CONTENTS INCLUDE:

Getting Started with ColdFusion 9

By Terry Ryan

ABOUT ADOBE COLDFUSION

Adobe ColdFusion is a rapid application development toolkit that is written in Java and runs on top of the JVM. Its core philosophies are: make things easy, and provide RAD without restricting developers.

This Refcard will take you through getting started with ColdFusion 9. It will take you through the various pieces and languages that comprise ColdFusion. It will also list various ways of communicating with databases, other tools, services, and languages. This Refcard is aimed at programmers in other languages that are considering taking a look at ColdFusion.

ColdFusion 9

ColdFusion 9 is currently in public beta and scheduled to be released before then end of 2009. It is a significant release in that it includes substantial additions to the feature set, including: Hibernate, Ehcache, full language support for scripting, interactivity with Java portal servers, Microsoft SharePoint, and Microsoft Office documents

GETTING STARTED

Installing ColdFusion

You can download a free version of ColdFusion 9 Developer from http://adobe.com/go/centaur. Download the appropriate version for your OS, and install. The installer will take you through a few methods of install:

Standard

This installs ColdFusion as a stand-alone server. Despite the fact that ColdFusion runs on a J2EE server the stand-alone version bundles the J2EE server. This makes the J2EE server inaccessible, but the whole package easier to administer for newcomers to J2EE. This is the easiest way to get started with ColdFusion.

Multiserver

This will install ColdFusion as a single instance running on JRun. This will allow you to install multiple ColdFusion servers on your box if you wish. Additionally, JRun is a J2EE server capable of running other EAR or WAR files.

EAR/WAR file

This will package ColdFusion and your installation options into a J2EE EAR or WAR file. This file can then be deployed to the J2EE server of your choice.

PARTS TO COLDFUSION

ColdFusion consists of two parts: the server that runs ColdFusion, and the code you run on it. The server is a little more significant to the toolkit than server components in most other competing products.

Server

The server component of ColdFusion consists of an executable server that can run as a service or daemon on your OS. This server component handles processing ColdFusion requests but also stores a lot of configuration for your applications centrally. This allows you to configure your datasources in one place, and then just refer to them by alias in your code. You can also set things like mail server defaults, central code collections, path mappings, third party licenses, etc. Most of what you can set here can be overridden at the application level, but by setting it here, you can avoid having to store that information in your applications.

Code

ColdFusion uses two languages for writing code:

CFML

CFML is a tag-based language that prefixes all ColdFusion specific calls with "cf." Some developers use CFML to write their entire application. Others however, use CFML for front end code, display code where CFML mixes well with HTML and use scripting for their business logic. Most examples of code in this Refcard will be given in CFML except where showing the script version is particularly important, but they all have CFscript analogues.

CFScript

CFScript most closely resembles JavaScript, but is influenced by a few different styles of scripting including C, JavaScript and ActionScript. Until recently, CFscript was not as fully featured as CFML. As of ColdFusion 9, one should be able to do anything they can do in CMFL in CFScript.

ColdFusion 9

www.dzone.com

BASICS

Types

ColdFusion is weakly typed. Variables themselves have no concept of type, but the underlying data does. This means that the same variable may act as multiple classical types without having to be set for them. For example:

```
<cfset test = 1 />
```

The variable test can act like a numeric:

```
<cfset newTest = test + test />.
```

However it can also act like a string in the call:

```
<cfset stringLen = len(test) />.
```

There are two classes of variables by type in ColdFusion: Simple and Complex.

Simple

Simple variables are variables that are limited to one unstructured value. They include: String, Numeric, Boolean, or DateTime.

Complex

Complex variables are those that either contain structured elements or binary data. They include: structs, arrays, queries, XML, images, and objects.

Scopes

All variables in ColdFusion live in a scope. The scopes are just collections of variables that share similar sources or audiences. They are all ColdFusion struct variables.

Single Request Scopes

These scopes only exist for the life of one page. They can only be accessed by calls made in that page.

CGI	Contains variables created by the Web server. It gives information like requesting host, ip address, and complete URI information.
Form	These are values passed by a form Post to a page.
Request	Prefixing variables with "request" creates this scope. It exists across all calls, functions, or custom tags, contained within one single request.
URL	These are values passed by a form GET to a page or appended as a query parameter on the tail end of URL request.
Variables	This is the default scope for variables in a request that are not in another scope. It is only accessible within the main flow of the request, and not directly in any other call like a custom tag or object call.

Persistent Scopes

The following scopes exists for more than the life of one request. They are often the place for storing variables which are used across the life of a user's session or an application's life.

Application	Variables persist across the life of an application. They are accessible by any request that has the same application name.
Client	Variables persist across the life of a user's session in a particular application. They are accessible from any call for a particular user in a particular application. Unlike every other scope, its values can be stored on disk or database.
Cookie	Variables persist either across the life of the user's browser session, or according to a date that is specified when set.
Server	Variables persist across the life of the server. They are accessible by any call anywhere on the server. Also contains information about the server instance of ColdFusion, like version, OS, etc.
Session	Variables persist across the life of a user's session in a particular application. They are accessible from any call for a particular user in a particular application. Unlike the Client scope, the values are stored in server memory.

Component and UDF Scopes

The following scopes are used with CFCs and user defined functions (UDFs.)

Arguments	Variables that are passed in as arguments to a UDF. Values set here accessible only to the currently running call of the UDF and are therefore thread-safe.
This	In CFC the variables scope is accessible externally and to any of the internal methods. It persists across the life of the instance of the CFC. Not thread-safe.
Local	Private scope for a UDF . Values set here accessible only to the currently running call of the UDF and are therefore thread-safe. Created by prefixing variables with "local" or by setting with "var" keyword.
Variables	In CFC the variables scope is accessible internally to the any of the methods and persists across the life of the instance of the CFC. Not thread-safe.

Custom Tag Scopes

The following scopes are used custom tags.

Attributes	Variables that are passed into a custom tag. Values set here accessible only to the currently running call of the custom tag and are therefore thread-safe.
Caller	An alias for the variables scope of the calling page of a custom tag.
Variables	The default scope for variables in a custom tag. Values here are only accessible in the custom tag itself.

Variables do not have to be explicitly scoped when referenced, but this is usually preferable for readability and performance. An unscoped variable call searches the following scope for existence:

1) Local (UDFs and CFCs only) Arguments (UDFs and CFCs only)
2) Variables
3) CGI
4) URL
5) Form
6) Cookie
7) Client

Session, Application, Server, and Request scopes will not yield their values without an explicit reference to them.

Types of ColdFusion Files

Page

A page is your basic ColdFusion file. Its file extension is ".cfm."

Here is a traditional "Hello World" page.

```
<cfset variable = "Hello World" />
<cfoutput>#variable#</cfoutput>
```

Custom Tag

A custom tag is a special type of page. It encapsulates commonly used code, and allows you to call it using <cfmodule>, or with the prefix <cf_. Its file extension is ".cfm."

Here is a basic custom tag named DisplayDate.cfm:

```
<cfparam name="attributes.date" default="#now()#"/>
<cfoutput>
<p>#DateFormat(attributes.date, "mmmm d, yyyy")#</p>
</cfoutput>
```

It takes a date that is passed in and formats it. If no date is passed in it sets the date to `Now()`.

You would call it like this:

```
<cf_displayDate >
<cf_displayDate date ="#CreateDate(2009,7,8)#">
```

It would display like this:

July 15, 2009

July 8, 2009

Component

A ColdFusion Component, or CFC for short, is a collection of properties and UDFs wrapped together. It is analogous to but

not exactly the same as a Java Class. Its file extension is ".cfm."

As of ColdFusion 9, properties yield implicit getters and setters.

They can be defined either in CFML:

```
<cfcomponent>
<cfproperty name="firstName" />
<cfproperty name="lastName" />
<cfproperty name="email"/>
<cffunction name="getDisplayName" returntype="string">
<cfreturn This.getFirstName() & " " & This.getLastName() />
</cffunction>
</cfcomponent>
```

Or in CFScript:

```
component{
property firstName;
property lastName;
property email;
string function getDisplayName(){
return This.getFirstName() & " " & This.getLastName();
}
}
```

Either version of the CFC would be called like this:

```
<cfset person = New cfcScriptExample() />
<cfset person.setfirstName('Terry') />
<cfset person.setlastName('Ryan') />
<cfset person.setEmail('terry@terrenceryan.com') />
<cfoutput>
<p>#person.getDisplayName()#</p>
<p>#person.getEmail()#</p>
</cfoutput>
```

And they would display this:

Terry Ryan

terry@terrenceryan.com

Application Framework

ColdFusion code is packaged into Applications. Applications are comprised of folders, files, and an Application.cfc. The Application.cfc stores settings that allow a developer to handle the behavior of applications, sessions and requests. There are a number of methods that ColdFusion can respond to in this file:

`onApplicationEnd`	Is not triggered when the application times outs or the server shuts down gracefully.
`onApplicationStart`	Is triggered the first time an application is called. You can use this function to initialize application variables.
`onError`	Is triggered whenever an uncaught exception is raised anywhere in the application.
`onMissingTemplate`	Triggered whenever a request is made for a cfm or cfc that doesn't exist in the application. Can be used to simulate virtual files.
`onRequest`	Replaces a request. If you use this method, you have to explicitly include the intended page or cfc. Useful for wrapping layout and formatting features around an application.
`onRequestEnd`	Triggered by the end of a request.
`onRequestStart`	Triggered by the start of a request.
`onSessionEnd`	Triggered the first time a particular user calls a URL in the application. Useful for setting session scoped variables.
`onSessionStart`	Triggered when user session ends, usually because the session timed out.

WORKING WITH DATA

ColdFusion was originally designed to be a language to bridge backend databases to Web pages. As such, handing record sets from database is a key part to using ColdFusion.

RDBMS

ColdFusion has built in support for many flavors of database including: Microsoft SQL, MySQL, Oracle, Derby, DB, and Postegres. However ColdFusion can interact with any RDBMS with a JDBC driver.

Datasources

In order to work with a particular database in ColdFusion, you must first create a datasource. A datasource is the collection of settings you use to communicate with a database, such as database type, sever, port, tablespace, or database, username and password. You collect all of these settings and give them a meaningful name. ColdFusion then allows you to refer to just the datasource name when connecting to the database. The server will handle maintaining connections, closing them, persisting them. In short ColdFusion abstracts database connections into datasources.

Queries

A recordset returned from a database will be turned into a special type of variables called, in ColdFusion, a query variable. `<cfloop>` and `<cfoutput>` along with a number of other tags allow you to pass a query attribute in for easy iteration. Within one of those iterators you can just refer to the column you are calling:

```
<cfloop query="personRS">
<cfset fullname = lastName & ", " & firstName />
</cfloop>
```

Alternately you can choose to manually loop through the query (broken up for readability):

```
<cfloop index="i" from="1" to="# personRS.recordCount#">
<cfset fullname = personRS .lastName[i]  />
<cfset fullname = fullname & ", " />
<cfset fullname = fullname &  personRS .firstName[i]  />
</cfloop>
```

In order to help looping and paging, queries have a few special properties.

`ColumnList`	A comma delimited list of columns in the query.
`CurrentRow`	The current row that is being accessed in the context of a cfoutput or cfloop call.
`RecordCount`	The total number of records.

SQL

Calling SQL is a matter of using `<cfquery>` with a datasource:

```
<cfquery name="resultSet" datasource="cfartgallery">
SELECT    * FROM    artists
</cfquery>
```

Additionally ColdFusion allows you to do parameterized queries using the cfqueryparam tag.

```
<cfquery name="resultSet" datasource="cfartgallery">
SELECT    *
FROM      artists
WHERE     artistid =
<cfqueryparam cfsqltype="cf_sql_integer" value="1" />
</cfquery>
```

Stored Procedure

Stored procedures can also be called from ColdFusion using the `<cfstoredproc>` tag.

ORM

As of ColdFusion 9, Hibernate is baked into ColdFusion and allows CFCs to be mapped to database tables.

Enabling ORM

To enable ORM for an application, the following settings are required in the Application.cfc:

`This.datasource`	The name of the datasource to use for ORM CFCs.
`This.ormenabled`	A Boolean which turns on ORM.

Basics

Assuming that a table named "person" with columns firstName, lastName, and email exists, and that there is a primary key personID, here is the code for a CFC named person.cfc that is mapped to the table:

```
<cfcomponent persistent="true">
<cfproperty name="personID" fieldtype="id" />
<cfproperty name="firstName" />
<cfproperty name="lastName" />
<cfproperty name="email"/>
</cfcomponent>
```

Data from the table can then be retrieved using the `EntityLoad()` function:

```
<cfset resultSet = entityLoad("artists") />
```

Or one record (with id = 5) can be requested:

```
<cfset resultSet = entityLoad("artists", 5) />
```

Creating a new record uses the `EntityNew()` function:

```
<cfset person = entityNew("artists") />
```

Then whether you are creating or updating a record, the code is the same:

```
<cfset person.setFirstName("Terry") />
<cfset person.setLastName("Ryan") />
<cfset person.setEmail("terry@terrenceryan.com") />
<cfset EntitySave(person) />
```

Deleting records would use the `EntityDelete()` Function:

```
<cfset EntityDelete(person) />
```

Relationships

CFCs support Hibernate relationships, and allow you to setup one-to-one, one-to-many, many-to-one, and many-to-many relationships.

Mappings

CFCs also support more advanced features of Hibernate including: Join Mapping, Collection Mapping, Inheritance Mapping, and Embedded Mapping.

Hibernate Options

In additional to all of the features that ColdFusion enables through CFCs, it also can take hbm.xml files to model your objects.

Also ColdFusion allows for making queries against the object model using HQL, Hibernate's SQL-like query language.

DISPLAYING DATA

Output

The `<cfoutput>` tag handles basic output of variables.

```
<cfset variable = "Hello World" />
<cfoutput>#variable#</cfoutput>
```

Pound Signs

You might notice the use of pounds signs. Pound signs are used to display or pass the literal value of a variable or operation. So normally when you set a variable to another variable, you just pass the variable:

```
<cfset displayName = FirstName & " " & LastName />
```

You can also pass the literal value:

```
<cfset displayName = "#FirstName# #LastName#" />
```

Dumping Data

One of the most helpful debugging tags in ColdFusion is `<cfdump>`. It will just output the value of the variable. For simple variables, this looks exactly like `<cfoutput>`, but for complex variables, `<cfdump>` will format a representation of the data to make it easy to understand. Take for example a call to a page:

http://localhost/index.cfm?action=edit&id=1

Those query parameters get passed to the URL scope as discussed Scopes. You can dump the URL scope because it is a struct:

```
<cfdump var="#url#"/>
```

It yields figure 1.

struct	
action	edit
id	1

If we were to dump a query:

```
<cfquery name="rs" datasource="cfartgallery" maxrows="5" >
SELECT    firstName, lastName, City, State
FROM      artists
</cfquery>
<cfdump var="#rs#">
```

It yields figure 2.

query

	CITY	FIRSTNAME	LASTNAME	STATE
1	Philadelphia	Aiden	Donolan	CO
2	Berkeley	Austin	Weber	CA
3	Los Angeles	Elicia	Kim	CA
4	Hollywood	Jeff	Baclawski	FL
5	Pierre	Lori	Johnson	SD

UI Components

In addition to being able to output variables to HTML templates, ColdFusion has built in a number of UI controls for handling of complex data.

INTERACTING WITH OTHER TECHNOLOGIES

ColdFusion can communicate with a large number of servers, products and languages using built-in tags.

Common Services

Mail Servers	`<cfmail>,<cfimap>, <cfpop>`
Web Servers	`<cfhttp> (support SSL)`
FTP Servers	`<cfftp> (support FTPS and SFTP)`
Directory Servers	`<cfldap>,<cfntauthenticate>`

Microsoft Servers

Exchange	<cfexchangeconnection>, <cfexchangecalender>, <cfexchangeccontact>, <cfexchangefilter>, <cfexchangetask>, <cfexchangemail>
SharePoint	<cfsharepoint>

Documents

PDF	<cfdocument>, <cfpdf>
Office Documents	<cfdocument>, <cfspreadsheet>, <cfpresentation>

Languages

Using the tag <cfobject> or the function CreateObject() you can create completely accessible objects for the Java, or .Net. For example you can pass info from .Net to Java in two lines of code:

```
<cfset pwd = CreateObject(".Net", "System.IO.Directory").
getCurrentDirectory() />
<cfset jStr = CreateObject("java", "java.lang.String").init(pwd)
/>
```

In addition, the object tools allow you to interface COM and CORBA.

SERVICES

A large part of current Web development is programming to and against service APIs. ColdFusion has a number of built in tools to consume and produce services using either SOAP or REST.

Consuming

SOAP

ColdFusion is able to consume SOAP Web services, by using either the <cfinvoke> or <cfobject> tags. You simply point them at the WDSL of the service.

Assume a Web service that provides nuggets of data with a method browse that takes a parameter named age. The idea is to give me the content that is age days old. It has a wsdl URL of http://forta.com/cf/tips/syndicate.cfc?wsdl. Here's how I would use it:

```
<cfset wsURL="http://forta.com/cf/tips/syndicate.cfc?wsdl" />
<cfinvoke webservice="#wsURL#"
method="browse"
returnvariable="result">
<cfinvokeargument name="age" value="0" />
</cfinvoke>
<cfdump var="#result#">
```

This yields figure 3.

struct	
AGE	2387
BODY	Here's another tip to improve Dreamweaver load time. In Preferences select the File Types / Editors screen and add .cfm and .cfc to the Open in Code View list. You'll lose support for design view and the Insert bar, but the load speed will be much faster.
DATE	{ts '2003-01-02 00:00:00'}
EXPIRES	570
HEADER	Faster Dreamweaver Loading
NEXT	false
PREV	true
PRODUCT	Dreamweaver
PRODUCT_VERSION	MX
TITLE	Ben Forta's ColdFusion Tip-of-the-Day
URL	http://www.forta.com/cf/tips/

REST

REST Web services can be consumed by using <cfhttp>.

```
<cfset restURL = "http://search.twitter.com/search.json" />
<cfset searchTerm = "ColdFusion" />
<cfhttp url="#restURL#?q=#searchTerm#" result="response" />
```

The result will be a ColdFusion struct with a number of keys. The json response will be the key fileContent. It will be in the json format and will have to be parsed into a ColdFusion XML variable:

```
<cfset serverResponse = "#deserializeJSON(response.fileContent)#"
/>
<cfdump var="#serverResponse#">
```

This yields figure 4.

struct	
completed_in	0.018972
max_id	2893712938
page	1
query	ColdFusion
refresh_url	?since_id=2893712938&q=ColdFusion
results	array
1	struct
created_at	Tue, 28 Jul 2009 17:40:20 +0000
from_user	TwelioNow
from_user_id	30833170
id	2893712938
iso_language_code	en
profile_image_url	http://s3.amazonaws.com/twitter_production /profile_images/329365228/penguin_normal.png
source	API
text	Mid-to-Senior Level Coldfusion Developer (Boston) | Web Design (URL: http://twelio.com /dz2p7h - TwitPic: http://twelio.com/77acn2)
to_user_id	null
results_per_page	15
since_id	2893712930

<cfhttp> supports the HTTP verbs that usually get used with REST: get, put, post, delete.

Formats

ColdFusion can parse the various formats that come back from Web services:

XML	<cfxml>, XmlParse()
JSON	DeserializeJSON()
RSS/Atom	cfxml>, <cffeed>, XmlParse()

Providing

SOAP

Any CFC can be turned into a SOAP Web service by adding access="remote" to a method:

```
<cfcomponent>
<cffunction name="now" access="remote" returntype="date">
<cfreturn Now() />
</cffunction>
</cfcomponent>
```

ColdFusion will autogenerate the wsdl for you if you append "?wsdl" to the URL of the CFC: http://localhost/ws.cfc?wsdl

REST

CFCs with remote methods can also be called as REST service. To call the Web service created in the preceding SOAP section you just append the method to the URL:

http://localhost/ws.cfc?method=now

However this might not strike some people as RESTful enough.

Instead you should write an intermediate page between the CFC and the caller that uses the HTTP verb to determine the method to take. This can be done either by introspecting the CGI scope or using the `GetHTTPHeaders()` function.

AJAX

While not a protocol per se, ColdFusion has a tag that allows remote methods to be called from JavaScript. `<cfajaxproxy>` create a JavaScript proxy for a specified CFC and call methods on it from JavaScript as a JavaScript object. Using the Web service created in the preceding SOAP we can call it and pass the value to alert:

```
<cfajaxproxy cfc="ws" jsclassname="ws" />
<script type="text/javascript">
wsInstance = new ws();
alert(wsInstance.now());
</script>
```

Format

The returnformat of any remote method can be configured to a couple of options:

plain	Just the output, assuming it is a simple variable type.
JSON	Serializes the results as JSON
RSS/Atom	Serializes the results as WDDX

AMF

Finally, any remotely exposed service is also available as a Flash Remoting or AMF service. This means that it can be consumed in Adobe Flex using the `RemoteObject` interface.

Adobe ColdFusion Site (Download and Licensing Information)	http://adobe.com/go/coldfusion
ColdFusion Developer Center (Articles and Tutorials)	http://adobe.com/devnet/coldfusion/
ColdFusion Bloggers Aggregator (ColdFusion Community)	http://coldfusionbloggers.org/
RIAForge (Open Source ColdFusion Projects)	http://riaforge.com
ColdFusion on Twitter (Announcements)	http://twitter.com/coldfusion

ABOUT THE AUTHOR

Terry Ryan is currently an Adobe Platform Evangelist for ColdFusion. His job is to drum up support and excitement among developers for Adobe ColdFusion. He has been working with it for over 10 years. He's presented at various Adobe ColdFusion events including cf.Objective, webDU, and Adobe Max. Prior to joining Adobe, Terry worked for the Wharton School of Business at the University of Pennsylvania in various roles around ColdFusion from Application Developer to System Administrator.

Terrence Ryan | Adobe Platform Evangelist | http://terrenceryan.com

RECOMMENDED BOOK

This Getting Started volume starts with Web and Internet fundamentals and database concepts and design, and then progresses to topics including creating data-driven pages, building complete applications, implementing security mechanisms, integrating with e-mail, building reusable functions and components, generating data-driven reports and graphs, building Ajax-powered user interfaces, and much more.

BUY NOW
books.dzone.com/books/adobe-coldfusion-8-web

DZone, Inc.
1251 NW Maynard
Cary, NC 27513

888.678.0399
919.678.0300

Refcardz Feedback Welcome
refcardz@dzone.com

Sponsorship Opportunities
sales@dzone.com

ISBN-13: 978-1-934238-87-5
ISBN-10: 1-934238-87-2
50795
9 781934 238875
$7.95