Adobe ColdFusion Help | What's new in ColdFusion Raijin release

Adobe Systems Incorporated

Version 2.0

15th Oct, 2015

Adobe ColdFusion Raijin Help What's new document

This is pre-release documentation and does not constitute final documentation relating to the product it is distributed with.

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

Adobe ColdFusion Enterprise Edition offers a single platform to rapidly build and deploy scalable, high-performing web and mobile applications. Leverage unique capabilities to develop, test, and debug mobile applications end to end. Generate high-quality PDF files and manipulate them easily.

Read on for a quick introduction to the new features and enhancements introduced in Adobe ColdFusion Raijin release.

Contents

What's New

NTLM Support

There is support for NTLM (NT LAN Manager) for <cfsharepoint>, <cfinvoke>, and <cfobject> tags. 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, which is then further used to authenticate the user.

If the *authType* attribute is NTLM, you must specify the *domain* attribute.

<cfsharepoint>

There are new NTLM attributes in this tag.

<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">

Where,

authType	The authentication type to use. You can use NTLM or BASIC.
ntlmdomain	Domain in which a user is registered.
workstation	Host name of the client machine.

<cfinvoke>

There are new NTLM attributes in this tag.

<cfinvoke
authType="NTLM"

webservice="Web service name or WSDL URL"
domain="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|no"
servicePort="WSDL port name"
timeout="request timeout in seconds"
username="username"
wsdl2javaargs="argument string">

Where,

authType	The authentication type to use. You can use NTLM or BASIC.
domain	Host name of the domain controller.
workstation	Host name of the client machine.

<cfobject>

There are new NTLM attributes in the tag:

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

Where,

authType	The authentication type to use. You can use NTLM or BASIC.
domain	Host name of the domain controller.
workstation	Host name of the client machine.

Swagger document generation

Overview

Swagger is a project specification that is used to describe and document RESTful APIs. In ColdFusion 12 release, 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, refer to 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.

Create your 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.

```
<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.

<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">
        <cfset this.restsettings.restDocInfo.licenseUrl="http://abc.com">
</cfcomponent>
```

Using new response messages attribute

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

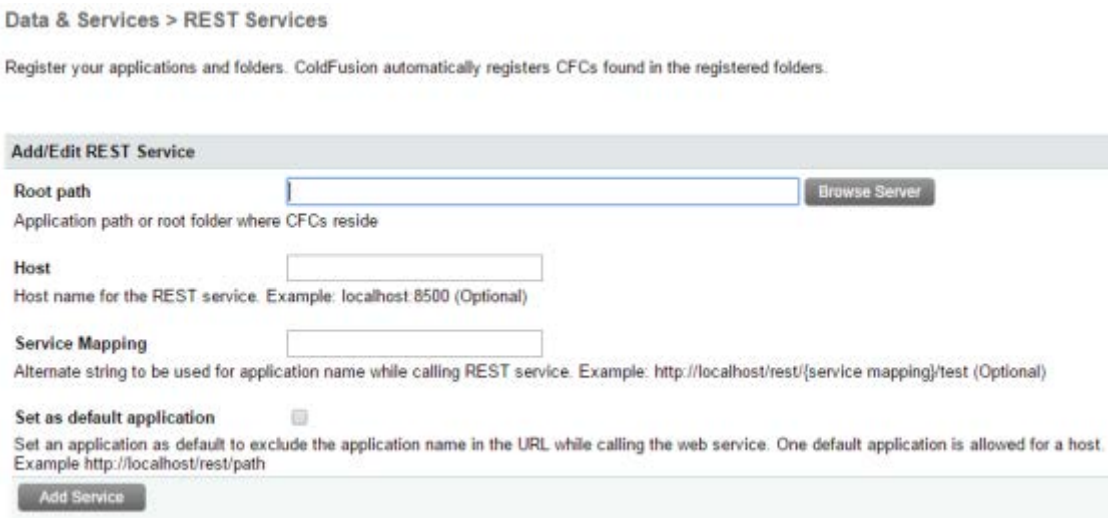
```
<cfcomponent rest="true" restPath="/cookieService"
produces="text/plain" >
    <!--- Test with various produces --->
    <cffunction name="sayPlainHelloUser" responseMessages="404:Not
Found,200:successful,10:notdefine" access="remote" returnType="String"
httpMethod="GET" produces="text/plain">
 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

Register your CFC application

Start ColdFusion Administrator. Click Data & Services > REST Services on the left pane and add configuration values as per the instructions in the following dialog.



1. Enter the root path where REST CFCs are available in 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.

Access swagger api-docs

The Swagger representation of the API is comprised of two file types:

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.

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"
                        },
                         {
                            "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"
            ],
            "properties":{
                "address":{
                    "$ref":"IndiaAddress"
                },
                "name":{
                    "type":"string"
                },
                "age":{
                    "type":"number",
                    "format":"double"
```

```
                }
            }
        },
        "IndiaAddress":{
            "id":"IndiaAddress",
            "description":"India address fromat",
            "required":[
            ],
            "properties":{
                "country":{
                    "type":"string"
                },
                "pin":{
                    "type":"number",
                    "format":"double"
                },
                "street":{
                    "type":"string"
                },
                "district":{
                    "type":"string"
                },
                "houseNo":{
                    "type":"number",
                    "format":"double"
                },
                "state":{
                    "type":"string"
                }
            }
        }
    }
```

```
215     }
```

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.

Swagger doc Field Name	Type	Description	CF Fields
SwaggerVersion	String	**Required.** Specifies the Swagger Specification version being used.	Update programmatically using API Manager
apis	Resource Object	**Required.** Lists the resources to be described by this specification implementation. The array can have 0 or more elements	N/A
apiVersion	string	Provides the version of the application API	Modify using application.cfc file
info	Info Object	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.	Modify using application.cfc file
authorizations	Authorizations Object	Provides information about the authorization schemes allowed on this API. The type of the authorization scheme. Values MUST be either"basicAuth", "apiKey" or "oauth2".	Update programmatically using API Manager

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.

Swagger doc Field Name	Type	Description	CF Field

basePath	string	**Required**. The root URL serving the API.	Add programmatically while parsing CFC
consumes	[string]	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.	Cfcomponent.consumes
produces	[string]	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.	Cfcomponent.produces
resourcePath	string	The relative path to the resource, from the basePath, which this API Specification describes.	Cfcomponent.restpath
apis	[API Object]	**Required**. A list of the APIs exposed on this resource. There MUST NOT be more than one API Object per path in the array.	Details in API Object
apiVersion	string	Provides the version of the application API (not to be confused by the (specification version).	N/A
swaggerVersion	string	**Required.** Specifies the Swagger Specification version being used.	N/A
authorizations	Authorizations Object	A list of authorizations schemes required for the operations listed in this API declaration. Individual operations may override this setting. If there are multiple authorization schemes described here, it means they're all applied.	Add programmatically as API Manager will update Authorization info
models	Models Object	A list of the models available to this resource. Note that these need	Generate programmatically

		to be exposed separately for each API Declaration.	

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.

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

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.

Swagger doc Field Name	Type	Description	CF Field
authorization	Authorizations Object	A list of authorizations required to execute this operation	Programmatically from API Manager
consumes	[string]	A list of MIME types this operation can consume.	Cffunction.consumes
method	String	**Required.** The HTTP method required to invoke this operation. The value MUST be one of the following values:	Cffunction. httpmethod

		"GET", "HEAD", "POST", "PUT","PATCH", "DELETE", "OPTIONS". The values MUST be in uppercase.	
nickname	String	**Required.** A unique id for the operation that can be used by tools reading the output for further and easier manipulation	Cffunction.name
notes	String	A verbose explanation of the operation behavior.	Cffunction.hint
parameters	[Parameter Object]	**Required.** The inputs to the operation. If no parameters are needed, an empty array MUST be included.	Details in parameter object
produces	[string]	A list of MIME types this operation can produce.	Cffunction.produces
responseMessages	[Response Message Object]	Lists the possible response statuses that can return from the operation.	New parameter introduced in Cfunction
summary	String	A short summary of what the operation does. For maximum readability in the swagger-ui, this field SHOULD be less than 120 characters.	Cffunction.description

Parameter object schema

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".

Swagger doc Field Name	Type	Description	CF Field
allowMultiple	boolean	Another way to allow multiple values for a "query", "header" or "path" parameter.	Not available in ColdFusion.
description	string	**Recommended.** A brief description of this parameter.	Cfargument.hint
name	string	**Required.** The unique name for the parameter.	Cfargument.name

paramType	string	**Required.** The type of the parameter. The value MUST be one of these values: "path", "query", "body","header", "form" Note: As per spec swagger dosen't support "Cookie", "Matrix" paramtype which we have in ColdFusion	Cfargument. restargsource
required	boolean	A flag to note whether this parameter is mandatory.	Cfargument.required

CFC/Swagger/Java types comparison

CFC	**Swagger**	**Java**	**Additional information**
string	string	string	
uuid	string	string	
guid	string	string	
query	custom model	coldfusion.xml.rpc.DocumentQueryBean	
void	void		for argument map to "body"
numeric	number(format double)	Double	
boolean	boolean	boolean	
date	string(format date)	java.util.Calendar	
any	object	java.lang.Object	
array	array of objects	java.lang.Object[]	
binary	??	byte[]	
struct	custom model	java.util.Map	
xml	string	org.w3c.dom.Documents	

Bugs/Issues fixed in this release

You can refer to a list of all the issues fixed in this release at **IssuesFixed.pdf** file, which is available in the prerelease path

PDF enhancements

ColdFusion Raijin release provides new actions for cfpdf tag, such as sanitization, import and export of metadata, import and export of comments, attaching files, annotation (stamp) and enhancement to pdf archival.

Sanitize

Overview

Sanitize action removes metadata from your pdf document so that sensitive information is not inadvertently passed along when you publish your PDF.

Sanitize action removes metadata, attached files, scripts, embedded search indexes, stored form data, review and comment data, obscured text and images, unreferenced data, links, actions and javascripts, and overlapping objects.

Usage scenario

A report may be sanitized before publishing to the general audience.

When dealing with classified information, sanitization attempts to reduce the document's classification level, possibly yielding an unclassified document.

Syntax

```
<cfpdf

required

action="sanitize"

source = "absolute or relative pathname to a PDF file|PDF document
variable|cfdocument variable"

Use any one of the following attributes:

destination = "PDF output file pathname"

name = “PDF document variable name”>
```

Attributes

Attribute name	Req/optional	Description
action	Req	Action to take: sanitize
source	Req	absolute or relative pathname to a PDF file| PDF document variable or cfdocument variable

Attribute name	Req/optional	Description
destination/name	Req	destination = "PDF output file pathname". name = "PDF document variable name". If the destination file exists, set the overwrite attribute to true. If the destination file does not exist, ColdFusion creates the file, if the parent directory exists.

Code samples

```
<cfpdf action="sanitize" source="C:\font1.pdf" name="Myvar"
overwrite="true">
```

Exporting and importing comments

Overview

Comments can be exported from a PDF document to XFDF file. Comments can also be imported from XFDF file in the specified PDF document.

For exporting comments, you can specify the source and the destination of the XFDF file. For importing comments, you can specify the source of the pdf file, the source of the XFDF file and the destination path.

Usage scenario

A PDF is sent for review to multiple reviewers. Reviewers reply back with their comments in the PDF. The sender gets the PDF back on email, extracts the comments first, and creates a master copy where all the comments are merged back to one single PDF.

Syntax

```
Export Comments:
 <cfpdf
 required
 action="export"
 type="comment"
 source = "absolute or relative pathname to a PDF file|PDF document
variable|cfdocument variable"
 exportTo = "destination of xfdf file">
Import Comments:
<cfpdf
```

```
required

action="import"

type="comment"

source = "absolute or relative pathname to a PDF file|PDF document
variable|cfdocument variable"

importFrom = "source of xfdf file"

Use any one of the following attributes:

destination = "PDF output file pathname"

name = “PDF document variable name”>
```

Code samples

Export comment

```
<cfpdf action="export" type="comment" source="c:\source.pdf"
exportto="c:\destination.xfdf">
</cfpdf>
```

Import comment

```
<cfpdf action="import" type="comment" source="c:\source.pdf"
importfrom="c:\import.xfdf" destination="c:\destination.pdf">
</cfpdf>
```

Sample xfdf file

A sample xfdf file (cf_comment3_exported.xfdf) is attached to this pdf for your reference. You can click the attachment icon on the left pane in pdf and save the file.

Attributes

Attribute name	Action	Req/optional	Description
action	N/A	Req	Action to take: export import
Type	comment	Req	To specify if it is for importing or exporting comment
source	export import	Req	absolute or relative pathname to a PDF file
destination/name	Import	Req	destination = "PDF output file pathname". name = “PDF document variable name”.

Attribute name	Action	Req/optional	Description
			If the destination file exists, set the overwrite attribute to yes. If the destination file does not exist, ColdFusion creates the file, if the parent directory exists.
exportTo	comment	Req	source of xfdf file for comment
importFrom	comment	Req	source of xfdf file for comment

Exporting and importing metadata

PDF documents created in Acrobat 5.0 or later contain document metadata in XML format. Metadata includes information about the document and its contents, such as the author's name, keywords, and copyright information, that can be used by search utilities.

For exporting metadata, you have to specify the source of the pdf and destination of the XMP file. For importing metadata, you have to specify the source of the pdf, the source of XMP file and the destination for the file to be placed. The Extensible Metadata Platform (XMP) provides Adobe applications with a common XML framework that standardizes the creation, processing, and interchange of document metadata across publishing workflows. You can save and import the document metadata XML source code in XMP format, making it easy to share metadata among different documents.

Note

Virtual file system is not supported for import and export actions.

```
Export Metadata:
<cfpdf
required
action="export"
type="metadata"
source = "absolute or relative pathname to a PDF file|PDF document variable|cfdocument variable"
exportTo = "destination of xmp file">
Import Metadata:
<cfpdf
required
action="import"
```

```
type="metadata"

source = "absolute or relative pathname to a PDF file|PDF document
variable|cfdocument variable"

importFrom  = "source of xmp file"

Use any one of the following attributes:

destination = "PDF output file pathname"

name = “PDF document variable name”>
```

Attributes

Attribute name	Action	Req/optional	Description
action	N/A	Req	Action to take: export import
Type	metadata	Req	To specify if it is for importing or exporting metadata
source	export import	Req	absolute or relative pathname to a PDF file
destination/name	import	Req	destination = "PDF output file pathname". name = “PDF document variable name”. If the destination file exists, set the overwrite attribute to yes. If the destination file does not exist, ColdFusion creates the file, if the parent directory exists.
exportTo	metadata	Req	source of xmp file for metadata.
importFrom	metadata	Req	source of xmp file for metadata.

Code samples

Export metadata

```
<cfpdf action="export" type="metadata" source="c:\source.pdf"
exportto="c:\destination.xmp"></cfpdf>
```

Import metadata

```
<cfpdf action="import" type="metadata" source="c:\source.pdf"
importfrom="c:\import.xmp" destination="c:\destination.pdf">
</cfpdf>
```

Sample xmp file

A sample xmp file (cf_metadata3_exported.xmp) is attached to this pdf for your reference. You can click the attachment icon on the left pane in pdf and save the file.

PDF archiving

Overview

You can archive pdf files. PDF/A is an ISO -standardized version of the Portable Document Format (PDF) specialized for the long term digital preservation of electronic documents. It guarantees a pdf can be viewed in the distant future. During PDF conversion, the file that is being processed is checked against the specified standard. PDF/A differs from PDF by prohibiting features ill-suited to long-term archiving, such as font linking (as opposed to font embedding).

In earlier versions of ColdFusion we used to support PDF/A-1b standard. In ColdFusion Raijin, we support PDF/A-2b.

Syntax

```
<cfpdf
required
action="archive"
source="#sourcefilename#"
Use any one of the following attributes:
destination = "PDF output file pathname"
name = “PDF document variable name”>
optional:
standard = "2b"/>
```

Attributes

Attribute name	Req/optional	Description
action	Req	Action to take: archive
source	Req	absolute or relative pathname to a PDF file| PDF document variable or cfdocument variable

Attribute name	Req/optional	Description
destination/name	Req	destination = "PDF output file pathname". name = “PDF document variable name”. If the destination file exists, set the overwrite attribute to yes. If the destination file does not exist, ColdFusion creates the file, if the parent directory exists.
standard	Optional	Specify the standard that you want to apply. Default value is “2b”.

Code samples

```
<cfpdf action="archive" source="C:\Hello World.pdf" name="Myvar"
overwrite="true" standard="2b"/>
```

Attaching files to a pdf

Overview

You can attach PDFs and other types of files to a PDF. If you move the PDF to a new location, the attachments move with it. By using cfpdfparam, you can specify the location of the attachment, the filename for the attachment, description of the attachment, encoding, and mimetype for attaching file. Files can be attached at document level.

Syntax

```
Attach file:
<cfpdf
 required
 action="addAttachments"
 source = "absolute or relative pathname to a PDF file|PDF document
 variable|cfdocument variable|directory path"
 Use any one of the following attributes:
 destination = "PDF output file pathname"
 name = “PDF document variable name”
<cfpdfparam
   required
   source= "path of attachment"
   filename = "filename for the attachment"
   encoding = "encoding for filename" >
```

```
   optional
   description = "descriptive text"
   mimetype = "eg: application/pdf, text/html">
/>
```

Note

- At least, one `cfpdfparam` is required to be used.
- Virtual file system is not supported for addAttachments action.

Attributes

Attribute name	Req/optional	Description
action	Req	Action to take: addAttachments
source	Req	absolute or relative pathname to a PDF file| PDF document variable or cfdocument variable
filename	Req	filename for the attachment
mimetype	Optional	Some of the sample mimetype attribute values are: text/html, application/pdf.
encoding	Req	type of encoding for filename
description	Optional	Description for the attachment
destination/name	Req	destination = "PDF output file pathname". name = "PDF document variable name". If the destination file exists, set the overwrite attribute to yes . If the destination file does not exist, ColdFusion creates the file, if the parent directory exists.

Code samples

```
<cfpdf action="addAttachments" source="C:\cf_file3.pdf"
name="Myvar">
<cfpdfparam  source="C:\file1.txt" filename="file1.txt"
description="descriptive text"  encoding="UTF-8" />
<cfpdfparam  source="C:\file2.txt" filename="file2.txt"
description="descriptive text"   encoding="UTF-8"/>
</cfpdf>
```

Adding stamp

Overview

You apply a stamp to a PDF in the same way you apply a rubber stamp to a paper document. You can choose from a list of predefined stamps.

You can specify the page to which he wants to add the stamp, the location at which he wants to add stamp, the icon name, the content of the stamp.

Syntax

```
<cfpdf
 required
 action="addStamp"
 source = "absolute or relative pathname to a PDF file|PDF document
 variable|cfdocument variable"
 Use any one of the following attributes:

 destination = "PDF output file pathname"
 name = “PDF document variable name”>

 <cfpdfparam
  pages = "page number|page range|comma-separated page numbers"
  coordinate = "x1,y1,x2,y2"
  iconName = "name of icon"
  note = "content of stamp" >
/>
```

Attributes

Attribute name	Req/optional	Description
action	Req	Action to take: addStamp
source	Req	absolute or relative pathname to a PDF file| PDF document variable or cfdocument variable
destination/name	Req	destination = "PDF output file pathname". name = “PDF document variable name”. If the destination file exists, set the overwrite attribute to yes . If the destination file does not exist, ColdFusion creates the file, if the parent directory exists.
iconname	optional(default is draft)	Mention the name of the icon. For a detailed list of supported icon names, refer to **the list** below.

Attribute name	Req/optional	Description
note	optional	A brief description of the content associated with icon.
pages	Req	Specify the page number to add stamp.
coordinates	Req	Add the position of co-ordinates in the pages to add stamp

Code samples

```
<cfpdf action="addStamp" source="#sourcefile#" destination="#destinationfile#" >
<cfpdfparam pages="2" coordinate = "397,532,519,564" iconname="Approved"
note="stamp1"/> </cfpdf>
```

List of icon names

Approved, Experimental, NotApproved, AsIs, Expired , NotForPublicRelease, Confidential, Final, Sold, Departmental, ForComment, TopSecret, Draft, ForPublicRelease.

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 code analyzer is a new feature introduced in ColdFusion Raijin. Security analyzer feature of the server is integrated into ColdFusion Builder now to enable developers avoid common security pitfalls/vulnerabilities while writing ColdFusion code.

By using this new feature in the builder, developer can view the following options:

- 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.

It helps developers to write safe and secure code.

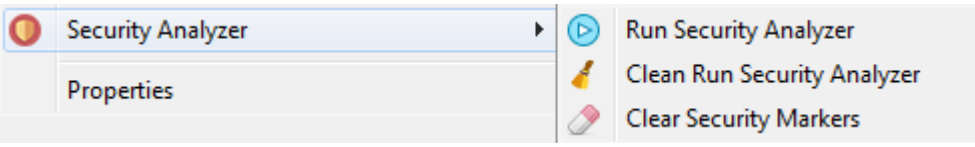
Note
This feature is available only in development server, it is not available in the production server.

The security analyzer feature of the server is exposed as a service, a request to which is made by the builder. The developers can get a list of security vulnerabilities for a file or for a folder or for an entire project with a right-click on them and selecting Run Security Analyzer.

Accessing security analyzer in builder

You can access security code analyzer in ColdFusion builder by following the steps below:

1. Right-click the project folder or project file in the navigator pane.
2. Choose **Security Analyzer> Run Security Analyzer** menu item.



As shown in the above snapshot, you have three options to choose in Security Analyzer:

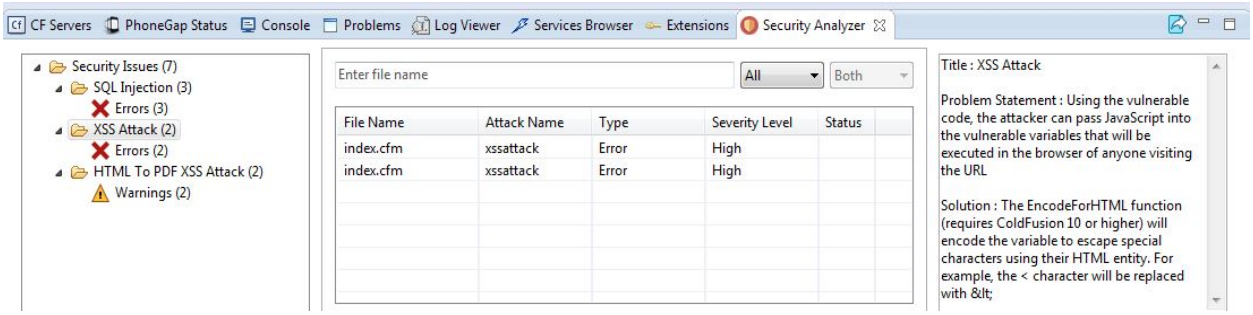
- **Run Security Analyzer –** analyzes the code for vulnerabilities and displays the vulnerabilities in Security Analyzer view.
- **Clean Run Security Analyzer -** analyzes the code for vulnerabilities in a clear environment. It clears the ignored vulnerabilities (which are marked as Ignore during the Run Security Analyzer) and displays all vulnerabilities for the project.
- **Clear Security Markers –** clears all the security vulnerabilities for a selected resource. You need to run the security analyzer again to view the vulnerabilities for your resource.

Using security analyzer

You can start using the security analyzer for your project folder or file by following the steps below:

1. Create a new ColdFusion project or use an existing project.
   Ensure 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.

   Note: If there is any syntax error in the code, a warning pop-up dialog displays:
   "*n number of files are not scanned due to syntax error. Please check the Syntax Error view for more information.*"
3. Click **OK**.
   You can view all the vulnerabilities in the bottom pane of the Editor as shown in the sample snapshot 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 and 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. Also, you can choose the security issue type as Error or Warning in the **Both** 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

   Re-running 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 actually fixed, then server reports these errors.
6. Click the export arrow icon on the upper-right corner of the Security Analyzer pane to export all the vulnerabilities to a report.htm file. You can view the graphical representation of all vulnerabilities for your resource in the exported file.

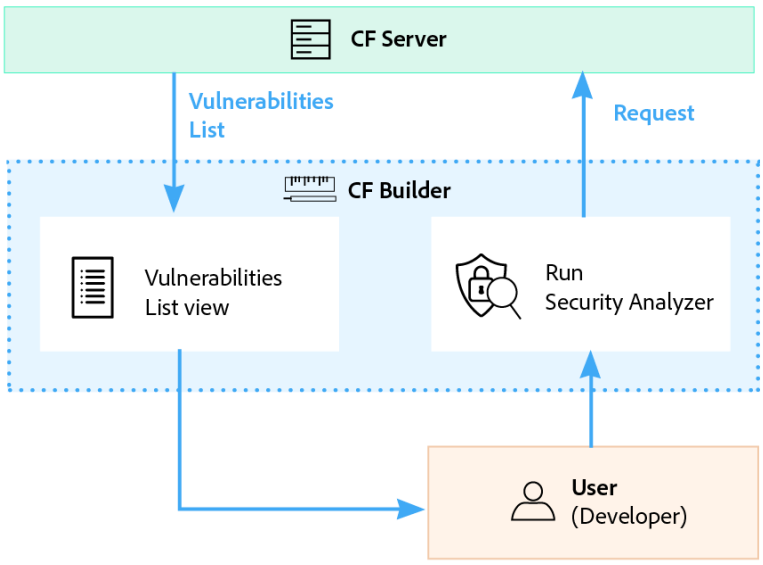
Syntax Error View

The Syntax Error view lists the files that are not scanned by the Security Analyzer. Double click the filename or path. It launches the respective file in the editor.

Workflow

The workflow of Security Analyzer feature:

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 on the gutter.
5. Also, by single click, you get a brief description about the attack and about possible ways to avoid it.



List of security vulnerabilities

The following list of security vulnerabilities are displayed to developers:

- SQL Injection
- XSS Attack
- PDF XSS Attack
- CSRF Attack
- Cookies Validations
- CFLocation Validation
- Passwords
- File upload Validation
- Get vs Post
- File injection
- Unnamed application

The detailed descriptions of each of these security vulnerabilities are mentioned below.

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 may also try to post XSS code that can 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>
  ```
  This is a simple example of XSS attack.
- `<cfparam name = "id12" default = "my default value" type="string">`
- `<cfoutput>#id12#</cfoutput>`

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 HTML is rendered by the server, You should be cautious while using 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 quite different from a client side cross site scripting attack. Some of the risks include denial of service, potential exploit for unknown vulnerabilities in Webkit, and network firewall bypass as the server may 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>
  ```

All the above code snippets are vulnerable to PDF XSS attack because unknown variables are used in the areas where the content is rendered for a PDF.

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.

Cookies Validations

Cookies can contain sensitive information that should not get leaked.

Vulnerable scenarios

- `<cfcookie name="sample" value="random" httponly="false" secure="false">`

  When both “httponly” & “secure” attributes are set to false explicitly, the code is vulnerable.

- `<cfcookie name="sample" value="random" httponly="true" secure="false">`

  When either of “httponly” & “secure” attributes is set to false explicitly, the code is vulnerable.

- `<cfcookie name="sample" value="random" >`

  When “httponly” & “secure” attributes are not set, default value of false is taken, making the code vulnerable.

CFLocation Validation

Avoid appending the session identifiers to the URL query string. End users will 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.

Passwords

Passwords should not be stored in plain text.

Vulnerable scenarios

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

In all of the 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

Sensitive information should not be sent over GET method.

Vulnerable scenarios

- ```
  <cfform method="get" action="sayHello.cfm">
      <cfinput name="userName" type="text" >
      <cfinput name="token" value="#CSRFGenerateToken()#"
  type="hidden" >
      <cfinput name="submit" value="Say Hello!!" type="submit" >
  </cfform>
  ```

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

- ```
  <cfform action="sayHello.cfm">
      <cfinput name="userName" type="text" >
      <cfinput name="token" value="#CSRFGenerateToken("a")#"
  type="hidden" >
      <cfinput name="submit"  value="Say Hello!!" type="submit" >
  </cfform>
  ```

  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 may read a configuration file that is not meant to be public.

Vulnerable scenarios

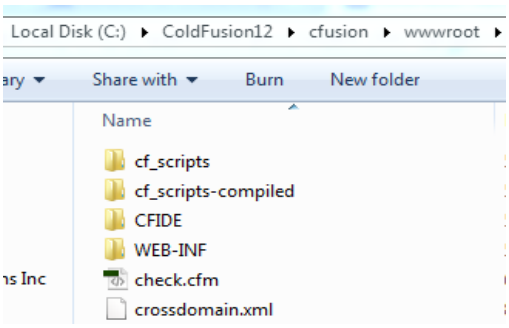
- ```
  <cfinclude template="constant/#somepath#">
  ```
- ```
  <cffile action="write" file="#filevar2#">
  ```
- ```
  <cfscript>

  myfile = DirectoryDelete(var);

  </cfscript>
  ```

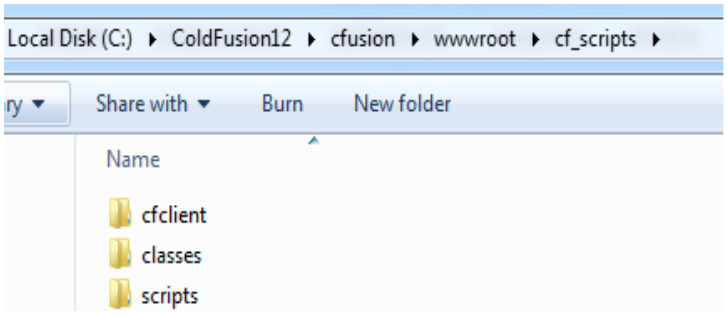
In all of the above scenarios, an unknown variable is used for file path or directory path, so they are vulnerable.

Unnamed Application

Avoid using unnamed application. An unnamed application is one where an application is not provided a name in the Application.cfm/Application.cfc.

Vulnerable scenarios
  ```

Application.cfm

- `<cfapplication name3=”app”>`
  …

  ”name” keyword is not set to give a name to the application.

- `<cfset var=”variable”>`

  …

  keyword ”name” is missing to name the application

Application.cfc

- ```
  <cfcomponent>
  <cfset this.SessionManagement=true>
  …
  </cfcomponent>
  ```

  ”this.name” is missing to give a name to the application.

- ```
  <cfcomponent>
  <cfset this.SessionManagement=true>
  <cfset this.name23=”app”>
  …
  </cfcomponent>
  ```

  Here, the keyword ”name” is not used to set the application name.

- ```
  component
  {
    this.name4=”app”;
  };
  ```

  Here, the keyword ”name” is not used to set the application name.

CFIDE-script separation

In earlier versions of ColdFusion, JavaScript files from ajax files and other scripts were accessible to users from CFIDE folder under root. CFIDE folders are generally protected by administrators for security reasons. To provide better lock down on webserver, now, CFIDE access is removed on web server. By default, Administrator access is blocked on the web server, it is available only on port 8500.

We have separated scripts folder from CFIDE. The CFIDE/scripts, CFIDE/classes, and CFIDE/cfclient folders have been moved to the wwwroot folder under cf_scripts sub folder.
  ```


Local Disk (C:) ▸ ColdFusion12 ▸ cfusion ▸ wwwroot ▸
Share with
Burn
New folder
Name
cf_scripts
cf_scripts-compiled
CFIDE
WEB-INF
check.cfm
crossdomain.xml


Local Disk (C:) ▸ ColdFusion12 ▸ cfusion ▸ wwwroot ▸ cf_scripts ▸
Share with
Burn
New folder
Name
cfclient
classes
scripts

Language enhancements

In ColdFusion Raijin, the language enhancements that have been introduced are safe navigation operator (“?.”) and collections support along with some other minor enhancements.

Safe navigation (?.)

Safe navigation operator can be used to access members of a Struct or values of an object.

Instead of "." using the safe operator ensures that exception is not thrown when the variable is not defined.

Sample usage - 1

<cfoutput>employee?.getDesignation</cfoutput> or <cfoutput>employee?.getDesignation()</cfoutput>

This code does not throw exception even if variable **employee** is not defined. If **employee** is defined, then it works like a dot operator.

> **Note**
> If safe navigation evaluates to an undefined value and if that value is assigned to a variable, usage of such variable could lead to an exception. For example, `<cfset x = employee?.getDesignation>.` In this example, if variable **employee** is not defined, then x will be assigned as undefined.

Simple usage

For writing the safe code, you were supposed to do something like this:

```
if(isDefined("foo"))
    writeOutput(foo.bar());
else
    writeOutput("not defined");
```

Now, you just have to write

```
writeOutput(foo?.bar());
```

Chaining

```
writeOutput(employee?.name?.firstname?.trim());
```

The above code does not output any value if any of the following is undefined/null:

`employee` OR
`employee.name` OR
`employee.name.firstname.`

```
writeOuptut(employee?.name.firstname.trim());
```

This code will throw an exception if employee is defined but employee.name is undefined.

Sample usage - 2

Current code

```
<cffunction name="employee">
    <cfreturn >
</cffunction>
<cfoutput>#employee()#</cfoutput>  //prints empty string(default behavior)
<cfoutput>#employee().trim()#</cfoutput> //throws error - Value must be initialized before use.
```

Safe Navigation Operator code

```
<cffunction name="employee">
    <cfreturn >
</cffunction>
<cfoutput>#employee()?.trim()#</cfoutput>  //Does not print any output
```

Collection support (Ordered)

StructNew("Ordered") creates a struct that maintains insertion order.

All other ways of creating struct will make normal struct without breaking backward compatibility.

Currently, keys are fetched in random order while looping over the struct. But, when you loop over the struct using StructNew("Ordered"), keys are fetched according to the insertion order.

All other properties and function will work as a normal struct.

Example

```
<cfset str = StructNew("Ordered")>
```

On iterating this struct, you will get the values in insertion order, which is the way you inserted the values.

```
<!--- Create a structure and set its contents. --->
<cfset departments=StructNew("Ordered")>
<cfset val=StructInsert(departments, "John", "Sales")>
<cfset val=StructInsert(departments, "Tom", "Finance")>
<cfset val=StructInsert(departments, "Mike", "Education")>
```

```
<!--- Build a table to display the contents --->
<cfoutput>
<table cellpadding="2" cellspacing="2">
    <tr>
        <td><b>Employee</b></td>
        <td><b>Department</b></td>
    </tr>

    <!--- Use cfloop to loop through the departments structure.
The item attribute specifies a name for the structure key. --->

    <cfloop collection=#departments# item="person">
        <tr>
            <td>#person#</td>
            <td>#Departments[person]#</td>
        </tr>
    </cfloop>
</table>
</cfoutput>
```

Output will always be generated in order:

John Sales

Tom Finance

Mike Education

Collection support (Sorted)

Similar to ordered struct you can also create a sorted struct passing order type as sorted in structNew, like structNew("Sorted").

StructNew("Sorted') accepts a second argument(sort order). Possible values of Sort order are "Asc", "Desc" or UDF.

In case of "Asc" or "Desc", sorting order is case insensitive alphabetical order.

The default value of sort order (second parameter) to structNew is "Asc", which maintains sorting order on ascending case insensitive alphabetical order.

Note
The sorting is always applied on keys.

There is no way to do a numeric sorting by default, you can do numeric sorting (or any custom sorting for instance) by passing a UDF.

Example-1

<cfset str = structNew("sorted")>

By default, this will maintain case insensitive alphabetical ascending order on keys.

Example-2

<cfset str = structNew("sorted", "asc")>

For ascending order, pass the second argument as “asc”.

Example-3

<cfset str = structNew("sorted", "desc")>

To reverse the order, pass the second argument as "desc"

Example-4

For custom sorting, UDF can be passed as second argument, just the way it is done in structSort. In structSort, the sorting is performed based on values whereas in structNew, the sorting is performed based on keys.

```
<cfscript>
     public numeric function numericDesc(x, y) {
        if(x < y)
            return (-1);
        else if(x == y)
            return 0;
        else if(x > y)
            return 1;
    }
    strSortedNum = structNew("sorted", numericDesc);
    strSortedNum.3 = "three";
    strSortedNum.5 = "five";
    strSortedNum.1 = "one";
    strSortedNum.6 = "six";
    strSortedNum.2 = "two";
    strSortedNum.4 = "four";
</cfscript>
<cfloop collection="#strSortedNum#" item="index">
    <cfoutput>#index#: #strSortedNum[index]#</cfoutput><br>
</cfloop>
```

Output

1: one

2: two

3: three

4: four

5: five

6: six

Scope search

Generally, a referenced variable is searched in various scopes before a match is found. In cases where variable is not found in common scopes like function, local scope, and variable scopes, the search continues in implicit scopes, which can be time consuming. Implicit scope search can be disabled by using a new application setting called **searchImplicitScopes**. It accepts a boolean value.

When set to false, an un-scoped variable is not searched in implicit scopes.

Arrays

Pass by reference

By default, arrays are passed by value, which is slower as compared to passing arrays by reference. You can now choose to pass arrays by reference by using a new application setting called **passArrayByReference**. It accepts a boolean value.

When set to true, if an array is passed as a UDF argument, it is passed by reference.

Query mutable attribute

You can now create immutable queries by setting this attribute in the cfquery tag. It accepts a boolean value.

When set to false, the query result cannot be modified using the query functions such as `querySetRow`, and `querySetCell`. A non-mutable query result is not duplicated each time the cfquery tag is used, and is therefore faster.

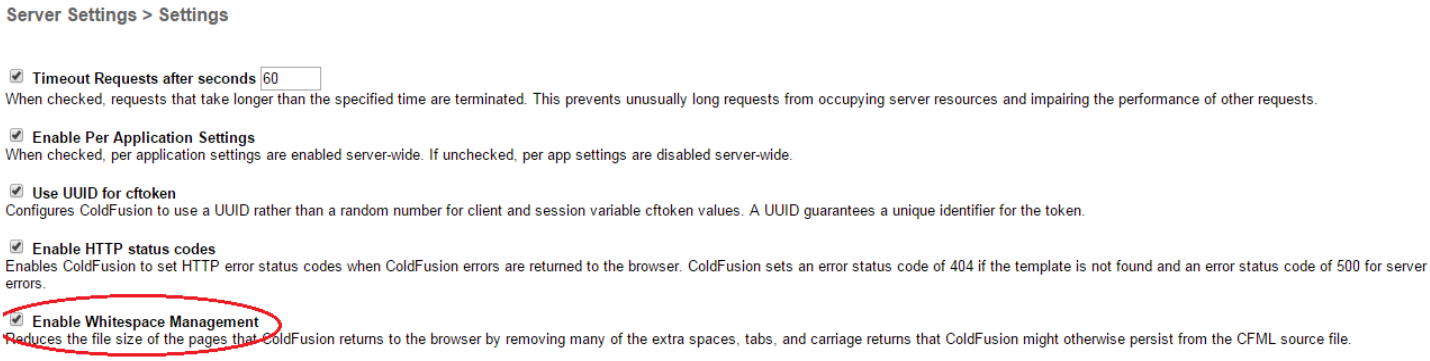
Usage

```
/*
* @file:  Application.cfc
* @description:  Using the performance settings in Raijin - An
example.
*/

component output="false" displayname=""{
this.name = "app_perf_stngs_optmzd";
this.passArrayByReference = true;
```

```
     this.searchImplicitScopes = false;
public void function onApplicationStart(){
arr_isthread_safe = false;
           arr = arrayNew(1, arr_isthread_safe);
           qry = new query( datasource="cfdocexamples", mutable=false,
cachedwithin="#CreateTimeSpan(0, 0, 30, 0)#" );
    }
}
```

Whitespace management

ColdFusion Raijin has a new way of managing whitespaces. In the administrator, the server level setting that used to suppress whitespaces at runtime can suppress them now at compile time.



Other enhancements

The following other language enhancements are included in ColdFusion Raijin release.

`ArrayFindNoCase`

Array(first argument) should be array of Strings or Boolean or Numeric only. And object to search(second argument) should be either String or Boolean or Numeric. Otherwise it will throw exception.

`replaceListNoCase`

It is the case-insensitive version of replaceList.

cfloop tag

A new attribute "item" is introduced in array cfloop for `file` and `list`. Earlier, only index was supported in cfloop which was not a clean behavior. Now, you can use both item and index or either of them.

- If item and index are present, item will hold the element and index will hold index.
- If only item is present, item will hold the element.
- If only index is present, index will hold the element (to support backward compatibility)

cfcollection tag

The “path” attribute is now optional. All collections are created in the location specified in **Solr Home** field in the ColdFusion administration page.

Miscellaneous Fixes

4018168	When a web service returns array, the client side receives it as Object[]. So ArrayAppend was adding the whole Object[] as an element in the destination array. Changing the behavior in web service client to create Array from Object[] and return will introduce backward compatibility issue for existing webservice clients. In the ArrayAppend function, if the second argument to add is an Object[], you can merge Object[] into destination Array.
3791418	When you register RESTful web services using auto registration, the “useHost” attribute’s default value changes “true”. Now, you can register a REST service with the host name that initializes the application. So you cannot access the REST service through any other host.

Issues Fixed

You can refer to a list of all the issues fixed in this release at **IssuesFixed.pdf** file, which is available in the prerelease path.

CLI

In ColdFusion Raijin, we have introduced a new Command Line Interface (CLI) for developers to run their cfm scripts without having to start an associated ColdFusion server.

`cf.bat` can be used 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, etc.,
- Database operations for monitoring, debugging, etc.,
- Net operations like mailing an error log or thread dump to a system admin, and so on.

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

Positional and named arguments can be passed 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.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 features like scheduled tasks which need a background server do not work in CLI.