": "modify student address", "notes": "modify the address for given studentId", "type": "address", "parameters": [ { "name": "studentId", "paramType": "path", "allowMultiple": false, "required": true, "type": "number" }, { "name": "address", "paramType": "body", "allowMultiple": false, "required": true, "type": "address" } ] } ], "models": { "address": { "id": "address", "description": "this is a address component", "required": [ ], "properties": { "country": { "type": "string" }, "street": { "type": "string" }, "houseNo": { "type": "number", "format": "double" }, "state": { "type": "string" } } }, "student": { "id": "student", "description": "this is a student component", "required": [ "address", "name", "age" ] } }
CFC and Swagger mapping structure
You can compare CFC field types and Swagger field types from the following mapping structures.

Resource listing schema
The Resource Listing serves as the root document for the API description. It contains general information about the API and an inventory of the available resources.
API declaration schema

The API declaration provides information about an API exposed on a resource. You should have only one file described per resource. The file MUST be served in the URL described by the path field.

<table>
<thead>
<tr>
<th>Swagger doc Field Name</th>
<th>Type</th>
<th>Description</th>
<th>CF Field</th>
</tr>
</thead>
<tbody>
<tr>
<td>SwaggerVersion</td>
<td>String</td>
<td>Required. Specifies the Swagger Specification version being used.</td>
<td>Update programmatically using API Manager</td>
</tr>
<tr>
<td>apiVersion</td>
<td>string</td>
<td>Provides the version of the application API</td>
<td>Modify using application.cfc file</td>
</tr>
<tr>
<td>info</td>
<td>Info Object</td>
<td>Provides metadata about the API. The metadata can be used by the clients if needed, and can be presented in the Swagger-UI for convenience.</td>
<td>Modify using application.cfc file</td>
</tr>
<tr>
<td>authorizations</td>
<td>Authorizations Object</td>
<td>Provides information about the authorization schemes allowed on this API. The type of the authorization scheme. Values MUST be either &quot;basicAuth&quot;, &quot;apiKey&quot; or &quot;oauth2&quot;.</td>
<td>Update programmatically using API Manager</td>
</tr>
<tr>
<td>basePath</td>
<td>string</td>
<td>Required. The root URL serving the API.</td>
<td>Add programmatically while parsing CFC</td>
</tr>
<tr>
<td>consumes</td>
<td>[string]</td>
<td>A list of MIME types the APIs on this resource can consume. This is global to all APIs but can be overridden on specific API calls.</td>
<td>Cfcomponent.consumes</td>
</tr>
<tr>
<td>produces</td>
<td>[string]</td>
<td>A list of MIME types the APIs on this resource can produce. This is global to all APIs but can be overridden on specific API calls.</td>
<td>Cfcomponent.produces</td>
</tr>
<tr>
<td>resourcePath</td>
<td>string</td>
<td>The relative path to the resource, from the basePath, which this API Specification describes.</td>
<td>Cfcomponent.restpath</td>
</tr>
<tr>
<td>apis</td>
<td>[API Object]</td>
<td>Required. A list of the APIs exposed on this resource. There MUST NOT be more than one API Object per path in the array.</td>
<td>Details in API Object</td>
</tr>
<tr>
<td>apiVersion</td>
<td>string</td>
<td>Provides the version of the application API (not to be confused by the specification version).</td>
<td>N/A</td>
</tr>
</tbody>
</table>
### API Object Schema

The API Object describes one or more operations on a single path. In the `apis` array, there MUST be only one API Object per path.

<table>
<thead>
<tr>
<th>Swagger doc Field Name</th>
<th>Type</th>
<th>Description</th>
<th>CF Field</th>
</tr>
</thead>
<tbody>
<tr>
<td>description</td>
<td>String</td>
<td>A short description of the resource.</td>
<td>Cffunction.description</td>
</tr>
<tr>
<td>operations</td>
<td>[Operation Object]</td>
<td>Required. A list of the API operations available on this path. The array may include 0 or more operations.</td>
<td>Details in Operation Object</td>
</tr>
<tr>
<td>Path</td>
<td>String</td>
<td>Required. The relative path to the operation, from the basePath, which this operation describes. The value SHOULD be in a relative (URL) path format.</td>
<td>Component.restpath + Cffunction.restpath</td>
</tr>
</tbody>
</table>

### Operation Object Schema

The Operation Object describes a single operation on a path. In the `operations` array, there must be only one Operation Object per method. This object includes the Data Type Fields in order to describe the return value of the operation. The `type` field must be used to link to other models.

This is the only object where the type may have the value of `void` to indicate that the operation returns no value.

<table>
<thead>
<tr>
<th>Swagger doc Field Name</th>
<th>Type</th>
<th>Description</th>
<th>CF Field</th>
</tr>
</thead>
<tbody>
<tr>
<td>authorization</td>
<td>Authorizations Object</td>
<td>A list of authorizations required to execute this operation</td>
<td>Programmatically from API Manager</td>
</tr>
<tr>
<td>consumes</td>
<td>[string]</td>
<td>A list of MIME types this operation can consume.</td>
<td>Cffunction.consumes</td>
</tr>
</tbody>
</table>
The Parameter Object describes a single parameter to be sent in an operation and maps to the parameters field in the Operation Object. This object includes the Data Type Fields in order to describe the type of this parameter. The type field must be used to link to other models.

If type is File, the consumes field must "multipart/form-data", and the paramType must be "form".

### Parameter object schema

<table>
<thead>
<tr>
<th>Swagger doc Field Name</th>
<th>Type</th>
<th>Description</th>
<th>CF Field</th>
</tr>
</thead>
<tbody>
<tr>
<td>allowMultiple</td>
<td>boolean</td>
<td>Another way to allow multiple values for a &quot;query&quot;, &quot;header&quot; or &quot;path&quot; parameter.</td>
<td>Not available in ColdFusion.</td>
</tr>
<tr>
<td>description</td>
<td>string</td>
<td><strong>Recommended.</strong> A brief description of this parameter.</td>
<td>Cfargument.hint</td>
</tr>
</tbody>
</table>
### Overview

The scaling of any web application depends on how user sessions are handled. One way to address scaling is to implement sticky sessions and session replication through node clusters.

But sticky sessions have their own bottlenecks. In such situations, you can implement an external cache server to handle various sessions across multiple nodes.

---

---

### CFC/Swagger/Java types comparison

<table>
<thead>
<tr>
<th>CFC</th>
<th>Swagger</th>
<th>Java</th>
<th>Additional information</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>string</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>paramType</td>
<td>string</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>required</td>
<td>boolean</td>
<td>boolean</td>
<td></td>
</tr>
</tbody>
</table>

- **name**: string
- **paramType**: string
- **required**: boolean
- **Additional information**: For argument map to "body"

---

---

**Note**: As per spec swagger doesn’t support “Cookie”, “Matrix” paramtype which we have in ColdFusion.
All ColdFusion sessions are stored in memory. But the sessions cannot be shared across ColdFusion nodes. You can replicate sessions, but as the number of nodes increase in a cluster, the configuration and management of sessions become increasingly difficult and resource-intensive.

One way to solve this problem is to use an external session storage like Redis.

**Redis support**

You can now store ColdFusion sessions in external cache servers like Redis, instead of in-memory storage in ColdFusion servers.

To set up session storage in Redis:

1. In ColdFusion administrator, click **Server Settings > Memory Variables**.
Select Redis from the Session Storage drop-down list.

Enter the Redis server details and the port number.

Note: If you select Use J2EE session variables, you cannot store ColdFusion sessions in Redis.

Session object serialization

Once a session object is received from Redis, there can be changes to the object within the timespan of a given request. The session is persisted (if modified) back to external storage on request end. The changes made by the current request on one node are available to all other nodes.

Note: If the nodes in a cluster use Redis, disable sticky sessions at the load balancer. In addition, clear the Sticky Sessions and Session Replication check-boxes in Cluster Manager.
Session invalidation

After a specified timeout period, sessions in both in-memory and Redis are invalidated and removed.

Backing up Redis

You can take a backup of Redis using rdb files. For example, you can create backups of rdb files every hour and save the snapshots to a different location.

The Redis server creates dump.rdb and you can copy this file to another location.

To retrieve Redis data, delete the file appendonly.aof and also delete the latest instance of dump.rdb. Copy the older dump.rdb file to the location.

**Note:** Before deleting the files, stop the Redis server. After copying the rdb file, restart Redis.

To modify the duration of snapshots of Redis, edit the redis.windows.conf.properties file and modify the snapshot properties.

NTLM support

NTLM support

There is support for NTLM (NT LAN Manager) for `<cfsharepoint>`, `<cfinvoke>`, and `<cfobject>` tags, and `CreateObject` function. NTLM is a suite of Microsoft security protocols that provides authentication, integrity, and confidentiality to users.

Unlike Basic Auth, NTLM is embedded in the application protocol and does not depend on the SSL (Secure Sockets Layer) to protect passwords during transmission. It saves the hash values of the password and discards the original password. Later the hashed value of the password is used to encrypt a challenge sent by the server to the client. The value is then used to authenticate the user.

If the authType attribute is NTLM, specify the ntlmdomain attribute.
<cfsharepoint> tag
There are new NTLM attributes in this tag.

```cfc
<cfsharepoint
  authType="NTLM"
  action="webservice action"
  params="parameter structure"
  domain="domain name"
  ntlmdomain="domain where user is registered"
  workstation="workstation name"
  name="result variable name"
  password="connection password"
  username="user ID"
  wsdl="WSDL file path">

or,

```cfc
<cfsharepoint
  action="webservice action"
  params="parameter structure"
  login = "credentials structure"
  name ="result variable name"
  wsdl="WSDL file path">
```

Where,

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>authType</td>
<td>The authentication type to use. You can use NTLM or BASIC.</td>
</tr>
<tr>
<td>ntlmdomain</td>
<td>Domain in which a user is registered.</td>
</tr>
<tr>
<td>workstation</td>
<td>Host name of the client machine.</td>
</tr>
</tbody>
</table>

Note: When the login attribute is used, then the NTLM attributes must go into the login struct. For example,

```cfc
<cfscript>
  loginStruct = {domain="myDomain", username="user name", password="pa$$w0rd", authtype="ntlm", ntlmdomain="myNTLMDomain"};
  cfssharepoint (action="getlistcollection", login=loginStruct, name="myResult");
  writeDump(myResult);
</cfscript>
```

<cfinvoke> tag
There are new NTLM attributes in this tag.
<cfinvoke
authType="NTLM"
webservice="Web service name or WSDL URL"
ntlmdomain="domain name"
workstation="workstation name"
method="operation name"
password="password"
proxyPassword="password for proxy server"
proxyPort="port on proxy server"
proxyServer="WSDL proxy server URL"
proxyUser="user ID for proxy server"
returnVariable="variable name"
refreshWSDL="yes\nno"
servicePort="WSDL port name"
timeout="request timeout in seconds"
username="username"
wsdl2javaargs="argument string">

Where,

<table>
<thead>
<tr>
<th>authType</th>
<th>The authentication type to use. You can use NTLM or BASIC.</th>
</tr>
</thead>
<tbody>
<tr>
<td>ntlmdomain</td>
<td>Host name of the domain controller.</td>
</tr>
<tr>
<td>workstation</td>
<td>Host name of the client machine.</td>
</tr>
</tbody>
</table>

<cfobject> tag
There are new NTLM attributes in this tag.

<cfobject
authType="NTLM"
ntlmdomain="domain name"
workstation="workstation name"
type="webservice"
name="ws"
webservice="WSDL URL"
username = "user name"
password = "password">

Where,

<table>
<thead>
<tr>
<th>authType</th>
<th>The authentication type to use. You can use NTLM or BASIC.</th>
</tr>
</thead>
<tbody>
<tr>
<td>ntlmdomain</td>
<td>Host name of the domain controller.</td>
</tr>
<tr>
<td>workstation</td>
<td>Host name of the client machine.</td>
</tr>
</tbody>
</table>

CreateObject function
This function includes support for NTLM.

For example,

```cftmpl
wsobj = createObject("webservice",
"http://localhost/AxisWs/services/Converter?wsdl",{refreshwsdl=true,username="user
name",password="password",
aauthtype="NTLM",workstation="workstation",ntlmdomain="ntlmdomain",wsversion=2});
```

Where,
Security enhancements

For any web application, security plays a critical role. It is important to avoid security pitfalls while developing web applications.

Security code analyzer

Security Analyzer is a new feature in Adobe ColdFusion (2016 release). This feature is integrated into ColdFusion Builder to enable developers to avoid common security pitfalls and vulnerabilities while writing ColdFusion code.

Use this feature to view:

- Vulnerable code in the editor
- Vulnerability or type of attack (Error and Warning)
- Severity level of vulnerability (High, Medium, and Low)
- Suggestion to avoid the vulnerability.

**Note:**

- Security Analyzer works only with ColdFusion Enterprise or Enterprise (trial) and with Developer profile.
- Security Analyzer does not work with ColdFusion Developer or Standard edition and with Production or Production secure profile.
- Security Analyzer does not work with ColdFusion Builder default local server.

The Security Analyzer feature of the server is exposed as a service, a request to which is made by the builder. You can get a list of security vulnerabilities for a file, folder, or a project.

**Accessing security analyzer in builder**

Follow the steps below to access Security Analyzer in ColdFusion Builder:

1. Right-click the project folder or project file in the navigator pane.

You have three options in Security Analyzer:

- **Run Security Analyzer** – Analyzes and displays vulnerabilities of the code.
- **Clean Run Security Analyzer** - Clears the history of all ignored messages and warnings. It clears the ignored vulnerabilities (which are marked as Ignore during the Run Security Analyzer) and displays all vulnerabilities for the project.

<table>
<thead>
<tr>
<th>authType</th>
<th>The authentication type to use. You can use NTLM or BASIC.</th>
</tr>
</thead>
<tbody>
<tr>
<td>ntlmdomain</td>
<td>Host name of the domain controller.</td>
</tr>
<tr>
<td>workstation</td>
<td>Host name of the client machine.</td>
</tr>
</tbody>
</table>

authType
dntlmdomain
workstation
• **Clear Security Markers** – Removes all security warnings and resources. Run the security analyzer again to view the vulnerabilities for your resource.

**Using the security analyzer**

Follow the steps below to use Security Analyzer for your project folder or file:

1. Create a ColdFusion project or use an existing project. Ensure that the project is aligned to the preferred server. You can verify it by choosing the appropriate server in project properties. (Right-click in the navigator and choose properties).

2. Right-click the project folder or project file and choose Security Analyzer > Run Security Analyzer. Security analyzer analyzes the code and displays a pop-up dialog when the task is completed.

3. Click OK. You can view all the vulnerabilities in the bottom pane of the Editor as shown below.

4. Click Security Issues on the left pane to view the list of vulnerabilities.
   a. As shown in the left pane of the snapshot, click the vulnerability type (such as SQL Injection or XSS attack) to view the corresponding problem statement. You can also view the suggested solution at the right pane.
   b. Alternatively, you can click any error on the middle pane to view the corresponding statement and solution at the right pane.
   c. Double-click each error on the middle pane to view the corresponding line in the Editor.
   d. Use filters for File Name, Attack Name, Severity Level, and Type in the middle pane. Start typing the file name in the search area to locate the files with vulnerabilities. You can narrow down your search based on severity level as high, medium and low by clicking All drop-down list.

   *Note:* You can notice the Both drop-down list as grayed out sometimes. This happens when your cursor is already pointing to Errors or warnings issue type in the left pane. You can bring it back to active state by selecting the Security Issues folder.

5. When you fix the error in the code, right-click the corresponding error on the middle pane and choose the status as Fixed. Mark the status as Ignore if you ignore the error.

   You can move the error back to To fix status by using the same step.

   *Note:* Rerunning Security analyzer (Security Analyzer > Run Security Analyzer) does not show the vulnerabilities that are ignored. If the user has marked the vulnerabilities as Fixed but are not fixed, then server reports these errors.

6. Click Export on the upper-right corner of the Security Analyzer pane to export all the vulnerabilities to a report.html file. You can view the graphical representation of all vulnerabilities for your resource in the exported file, as shown below.
Increasing the Security Analyzer timeout

You can increase the Security Analyzer timeout in the RDS Configuration settings. The default is 30 seconds.

1. Right click on the server or choose Windows > Show View > Other.
2. Type RDS in the text field.

Additional setup configurations

Perform one of the following:

- Open access to port 8500 using the Windows firewall.
- Set up a virtual directory for the site for /CFIDE in IIS and the urireworkermap.properties file for the given connector. In the file, remove the ! in front of /CFIDE/* = cffusion.

Workflow of Security Analyzer

1. Security Analyzer is exposed as a service by the ColdFusion Server.
2. By running Security Analyzer for a file or a set of files, the builder makes a request to this service.
3. The builder displays the vulnerabilities in a separate view for each file, along with corresponding line numbers.
4. You can double-click on the vulnerabilities and open the file in the editor window with cursor pointing at the corresponding line with a red icon.
5. Also, by single click, you get a brief description about the attack and about possible ways to avoid it.

Last updated 2/18/2016
List of security vulnerabilities

- **SQL Injection**

  As shown in the following sample code, the attacker can create arbitrary SQL statements to execute against the database by passing values into the url.id variable. For example, the attacker can pass a value of 1 `DELETE FROM news` to delete all news articles in the table or 0 `UNION SELECT username, password FROM users` to extract username and password values from the database.

  ```
  <cfquery>
  SELECT headline, story
  FROM news
  WHERE id = #url.id#
  </cfquery>
  ```

  **Vulnerable scenarios**

  - `<cfquery name="SelectExample" datasource="cfdocexamples">
    select FROM Employee
    WHERE Emp_ID=#var#
  </cfquery>`
  - `<cfset result = QueryExecute("select * from Employees where empid=#id5#")>
  - `<cfset v3="#form.vf#"><cfset employees = ORMExecuteQuery("from Employee where name=#v3#")>

  All the above code samples use an unknown variable inside the query statement, which makes them vulnerable.

- **XSS Attack**

  ```
  <cfoutput>Hello #url.name#</cfoutput>
  ```

  Using the above code, the attacker can pass JavaScript into the url.name variable to be executed in the browser of anyone visiting the URL. Attackers also try to post XSS code that can be stored in a database and execute later. For example, posting a comment to display for all visitors of a page.
Vulnerable scenarios

- `<cfoutput>Hello #name2#</cfoutput>
- `<cfparam name = "id12" default = "my default value" type="string">
- `<cfoutput>#id12#` When a variable declared through cfparam is of type “string”, it is vulnerable code.
- `<cfoutput> <b>LINK to URL:</b> <a target="_blank" href="http://#url#">#url#</a> </cfoutput> As an unknown variable is used for the url link in the anchor tag, it is vulnerable to XSS attack.

PDF XSS Attack

The cfhtmltopdf tag, introduced in ColdFusion 11 provides powerful HTML rendering, powered by WebKit to produce PDF files. As the server renders the HTML, be cautious while using variables in the PDF document.

All preventative measures pertaining to cross site scripting also apply to variables written in the cfhtmltopdf tag. JavaScript can be executed during rendering, in the cfhtmltopdf tag.

Because the JavaScript would be executed on the server during rendering, the risks are different from a client side cross site scripting attack. Some of the risks include denial of service and potential exploit for unknown vulnerabilities in Webkit. In addition, there is a risk of bypassing the network firewall as the server can be behind a firewall with network access to other systems.

Vulnerable scenarios

- `<cfhtmltopdf>
  <h1>Hello <cfoutput>#pf2#</cfoutput></h1>
</cfhtmltopdf>

- `<cfhtmltopdf>
  <h1>Hello <cfoutput>#url.name#</cfoutput></h1>
</cfhtmltopdf>

- `<cfdocument format="PDF">
  <cfoutput> #hello#
  </cfoutput>
  <cfdocumentitem type="header" >
  <cfoutput>#abc#</cfoutput>
  </cfdocumentitem>
</cfdocument>

CSRF Attack

Cross Site Request Forgeries (CSRF) vulnerabilities are exploited when an attacker can trick an authenticated user into clicking a URL, or by embedding a URL in a page requested by a user's authenticated browser.

Vulnerable scenarios

- `<cfform method="POST"> <cfinput type="submit" value="Make Administrator"/></cfform>
When CSRFGenerateToken() function is not used, the code is vulnerable.

- `<cfform method="POST"> <cfinput type="hidden" name="token" value="#CSRFGenerateToken()#" />
  <cfinput type="submit" value="Make Administrator" /></cfform>
When there is no corresponding CSRFVerifyToken() function for CSRFGenerateToken function, the code is vulnerable.

- `<cfset var2 = CSRFGenerateToken2("make-admin")>`
- `<cfform method="POST" action="/csrf/dummy.cfm">`
- `<cfinput type="hidden" name="token" value="#var2#"/>
- `<cfinput type="submit" value="Make Administrator"/>
- `</cfform`

When there is no corresponding CSRFVerifyToken() function for CSRFGenerateToken function in the action page that is specified.

- **CFLocation Validation**
  Avoid appending the session identifiers to the URL query string. End users email, or publish URLs without realizing their session identifier is in the url.

**Vulnerable scenarios**

- `<cflocation url="random.cfm" addtoken="true">`
  When the attribute “addtoken” is explicitly set to true.

- `<cflocation url="random.cfm">`
  When the “addtoken” is not specified, the default value of true is taken.

- `<cfset addtoken1 = "true">`
- `<cflocation url="random.cfm" addtoken="#addtoken1#">`
  When a variable is used by “addtoken” attribute, which is set to true.

- **Cookies Validations**
  Cookies can contain sensitive information.

**Vulnerable scenarios**

- `<cfcookie name="sample" value="random" httponly="false" secure="false">`
  When both “httponly” and “secure” attributes are set to false explicitly, the code is vulnerable.

- `<cfcookie name="sample" value="random" httponly="true" secure="false">`
  When either of “httponly” and “secure” attributes is set to false explicitly, the code is vulnerable.

- `<cfcookie name="sample" value="random">`
  When “httponly” and “secure” attributes are not set, default value of false is taken, making the code vulnerable.

- **Passwords**
  Do not store passwords in plain text.

**Vulnerable scenarios**

- `<cfcache action="get" timespan="#createTimeSpan(0,0,10,0)#" password="pwd">`
- `<cfset password = "abc">`

Last updated 2/18/2016
In all above scenarios, hardcoded passwords are used which makes the code vulnerable.

**File upload Validation**

Whenever files are uploaded to the server, take extreme caution to ensure that you have properly validated the file path and file type.

**Vulnerable scenarios**

- `<cffile action = "upload" fileField = "FileContents" destination = "c:\folder1\folder2" accept = "text/html" nameConflict = "MakeUnique" strict="false">
- `<cffile action="upload" filefield="photo" accept="image/gif,image/png,image/jpg" destination="#getTempDirectory()#" nameconflict="overwrite" strict="false">
- `<cffile action="uploadall" destination="#expandpath('./upload')#" accept="text/html" strict="false">

In the above scenarios, strict is explicitly set to false, hence making it vulnerable. Also when “getTempDirectory()” function is not used for destination, it throws a warning.

**Get vs Post**

Do not send sensitive information over a GET method.

**Vulnerable scenarios**

- `<cfform method="get" action="sayHello.cfm">
- `<cfform action="sayHello.cfm">

When method is explicitly set to “get”, the code is vulnerable.

- `<cfform method="get" action="sayHello.cfm">
- `<cfform action="sayHello.cfm">

When method is not set to any value, by default "get " method is used.

**File injection**

`<cfinclude template="views/#header#">
The above vulnerable sample code does not validate the value of the #header# variable before using it in a file path. An attacker can use the vulnerable code to read any file on the server that ColdFusion has access to. For example, by requesting ?header=../../server-config.txt the attacker can read a configuration file that is not meant to be public.

**Vulnerable scenarios**

- `<cfinclude template="constant/#somepath#">`
- `<cffile action="write" file="#filevar2#">`
- `<cfscript> myfile = DirectoryDelete(var); </cfscript>`

In all of the above scenarios, an unknown variable is used for file path or directory path, so they are vulnerable.