



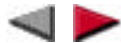
About Cold Fusion Graphlets

Using The Cold Fusion 3.1 Graphlets

The Cold Fusion 3.1 Graphlets are a rich set of graphing applets that use the Allaire Database Component Framework (DCF), built with Sun's Java[®] programming environment. The Graphlet charts include: pie, bar, multibar, area, line, and 3D multibar. With these Graphlets you can add advanced, database-driven graphing to your Cold Fusion applications today.

Contents

- [Bar Chart and Pie Chart](#)
- [Dynamic Population of Graph Parameters](#)
- [Example BarChart applet](#)
- [MultiBar, 3D-MultiBar, Area and Line Charts](#)
- [Example MultiBarChart3D Applet](#)





About Cold Fusion Graphlets

◀▶ Bar Chart and Pie Chart

The PieChart and BarChart applets allow you to graphically represent static or dynamic data on your HTML pages. Both applets require the following parameters in the <APPLET> tag: CODE, CODEBASE, WIDTH, and HEIGHT.

To add a pie chart to a Cold Fusion template, use the applet tag:

```
<APPLET CODE="PieChart.class"
        CODEBASE="/classes/CFGraphs/"
        WIDTH="width"
        HEIGHT="height">
    parameters
</APPLET>
```

To add a bar graph to a Cold Fusion template, use the applet tag:

```
<APPLET CODE="BarChart.class"
        CODEBASE="/classes/CFGraphs/"
        WIDTH="width"
        HEIGHT="height">
    parameters
</APPLET>
```

In both cases, width and height specify the size of the chart in screen pixels.

Applet parameters

To define the graph, you must specify the graph's items and their values with three parameter tags. In addition to these settings, there are a number of parameters that let you define other attributes of the graph, including characteristics such as the title, border, legend, and background color. Finally, there are a set of parameters that let you create a graph that is automatically refreshed at regular intervals. All of the parameters can be set with static values or you can use data retrieved

from a database by Cold Fusion with a <CFQUERY> or submitted by a user in a form.

Basic graph parameters

You must include three parameters which define each item (bar or slice) and its value:

```
<PARAM NAME="ChartData.Columns"
  VALUE="Items,Values">
<PARAM NAME="ChartData.Items"
  VALUE="ItemName1,ItemName2,...,ItemNameN">
<PARAM NAME="ChartData.Values"
  VALUE="ItemValue1,ItemValue2,...,ItemValueN">
```

ItemName1 is the name of the first item (e.g. , "January" for temperatures in January). ItemValue1 is the value of the first item (e.g., 60°).

Note: These applets represent non-numeric values as 0, and the PieChart applet draws negative values as positive.

Colors in the graph

The graphing applets will automatically choose colors for the items in the following order: red, blue, green, yellow, magenta, cyan, orange, pink, dark gray. Additional items will simply repeat the cycle in a darker shade. You can assign custom colors to each item by adding the ChartData.Colors parameter with a list of colors. You can use any of the nine colors listed above. The example below shows these parameters as set with static values:

```
<PARAM NAME="ChartData.Columns"
  VALUE="Items,Values,Colors">
<PARAM NAME="ChartData.Items"
  VALUE="January,February,March,April,May,June">
<PARAM NAME="ChartData.Values"
  VALUE="50.3,53,56.7,78,89.4,92.3">
<PARAM NAME="ChartData.Colors"
  VALUE="blue,green,yellow,orange,pink,red">
```





About Cold Fusion Graphlets

◀▶ Dynamic Population of Graph Parameters

You can use a <CFQUERY> to select data and then dynamically populate the values in the graph within a Cold Fusion template. For example, if you have a table in your database which stores information about average temperatures during the year, first you would query the table:

```
<CFQUERY NAME="GetClimate" DATASOURCE="Climate">
    SELECT Month, Temperature FROM Temperature
</CFQUERY>
```

You would then define the parameter values using variables from the query and the ValueList formatting function:

```
<CFOUTPUT>
<PARAM NAME="ChartData.Columns"
    VALUE="Items,Values,Colors">
<PARAM NAME="ChartData.Items"
    VALUE="#ValueList(GetClimate.Month)#">
<PARAM NAME="ChartData.Values"
    VALUE="#ValueList(GetClimate.Temperature)#">
<PARAM NAME="ChartData.Colors"
    VALUE="blue,green,yellow,orange,pink,red">
</CFOUTPUT>
```

Note: If your query returns more items than the number of colors you defined, all of the additional items will be black. It is better to avoid using a color list for queries that return a random number of items. You can also store the colors in the table as a separate column.

Optional parameters

Below are a list of parameters you can use to customize the graphs created by the PieChart and BarChart applets:

Option	Description	Default Value	Note
Title	Chart title	Chart	.
TitleFontName	Font for the chart title	Times Roman	.
TitleFontHeight	Font size for the chart title	12	.
ShowLegend	Show a legend for the chart	yes	PieChart only
LegendFontName	Font for the legend descriptions	Times Roman	.
LegendFontHeight	Font size for the legend descriptions	10	.
DrawBorders	Outlines the slices of bars with the font color	no	.
ShowDateTime	Date/time below the chart title	yes	.
Orientation	Orientation of the bars (horizontal/vertical)	vertical	BarChart only
Shadow	Number of pixels for the shadow	4	BarChart only
BackgroundColor	Background color	c0c0c0	.
FontColor	Font and borders color	000000	.
GridLineColor	Grid line and axes descriptions color	808080	.

Using any of these additional parameters is straightforward. For example, if you want the title of your chart to be "Average Temperatures on Coral Islands" and you want to use the Arial font with size 9 for the legend, your parameter list would include:

```
<PARAM NAME="Title"
  VALUE="Average Temperatures on Coral Islands">
<PARAM NAME="TitleFontName"
  VALUE="Arial">
<PARAM NAME="TitleFontHeight"
  VALUE="9">
```

Data refresh

With two additional parameters you can set the graphs to refresh the data shown in the chart with a new set of data at regular intervals. The RefreshTime parameter is the time in seconds when the chart should be updated with data defined in a template which should be specified in the RefreshDataFromURL parameter. By default the refresh time is 0 (the data is never refreshed). If you are using a Cold Fusion template called monitor.cfm to refresh the data and the refresh time is 10 seconds, your parameter definition should look like this:

```
<PARAM NAME="RefreshTime" VALUE="10">
<PARAM NAME="RefreshDataFromURL"
  VALUE="http://server_name/your_mapping/monitor.cfm">
```

The monitor.cfm template should only contain the query definition and the query output:

```
<!-- Beginning of Monitor.cfm template -->
<!-- Query to get data -->
<CFQUERY NAME="GetCurrentTemp" DATASOURCE="Climate">

  SELECT City, Temperature
  FROM CurrentTemperatures

</CFQUERY>

Columns: Items, Values

<CFOUTPUT QUERY="GetCurrentTemp">
  #City#, #Temperature#
</CFOUTPUT>
```



About Cold Fusion Graphlets

Example BarChart applet

This code is an example (/cfdocs/examples/cfgraphs/SalesByContinent.cfm) of the complete applet tag for a Bar Chart:

```
<!--- Display The Graph --->
<APPLET CODE="BarChart.class"
  CODEBASE="/classes/CFGraphs/"
  WIDTH="450"
  HEIGHT="250">

<!--- Set required record set parameters --->

  <PARAM NAME="ChartData.Columns" VALUE="Items,Values,Colors">
  <PARAM NAME="ChartData.Items"
    VALUE="#ValueList(GetSales.Continent)#">
  <PARAM NAME="ChartData.Values"
    VALUE="#ValueList(GetSales.TotalSalesOnContinent)#">
  <PARAM NAME="ChartData.Colors"
    VALUE="black,yellow,blue,red,green">

<!--- Set the optional display parameters --->
<PARAM NAME="Title" VALUE="Sales By Continent">
<PARAM NAME="TitleFontName" VALUE="Arial">
<PARAM NAME="TitleFontHeight" VALUE="9">

<!--- Set the optional refresh parameters --->
<PARAM NAME="RefreshTime" VALUE="10">
```



About Cold Fusion Graphlets

◀▶ MultiBar, 3D-MultiBar, Area and Line Charts

The MultiBar Chart, 3D-MultiBar Chart, Area Chart and Line Chart applets are similar to the Bar Chart and the Pie Chart applets, but they can represent more complex data. The class names for these charts are, respectively:

- MultiBarChart.class
- MultiBarChart3D.class
- AreaChart.class
- LineChart.class

You can use the same format as you would in the bar or pie charts for creating the applet tag, but there are some differences in the parameters.

Applet parameters

These record set parameters pass the data that will be represented in the chart:

```
<PARAM NAME="ChartData.Columns"
  VALUE="Groups,Items,Values">
<PARAM NAME="ChartData.Groups"
  VALUE="GroupName1,...,
  GroupName1,GroupName2,...,
  GroupName2,...,GroupNameM,...,GroupNameM">
<PARAM NAME="ChartData.Items"
  VALUE="ItemName1,...,
  ItemNameN,ItemName1,..., ItemNameN,...,
  ItemName1,...,ItemNameN">
<PARAM NAME="ChartData.Values"
  VALUE="ValueOfItem1InGroup1,...,ValueOfItemNInGroup1,&iquest;
  ValueOfItem1InGroup2,...,ValueOfItemNInGroup2,...,&iquest;
  ValueOfItem1InGroupM,...,ValueOfItemNInGroupM">
```

In these parameters, GroupName1 is the name of the first group, ItemName1 is the name of the first item and ValueOfItem1InGroup1 is its value in the first group (applets represent non-numeric values as 0 and negative values as positive).

Example for record set parameters

```
<PARAM NAME="ChartData.Columns"
  VALUE="Groups,Items,Values">
<PARAM NAME="ChartData.Groups"
  VALUE="USA,USA,USA,Canada,Canada,
  Mexico,Mexico,Mexico">
<PARAM NAME="ChartData.Items"
  VALUE="Blend1,Blend2,Blend3,Blend1,
  Blend2, Blend1,Blend3,Blend2 ">
<PARAM NAME="ChartData.Values"
  VALUE="50.3,20.4,3.2,53.6,22,56.8,2.3,25.8">
```

It is not necessary to have all the group-item combinations. As you can see in the example, there are no sales of Blend3 in Canada. The items in the groups don't have to be in the same order in all groups (e.g., Blend2 and Blend3 in Mexico), however, it is necessary that all of the items in a group be together.

The applet automatically chooses colors for items (in the order red, blue, green, yellow, magenta, cyan, orange, pink, and dark gray). If you are using more than nine items, the tenth is darker red, the eleventh is darker blue, etc. Unlike the PieChart and BarChart applets, you cannot assign custom colors to the items in the charts.

Dynamic population of graph parameters

As with the other Graphlets, you can use a Cold Fusion template to generate the record set for a graph. Using the previous example, imagine you have a CoffeeSales table that stores information about revenues from sales of coffee blends by country. The query for retrieving the data from this table will look like this:

```
<CFQUERY NAME="GetSales" DATASOURCE="YourDSN">
  SELECT Country, Blend, SUM(Revenues)
  FROM CoffeeSales
  GROUP BY Country, Blend
</CFQUERY>
```

The parameters definition will have the following form:

```
<CFOUTPUT>
<PARAM NAME="ChartData.Columns"
  VALUE="Groups,Items,Values">
<PARAM NAME="ChartData.Groups"
  VALUE="#ValueList(GetSales.Country)#">
<PARAM NAME="ChartData.Items"
```

```

    VALUE="#ValueList(GetSales.Blend)#">
<PARAM NAME="ChartData.Values"
    VALUE="#ValueList(GetSales.Revenues)#">
</CFOUTPUT>

```

Optional parameters

The MultiBar Chart, 3D-MultiBar Chart, Area Chart and Line Chart applets use the same set of optional parameters for chart customization as the Pie Chart and Bar Chart applets with several additional parameters.

Parameter	Description	Default Value	Note
Cumulative	Cumulate the data	no	AreaChart and LineChart only
DotSize	Size of the dots	2	LineChart only
Rotation	Rotation angle (0-90)	10	MultiBarChart3D only
Elevation	Elevation angle (0-90)	30	MultiBarChart3D only
Distance	Viewing distance in graph widths	3	MultiBarChart3D only

Data refresh

Like the other Graphlets, you can set these to refresh automatically. If you want to periodically refresh the data shown in the charts with a new set of data ,you can use two optional parameters: RefreshTime and RefreshDataFromURL. The URL must refer to a template or a static HTML page that returns the data in the following format:

```

Columns:Groups,Items,Values
GroupName1,ItemName1,ValueOfItem1InGroup1
...
GroupName1,ItemNameN,ValueOfItemNInGroup1
GroupName2,ItemName1,ValueOfItem1InGroup2
...
GroupName2,ItemNameN,ValueOfItemNInGroup2
...
GroupNameM,ItemName1,ValueOfItem1InGroupM
...

```

GroupNameM, ItemNameN, ValueOfItemNInGroupM

For example, you can create a Cold Fusion template called SalesMonitor.cfm to provide fresh data at regular 10 second intervals. In the applet you would include the following additional parameters:

```
<PARAM NAME="RefreshTime" VALUE="10">
<PARAM NAME="RefreshDataFromURL"
  VALUE="http://server_name/your_mapping/SalesMonitor.cfm">
```

In the SalesMonitor.cfm Cold Fusion template, you would simply include the query definition and the query output that would return the data in the format described above:

```
<!-- SalesMonitor.cfm template -->
<CFQUERY NAME="GetSales" DATASOURCE="YourDSN">
  SELECT Country, Blend, Revenues
  FROM CoffeeSales
  GROUP BY Country, Blend
</CFQUERY>
```

Columns:Groups,Items,Values

```
<CFOUTPUT QUERY="GetSales">
  #Country#,#Blend#,#Revenues#
</CFOUTPUT>
```



About Cold Fusion Graphlets

◀▶ Example MultiBarChart3D Applet

This code is an example of the complete applet tag for a 3D-MultiBar Chart (/cfdocs/examples/cfgraphs/salesbycontinent.cfm):

```
<HTML><HEAD>
  <TITLE>Sales By Continent</TITLE>
</HEAD><BODY>

<!-- Query the database to get data -->
<CFQUERY NAME="GetSales" DATASOURCE="Coffee Valley">
  SELECT C.Continent, Sum(S.Sales) AS TotalSalesOnContinent
  FROM Sales S, Continents C
  WHERE S.Continent = C.Continent_ID
  GROUP BY C.Continent
</CFQUERY>

<!-- Display The Graph -->
<APPLET CODE="MultiBarChart3D.class"
  CODEBASE="/classes/CFGraphs/"
  WIDTH="450"
  HEIGHT="250">

<!-- Set required record set parameters -->
<CFOUTPUT>
  <PARAM NAME="ChartData.Columns" VALUE="Items,Values,Colors">
  <PARAM NAME="ChartData.Items"
    VALUE="#ValueList(GetSales.Continent)#">
  <PARAM NAME="ChartData.Values"
    VALUE="#ValueList(GetSales.TotalSalesOnContinent)#">
  <PARAM NAME="ChartData.Colors" VALUE="black,yellow,blue,red,green">
</CFOUTPUT>

<!-- Set the optional display parameters -->
<PARAM NAME="Title" VALUE="Sales By Continent">
<PARAM NAME="TitleFontName" VALUE="Arial">
<PARAM NAME="TitleFontHeight" VALUE="9">
<PARAM NAME="DebugInfoEnabled" VALUE="yes">

<!-- Set the optional refresh parameters -->
<PARAM NAME="RefreshTime" VALUE="10">

<CFIF ParameterExists(CGI.HTTP_HOST)>
  <CFSET host = CGI.HTTP_HOST>
<CFELSE>
  <CFSET host = CGI.HOST>
</CFIF>

<CFOUTPUT>
  <PARAM NAME="RefreshDataFromURL"
    VALUE="http://#host#/cfdocs/examples/cfgraphs/
    salesbycontinent_data.cfm">
</CFOUTPUT>

<!-- If the Browser does not support Java show the following information -->
<H1> Your Browser Does Not Support JAVA!</H1>

</APPLET>
</CENTER>

<HR>

<FONT SIZE="-1">
If refresh doesn't work:
<LI>be sure that Debug Output in the CF Admin is switched off for your IP address
<LI>be sure that the HTTP server in the URL is specified as <B><CFOUTPUT>#host#</CFOUTPUT></B>
<LI>reload the page
</FONT>

</BODY>
```

ColdFusion 3.1 Graphlet Parameters – Quick Reference

Single Barchart

```
<APPLET CODE="BarChart.class" WIDTH="width" HEIGHT="height">
<!-- Data-driven parameters -->
<PARAM NAME="ChartData.Columns"
  VALUE="Items,Values,Colors">
<PARAM NAME="ChartData.Items"
  VALUE="XItem1,XItem2,XItem3,XItem4,XItem5,XItem6[...]">
<PARAM NAME="ChartData.Values"
  VALUE="YItem1,YItem2,YItem3,YItem4,YItem5,YItem6[...]">
<PARAM NAME="ChartData.Colors"
  VALUE="Color1,Color2,Color3,Color4,Color5,Color6[...]">
<!-- Optional parameters -->
<PARAM NAME="Title" VALUE="DefaultEqualstheword_Chart">
<PARAM NAME="TitleFontName" VALUE="DefaultEquals_TimesRoman">
<PARAM NAME="TitleFontHeight" VALUE="DefaultEquals_12">
<PARAM NAME="LegendFontName" VALUE="DefaultEquals_TimesRoman">
<PARAM NAME="LegendFontHeight" VALUE="DefaultEquals_10">
<PARAM NAME="DrawBorders" VALUE="DefaultEquals_No">
<PARAM NAME="ShowDateTime" VALUE="DefaultEquals_No">
<PARAM NAME="Orientation" VALUE="DefaultEquals_vertical">
<PARAM NAME="Shadow" VALUE="DefaultPixelWidthEquals_4">
<PARAM NAME="BackgroundColor" VALUE="DefaultEquals_C0C0C0">
<PARAM NAME="FontColor" VALUE="DefaultEquals_000000">
<PARAM NAME="GridLineColor" VALUE="DefaultEquals_808080">
</APPLET>
```

Where:

XItem values = each tick on the X-axis

YItem values = each data value on the Y-axis

Color values = a color for each data value. Possible colors for each value are:

red, blue, green, yellow, magenta, cyan, orange, pink, dark gray

If no colors are specified, the applet will choose the colors automatically in the order above.

ColdFusion 3.1 Graphlet Parameters – Quick Reference

Piechart

```
<APPLET CODE="PieChart.class" WIDTH="width" HEIGHT="height">
<!-- Data-driven parameters -->
<PARAM NAME="ChartData.Columns"
  VALUE="Items,Values,Colors">
<PARAM NAME="ChartData.Items"
  VALUE="XItem1,XItem2,XItem3,XItem4,XItem5,XItem6[...]">
<PARAM NAME="ChartData.Values"
  VALUE="YItem1,YItem2,YItem3,YItem4,YItem5,YItem6[...]">
<PARAM NAME="ChartData.Colors"
  VALUE="Color1,Color2,Color3,Color4,Color5,Color6[...]">
<!-- Optional parameters -->
<PARAM NAME="Title" VALUE="DefaultEqualstheWord_Chart">
<PARAM NAME="TitleFontName" VALUE="DefaultEquals_TimesRoman">
<PARAM NAME="TitleFontHeight" VALUE="DefaultEquals_12">
<PARAM NAME="ShowLegend" VALUE="DefaultEquals_Yes">
<PARAM NAME="LegendFontName" VALUE="DefaultEquals_TimesRoman">
<PARAM NAME="LegendFontHeight" VALUE="DefaultEquals_10">
<PARAM NAME="DrawBorders" VALUE="DefaultEquals_No">
<PARAM NAME="ShowDateTime" VALUE="DefaultEquals_No">
<PARAM NAME="BackgroundColor" VALUE="DefaultEquals_C0C0C0">
<PARAM NAME="FontColor" VALUE="DefaultEquals_000000">
<PARAM NAME="GridLineColor" VALUE="DefaultEquals_808080">
<PARAM NAME="RefreshTime" VALUE="ColdFusionTemplateRequiredQV">
<PARAM NAME="RefreshDataFromURL" VALUE="ColdFusionTemplateRequiredQV">
</APPLET>
```

Where:

XItem values = each tick on the X-axis

YItem values = each data value on the Y-axis

Color values = a color for each data value. Possible colors for each value are:

red, blue, green, yellow, magenta, cyan, orange, pink, dark gray

If no colors are specified, the applet will choose the colors automatically in the order above.

ColdFusion 3.1 Graphlet Parameters – Quick Reference

Area Chart

```
<APPLET CODE="AreaChart.class" WIDTH="width" HEIGHT="height">
<!-- Data-driven parameters -->
<PARAM NAME="ChartData.Columns"
  VALUE="Groups,Items,Values,Colors">
<PARAM NAME="ChartData.Groups"
  VALUE="Group1,Group2,Group3,Group4,Group5,Group6[...]">
<PARAM NAME="ChartData.Items"
  VALUE="XItem1,XItem2,XItem3,XItem4,XItem5,XItem6[...]">
<PARAM NAME="ChartData.Values"
  VALUE="YItem1,YItem2,YItem3,YItem4,YItem5,YItem6[...]">
<PARAM NAME="ChartData.Colors"
  VALUE="Color1,Color2,Color3,Color4,Color5,Color6[...]">
<!-- Optional parameters -->
<PARAM NAME="Title" VALUE="DefaultEqualstheWord_Chart">
<PARAM NAME="TitleFontName" VALUE="DefaultEquals_TimesRoman">
<PARAM NAME="TitleFontHeight" VALUE="DefaultEquals_12">
<PARAM NAME="ShowLegend" VALUE="DefaultEquals_Yes">
<PARAM NAME="LegendFontName" VALUE="DefaultEquals_TimesRoman">
<PARAM NAME="LegendFontHeight" VALUE="DefaultEquals_10">
<PARAM NAME="DrawBorders" VALUE="DefaultEquals_No">
<PARAM NAME="ShowDateTime" VALUE="DefaultEquals_No">
<PARAM NAME="BackgroundColor" VALUE="DefaultEquals_C0C0C0">
<PARAM NAME="FontColor" VALUE="DefaultEquals_000000">
<PARAM NAME="GridLineColor" VALUE="DefaultEquals_808080">
<PARAM NAME="RefreshTime" VALUE="ColdFusionTemplateRequiredQV">
<PARAM NAME="RefreshDataFromURL" VALUE="ColdFusionTemplateRequiredQV">
<PARAM NAME="Cumulative" VALUE="DefaultEquals_No">
</APPLET>
```

Where:

XItem values = each tick on the X-axis

YItem values = each data value on the Y-axis

Color values = a color for each data value. Possible colors for each value are:

red, blue, green, yellow, magenta, cyan, orange, pink, dark gray

If no colors are specified, the applet will choose the colors automatically in the order above.

ColdFusion 3.1 Graphlet Parameters – Quick Reference

Line Chart

```
<APPLET CODE="LineChart.class" WIDTH="width" HEIGHT="height">
  <!-- Data-driven parameters -->
  <PARAM NAME="ChartData.Columns"
    VALUE="Groups,Items,Values,Colors">
  <PARAM NAME="ChartData.Groups"
    VALUE="Group1,Group2,Group3,Group4,Group5,Group6[...]">
  <PARAM NAME="ChartData.Items"
    VALUE="XItem1,XItem2,XItem3,XItem4,XItem5,XItem6[...]">
  <PARAM NAME="ChartData.Values"
    VALUE="YItem1,YItem2,YItem3,YItem4,YItem5,YItem6[...]">
  <PARAM NAME="ChartData.Colors"
    VALUE="Color1,Color2,Color3,Color4,Color5,Color6[...]">
  <!-- Optional parameters -->
  <PARAM NAME="Title" VALUE="DefaultEqualstheWord_Chart">
  <PARAM NAME="TitleFontName" VALUE="DefaultEquals_TimesRoman">
  <PARAM NAME="TitleFontHeight" VALUE="DefaultEquals_12">
  <PARAM NAME="ShowLegend" VALUE="DefaultEquals_Yes">
  <PARAM NAME="LegendFontName" VALUE="DefaultEquals_TimesRoman">
  <PARAM NAME="LegendFontHeight" VALUE="DefaultEquals_10">
  <PARAM NAME="DrawBorders" VALUE="DefaultEquals_No">
  <PARAM NAME="ShowDateTime" VALUE="DefaultEquals_No">
  <PARAM NAME="BackgroundColor" VALUE="DefaultEquals_C0C0C0">
  <PARAM NAME="FontColor" VALUE="DefaultEquals_000000">
  <PARAM NAME="GridLineColor" VALUE="DefaultEquals_808080">
  <PARAM NAME="RefreshTime" VALUE="ColdFusionTemplateRequiredQV">
  <PARAM NAME="RefreshDataFromURL" VALUE="ColdFusionTemplateRequiredQV">
  <PARAM NAME="Cumulative" VALUE="DefaultEquals_No">
  <PARAM NAME="DotSize" VALUE="DefaultEquals_2">
</APPLET>
```

Where:

XItem values = each tick on the X-axis

YItem values = each data value on the Y-axis

Color values = a color for each data value. Possible colors for each value are:

red, blue, green, yellow, magenta, cyan, orange, pink, dark gray

If no colors are specified, the applet will choose the colors automatically in the order above.

ColdFusion 3.1 Graphlet Parameters – Quick Reference

MultiBarChart

```
<APPLET CODE=
  {MultiBarChart.class | MultiBarChart3D.class}" WIDTH="width" HEIGHT="height">
  <!-- Data-driven parameters -->
  <PARAM NAME="ChartData.Columns"
    VALUE="Groups,Items,Values,Colors">
  <PARAM NAME="ChartData.Groups"
    VALUE="Group1,Group2,Group3,Group4,Group5,Group6[...]">
  <PARAM NAME="ChartData.Items"
    VALUE="XItem1,XItem2,XItem3,XItem4,XItem5,XItem6[...]">
  <PARAM NAME="ChartData.Values"
    VALUE="YItem1,YItem2,YItem3,YItem4,YItem5,YItem6[...]">
  <PARAM NAME="ChartData.Colors"
    VALUE="Color1,Color2,Color3,Color4,Color5,Color6[...]">
  <!-- Optional parameters -->
  <PARAM NAME="Title" VALUE="DefaultEqualstheword_Chart">
  <PARAM NAME="TitleFontName" VALUE="DefaultEquals_TimesRoman">
  <PARAM NAME="TitleFontHeight" VALUE="DefaultEquals_12">
  <PARAM NAME="ShowLegend" VALUE="DefaultEquals_Yes">
  <PARAM NAME="LegendFontName" VALUE="DefaultEquals_TimesRoman">
  <PARAM NAME="LegendFontHeight" VALUE="DefaultEquals_10">
  <PARAM NAME="DrawBorders" VALUE="DefaultEquals_No">
  <PARAM NAME="ShowDateTime" VALUE="DefaultEquals_No">
  <PARAM NAME="BackgroundColor" VALUE="DefaultEquals_C0C0C0">
  <PARAM NAME="FontColor" VALUE="DefaultEquals_000000">
  <PARAM NAME="GridLineColor" VALUE="DefaultEquals_808080">
  <PARAM NAME="RefreshTime" VALUE="ColdFusionTemplateRequiredQV">
  <PARAM NAME="RefreshDataFromURL" VALUE="ColdFusionTemplateRequiredQV">
  <PARAM NAME="Rotation" VALUE="DefaultEquals_10">
  <PARAM NAME="Elevation" VALUE="DefaultEquals_30">
  <PARAM NAME="Distance" VALUE="DefaultEquals_3">

</APPLET>
```

Where:

XItem values = each tick on the X-axis

YItem values = each data value on the Y-axis

Color values = a color for each data value. Possible colors for each value are:

red, blue, green, yellow, magenta, cyan, orange, pink, dark gray

If no colors are specified, the applet will choose the colors automatically in the order above.

Developing Java Applets by Using DCF

You've seen how to use the DCF Graphlets that Allaire provides. But the DCF can also be used to Cold-Fusion-enable Java applets that you develop yourself. This may be more than you want to get into, since Java is a relatively complex development language; but if you're willing to invest the time and effort in learning Java, you can easily roll your own DCF applets.

Using the Java Language

It is beyond the scope of this book to introduce you to Java programming, which is comparable in syntax and complexity to C++. Instead, we will assume a little bit of Java knowledge in this section and provide a simple description of the Database Component Framework.

What you will need, then, to use this section is a Java development environment and a little background knowledge of Java. You can use the freeware Java Development Kit from Sun, available for download from <http://www.javasoft.com>. Or, you can use one of the many Java development environments available now, such as Symantec Café, Sun's Java Workshop, or Microsoft Visual J++.

Once you have this installed, you will also have to include the Allaire DCF classes within your classpath. The classpath is the path that the Java development environment uses to find classes included in the classes you compile. The DCF classes themselves are installed in the /Classes/CFGraphs/Allaire/ directory within your Web server's document tree. You can either include this directory in your classpath, or you can copy the Allaire directory to your existing Java library root directory. Check your Java development environment's documentation for more details on setting the classpath.

Building a Simple DCF Applet

For this example, we will build a very simple applet, possibly the simplest DCF applet you can make. Our applet will not perform any complex graphing, like the Allaire DCF Graphlets, but instead will display two fields, ISBN and NumberInStock, from our Inventory database. The ISBN number will be listed in a choice box, which is the Java name for a drop-down box; and when the user selects an entry from this choice box, the matching NumberInStock value will be displayed to the right by using a label control.

Most of Listing 28.9 is typical applet code to handle starting, stopping, and refreshing details. The part you want to read carefully is the retrieveData method, which uses the DCF classes to retrieve data from the template that calls the applet and from a refresh template, if one exists.

Listing 28.9 DCFDEMO.JAVA--Java Code for Our Simple DCF Applet

```
// Import the java classes for this applet
import java.awt.*;
import java.applet.Applet;
import java.awt.Graphics;
import java.awt.Color;
import java.awt.Font;
import java.awt.FontMetrics;
import java.io.*;
import java.lang.*;
import java.net.URL;
// Import the DCF classes
```

```

import allaire.dcf.recordset.*;
import allaire.util.*;
public class DCFDemo extends Applet implements Runnable {
    // This thread is used to run the applet
    Thread      runner;
    // These variables are for use with the DCF.
    // The Recordset and Query instances come
    // from the DCF classes of the same name.
    // The refreshTime and debugInfoEnabled
    // are used to set DCF-specific parameters.
    Recordset    formData;
    Query        query;
    int          refreshTime = 0;
    boolean      debugInfoEnabled = true;
    // These variables are used for the data
    // retrieved using the DCF.
    String[]     ISBN;
    String[]     numberInStock;
    int          columns;
    // These variables are used to display the data
    Label labelISBN;
    Label labelInStock;
    Choice choiceISBN;
    Label displayNumberInStock;
    // standard construction start, stop, run for applet
    // running as thread
    public void start() {
        if (runner == null) {
            runner = new Thread(this);
            runner.start();
        }
    }
    public void stop() {
        if (runner != null) {
            runner.stop();
            runner = null;
        }
    }
    public void run() {
        while (true) {
            if ( refreshTime == 0 ) {
                // stop thread if no refresh required
                stop();
                return;
            }
            pause(refreshTime * 1000);
            // refresh data and repaint applet
            this.retrieveData();
            repaint();
        }
    }
    private void pause(int time) {
        try { Thread.sleep(time); }
        catch (InterruptedException e) {
            if ( debugInfoEnabled ) Debug.write(e.getMessage());
        }
    }
    // The init method creates the screen layout,
    // and calls the data retrieval method.
    public synchronized void init() {

```

```

super.init();
setLayout(new FlowLayout());
resize(300,200);
labelISBN = new Label("ISBN:");
add(labelISBN);
labelISBN.reshape(10,10,50,20);
choiceISBN = new Choice();
add(choiceISBN);
labelInStock = new Label("Number In Stock:");
add(labelInStock);
displayNumberInStock = new Label();
add(displayNumberInStock);
// This section retrieves the refresh time and URL, if any.
// If they exist, the applet will append the refresh time to
// the query URL when it is called, so that you can read that
// in your refresh query template if you want.
String rs;
rs = getParameter("RefreshTime");
if (rs == null) {
    refreshTime = 0;
} else {
    refreshTime = Integer.parseInt(rs);
    rs = getParameter("RefreshDataFromURL");
    if (rs == null) {
        // if no URL selected - do not refresh
        refreshTime = 0;
    } else {
        // query definition
        query = new Query(rs);
        // add refreshTime parameter to the URL
        query.addParam( "RefreshTime", String.valueOf(refreshTime) );
    }
}
this.retrieveData();
}
// Here is the most important method in this applet.
// This method retrieves data from a Cold Fusion query
// using instances of the DCF Query and Recordset classes.
private void retrieveData() {
    try {
        // This decision tree either retrieves data from the
        // initial applet parameters, or retrieves it from a
        // refresh template.
        if ( formData == null ) {
            formData = new AppletParamRecordset(this);
        } else {
            query.execute();
            formData = query.getRecordset();
        }
        columns = formData.getRowCount();
        ISBN = new String[columns];
        numberInStock = new String[columns];
        // This for loop retrieves the data from the comma-delimited
        // lists in the template, and adds ISBN numbers to the choice
        // box.
        for (int i = 0; i < columns; i++) {
            ISBN[i] = formData.getData(i + 1, "ISBN");
            numberInStock[i] = formData.getData(i + 1, "NumberInStock");
            choiceISBN.addItem(ISBN[i]);
        }
    }
}

```

```

        displayNumberInStock.setText(numberInStock[0]);
    } catch (Exception e) {
        if (debugInfoEnabled) Debug.write(e.getMessage());
    }
}
// This event method is triggered by selecting an entry from the
// choice box.
public void selectedChoiceISBN() {
    int selectedISBNIndex = choiceISBN.getSelectedIndex();
    displayNumberInStock.setText(numberInStock[selectedISBNIndex]);
}
public boolean handleEvent(Event event) {
    if (event.id == Event.ACTION_EVENT && event.target == choiceISBN) {
        selectedChoiceISBN();
        return true;
    }
    return super.handleEvent(event);
}
}

```

First, we'll examine the `init` method, which is used to prepare the applet for display. In the case of a DCF applet, this also means retrieving parameters and initial data sets. Our simple applet is retrieving only two regular parameters, and it uses the standard `getParameter` method to do so. These parameters are `RefreshTime` and `RefreshDataFromURL`. If these parameters exist, they will be used for the query execution in the `retrieveData` method every time that method is called from the `run` method.

The part deserving most of your attention, the `retrieveData` method, is doing all the interesting stuff. The entire method is wrapped in a try/catch exception catcher, in case there is any problem retrieving data. Within the try portion, the first step is to determine whether the applet has previously read any data in, since this method is called every time the applet repaints, and then either read the initial data or execute the refresh query:

```

if (formData == null) {
    formData = new AppletParamRecordset(this);
} else {
    query.execute();
    formData = query.getRecordset();
}

```

The next step is to find out how many columns of data there are, and then create string arrays with enough space to hold all the columns:

```

columns = formData.getRowCount();
ISBN = new String[columns];
numberInStock = new String[columns];

```

Finally, in this method you retrieve the data from the recordset into the array variables, and then fill the choice box. The last line shown here initializes the display label with the value for the first ISBN number:

```

for (int i = 0; i < columns; i++) {
    ISBN[i] = formData.getData(i + 1, "ISBN");
    numberInStock[i] = formData.getData(i + 1, "NumberInStock");
    choiceISBN.addItem(ISBN[i]);
}

```

```
displayNumberInStock.setText(numberInStock[0]);
```

Listing 28.10 shows the template you'll use to call this applet. It uses a query to get the initial data for the applet.

Listing 28.10 DCFDEMO.CFM--This Template Calls and Initializes the Applet

```
<CFQUERY NAME="DCFDemo" DATASOURCE="A2Z Books">
    SELECT ISBN, NumberInStock FROM Inventory
</CFQUERY>
<HTML>
<HEAD>
<TITLE>DCF Demo</TITLE>
</HEAD>
<BODY>
<APPLET CODE="DCFDemo.class" CODEBASE="/Classes/CFGraphs/" WIDTH="300"
HEIGHT="300">
<PARAM NAME="columns" value="ISBN,NumberInStock">
<CFOUTPUT QUERY="DCFDemo">
<PARAM NAME="ISBN" VALUE="#ValueList(DCFDemo.ISBN)#">
<PARAM NAME="NumberInStock" VALUE="#ValueList(DCFDemo.NumberInStock)#">
</CFOUTPUT>
<PARAM NAME="RefreshTime" VALUE="0">
<PARAM NAME="RefreshDataFromURL"
VALUE="http://www.a2z.com/ch31/DCFDemo/refresh.cfm">
</APPLET>
</BODY>
</HTML>
```

The refresh template contains the same query as the previous template, but outputs just the data as explained in the Graphlet section.

This has been a very simple introduction to DCF programming. If you're interested in learning more, a good idea would be to DCF-enable some of the sample applets that come with the Java Development Kit. Listing 28.11 shows a DCF-enabled version of the Chart applet from the JDK.

Listing 28.11 DCFCHART.JAVA--A Sample DCF-Enabled Chart Applet

```
import java.awt.Graphics;
import java.awt.Color;
import java.awt.Font;
import java.awt.FontMetrics;
import java.io.*;
import java.lang.*;
import java.net.URL;
import allaire.dcf.recordset.*;
import allaire.util.*;
public class DCFChart extends java.applet.Applet implements Runnable {
    /*** the Runnable interface - applet can run as separate thread
    static final int    VERTICAL = 0;
    static final int    HORIZONTAL = 1;
    static final int    SOLID = 0;
    static final int    STRIPED = 1;
    int                orientation;
    String              title;
    Font                titleFont;
```

```

FontMetrics      titleFontMetrics;
int              titleHeight = 15;
int              columns;
int              values[];
Object           colors[];
Object           labels[];
int              styles[];
int              scale = 10;
int              maxLabelWidth = 0;
int              barWidth;
int              barSpacing = 10;
int              max = 0;
// DCF declarations for recordset and query
Recordset        chartData;
Query            query;
int              refreshTime;
boolean          debugInfoEnabled;
Thread           runner;
public synchronized void init() {
    // The return string variable used to get parameters
    String rs;
    // Get the title parameter
    titleFont = new java.awt.Font("Courier", Font.BOLD, 12);
    titleFontMetrics = getFontMetrics(titleFont);
    title = getParameter("title");
    if (title == null) {
        title = "Chart";
    }
    // Get the scale parameter
    rs = getParameter("scale");
    if (rs == null) {
        scale = 10;
    } else {
        scale = Integer.parseInt(rs);
    }
    // Get the orientation parameter
    rs = getParameter("orientation");
    if (rs == null) {
        orientation = VERTICAL;
    } else if (rs.toLowerCase().equals("vertical")) {
        orientation = VERTICAL;
    } else if (rs.toLowerCase().equals("horizontal")) {
        orientation = HORIZONTAL;
    } else {
        orientation = VERTICAL;
    }
    // Get the data refresh parameters
    rs = getParameter("RefreshTime");
    if (rs == null) {
        refreshTime = 0;
    } else {
        refreshTime = Integer.parseInt(rs);
        rs = getParameter("RefreshDataFromURL");
        if (rs == null) {
            // if no URL selected - do not refresh
            refreshTime = 0;
        } else {
            // query definition
            query = new Query(rs);
            // add refreshTime parameter to the URL

```

```

        query.addParam( "RefreshTime", String.valueOf(refreshTime) );
    }
}
// should the debug info window be enabled?
rs = getParameter("DebugInfoEnabled");
if (rs == null) {
    debugInfoEnabled = false;
} else if ( rs.toLowerCase().equals("yes") ) {
    debugInfoEnabled = true;
} else {
    debugInfoEnabled = false;
}
// retrieve graph data from params
this.getData();
}
// getData method retrieves data from applet params
// or from refresh template
private void getData() {
    int      i;
    String   rs;
    int      value;
    try {
        if ( chartData == null ) {
            // initial data come from the applet params ...
            chartData = new AppletParamRecordset(this, "ChartData");
        } else {
            // ... otherwise get data from refresh template
            query.execute();
            chartData = query.getRecordset();
        }
        // show info in the status bar
        if ( refreshTime == 0 ) {
            getAppletContext().setStatus("Done");
        } else {
            getAppletContext().setStatus("...");
        }
        columns = chartData.getRowCount();
// set dimension for the arrays
        values = new int[columns];
        colors = new Color[columns];
        labels = new String[columns];
        styles = new int[columns];
        // scroll thru records
        for ( i = 0; i < columns; i++) {
            // retrieve style type or set default if not defined
            if ( !chartData.columnExists("Styles") ) {
                styles[i] = (chartData.getData ( i + 1, "Labels").toLowerCase().
                    ? STRIPED : SOLID ;
            } else {
                styles[i] = SOLID;
            }
            // retrieve label
            labels[i] = chartData.getData ( i + 1, "Labels" );
            maxLabelWidth = Math.max(
                titleFontMetrics.stringWidth( chartData.getData ( i+1, "Labels"
                    maxLabelWidth);
            // retrieve value
            try {
                value = Integer.parseInt( chartData.getData ( i+1, "Values" )
            } catch (Exception e) {

```

```

        value = 0;
    }
    values[i] = value;
    max = Math.max( max, value);
    // retrieve color
    if ( !chartData.columnExists("Colors") ) {
        colors[i] = Color.black;
    } else {
        rs = chartData.getData ( i + 1, "Colors" );
        if ( rs.equals ( "red" ) ) {
            colors[i] = Color.red;
        } else if (rs.equals ( "green" ) ) {
            colors[i] = Color.green;
        } else if (rs.equals("blue")) {
            colors[i] = Color.blue;
        } else if (rs.equals("pink")) {
            colors[i] = Color.pink;
        } else if (rs.equals("orange")) {
            colors[i] = Color.orange;
        } else if (rs.equals("magenta")) {
            colors[i] = Color.magenta;
        } else if (rs.equals("cyan")) {
            colors[i] = Color.cyan;
        } else if (rs.equals("white")) {
            colors[i] = Color.white;
        } else if (rs.equals("yellow")) {
            colors[i] = Color.yellow;
        } else if (rs.equals("gray")) {
            colors[i] = Color.gray;
        } else if (rs.equals("darkGray")) {
            colors[i] = Color.darkGray;
        } else {
            colors[i] = Color.black;
        }
    }
}
} catch (Exception e) {
    if ( debugInfoEnabled ) Debug.write(e.getMessage());
}
switch (orientation) {
    case VERTICAL:
    default:
        barWidth = maxLabelWidth;
        resize(Math.max(columns * (barWidth + barSpacing),
            titleFontMetrics.stringWidth(title)) +
            titleFont.getSize() + 5,
            (max * scale) + (2 * titleFont.getSize()) + 5 + titleFont
        break;
    case HORIZONTAL:
        barWidth = titleFont.getSize();
        resize(Math.max((max * scale) + titleFontMetrics.stringWidth(" "
            titleFontMetrics.stringWidth(title)) + maxLabelW
            (columns * (barWidth + barSpacing)) + titleFont.getSize()
        break;
}
}
// standard construction start, stop, run for applet
// running as thread
public void start() {
    if (runner == null) {

```

```

        runner = new Thread(this);
        runner.start();
    }
}
public void stop() {
    if (runner != null) {
        runner.stop();
        runner = null;
    }
}
public void run() {
    while (true) {
        if ( refreshTime == 0 ) {
            // stop thread if no refresh required
            stop();
            return;
        }
        pause(refreshTime * 1000);
        // refresh data and repaint graph
        this.getData();
        repaint();
    }
}
private void pause(int time) {
    try { Thread.sleep(time); }
    catch (InterruptedException e) {
        if ( debugInfoEnabled ) Debug.write(e.getMessage());
    }
}
public synchronized void paint(Graphics g) {
    int i, j;
    int cx, cy;
    char l[] = new char[1];
    // draw the title centered at the bottom of the bar graph
    g.setColor(Color.black);
    i = titleFontMetrics.stringWidth(title);
    g.setFont(titleFont);
    g.drawString(title, Math.max((size().width - i)/2, 0),
        size().height - titleFontMetrics.getDescent());
    for (i=0; i < columns; i++) {
        switch (orientation) {
            case VERTICAL:
            default:
                // set the next X coordinate to account for the label
                // being wider than the bar size().width.
                cx = (Math.max((barWidth + barSpacing),maxLabelWidth) * i) +
                    barSpacing;
                // center the bar chart
                cx += Math.max((size().width - (columns *
                    (barWidth + (2 * barSpacing))))/2,0);
                // set the next Y coordinate to account for the size().height
                // of the bar as well as the title and labels painted
                // at the bottom of the chart.
                cy = size().height - (values[i] * scale) - 1 - (2 * titleFont.getSiz
                // draw the label
                g.setColor(Color.black);
                g.drawString((String)labels[i], cx,
                    size().height - titleFont.getSize() - titleFontMetrics.
                // draw the shadow bar
                if (colors[i] == Color.black) {

```

```

        g.setColor(Color.gray);
    }
    g.fillRect(cx + 5, cy - 3, barWidth, (values[i] * scale));
    // draw the bar with the specified color
    g.setColor((Color)(colors[i]));
    switch (styles[i]) {
        case SOLID:
        default:
            g.fillRect(cx, cy, barWidth, (values[i] * scale));
            break;
        case STRIPED:
            {
                int steps = (values[i] * scale) / 2;
                int ys;
                for (j=0; j < steps; j++) {
                    ys = cy + (2 * j);
                    g.drawLine(cx, ys, cx + barWidth, ys);
                }
            }
            break;
    }
    g.drawString("" + values[i],
                cx,
                cy - titleFontMetrics.getDescent());
    break;
case HORIZONTAL:
    // set the Y coordinate
    cy = ((barWidth + barSpacing) * i) + barSpacing;
    // set the X coordinate to be the size().width of the widest
    // label
    cx = maxLabelWidth + 1;
    cx += Math.max((size().width - (maxLabelWidth + 1 +
                                titleFontMetrics.stringWidth("" +
                                                                max) +
                                                                (max * scale))) / 2, 0);
    // draw the labels and the shadow
    g.setColor(Color.black);
    g.drawString((String)labels[i], cx - maxLabelWidth - 1,
                cy + titleFontMetrics.getAscent());
    if (colors[i] == Color.black) {
        g.setColor(Color.gray);
    }
    g.fillRect(cx + 3,
                cy + 5,
                (values[i] * scale),
                barWidth);
    // draw the bar in the current color
    g.setColor((Color)(colors[i]));
    switch (styles[i]) {
        case SOLID:
        default:
            g.fillRect(cx,
                cy,
                (values[i] * scale),
                barWidth);
            break;
        case STRIPED:
            {
                int steps = (values[i] * scale) / 2;
                int ys;

```

```

        for (j=0; j < steps; j++) {
            ys = cx + (2 * j);
            g.drawLine(ys, cy, ys, cy + barWidth);
        }
        break;
    }
    g.drawString(" " + values[i],
        cx + (values[i] * scale) + 3,
        cy + titleFontMetrics.getAscent());
    break;
}
}
}
}
}

```

DCF Class Descriptions

You'll need to know the details of the DCF classes, including their variables, constructors, and methods, if you want to use all the features of the DCF.

Class `allaire.dcf.recordset.Query`. The public class `Query` extends `java.lang.Object`. This class executes Cold Fusion templates on remote servers. The templates typically contain one or more SQL queries, which are returned in text format and then parsed into `Recordset` objects for use by the client object. The constructor for this class is

```
public Query(String strTemplateURL)
```

which creates a new query object that's linked to a template on a remote server. Constructing a `Query` object does not cause the `Query` to be executed (this is accomplished by using the `execute()` method). The `strTemplateURL` parameter is the fully qualified URL of the template to be used by the `Query`.

The `addParam` method,

```
public void addParam(String strName, String strValue)
```

adds a named parameter to the query, which will be URL-encoded and appended to the URL submitted to the remote server.

The `resetParams` method,

```
public void resetParams()
```

resets the parameter list to empty. This erases all parameters previously added by using the `addParam()` method.

The `execute` method,

```
public void execute() throws Exception
```

executes the template on the remote server by using the specified parameters. All `Recordsets` returned by the template will be parsed by using the `getRecordset` class methods.

The `getRecordset` method,

```
public Recordset getRecordset() throws Exception
```

or

```
public Recordset getRecordset(String strName) throws Exception
```

gets the Recordset, or one of a group of named Recordsets, fetched by the query from the remote template.

The `getRecordsets` method,

```
public Dictionary getRecordsets()
```

returns all Recordsets from the query.

Class `allaire.dcf.recordset.Recordset`. The public class `Recordset` extends `java.lang.Object`. This class represents comma-delimited data from templates as a set of rows and columns. Recordsets can be created from applet parameters or executed queries.

The constructor for this class is

```
public Recordset()
```

which creates a new empty `Recordset`. Usually, you will have already created a recordset by using the `Query` or `AppletParamRecordset` classes.

The `getRowCount` method,

```
public int getRowCount()
```

returns the number of rows in the `Recordset`.

The `getColumnNames` method,

```
public Vector getColumnNames()
```

returns the column names in the recordset.

The `columnExists` method,

```
public boolean columnExists(String strColumnName)
```

checks whether the column named in the parameter exists in the `Recordset`.

The `getData` method,

```
public String getData(int iRow, String strColumnName) throws Exception
```

returns the data value from the intersection of the row value and the column name.

The getColumnData method,

```
public Vector getColumnData(String strColumnName) throws Exception
```

returns all the data from the named column.

Class `allaire.dcf.recordset.AppletParamRecordset`. The public class `AppletParamRecordset` extends `Recordset`. This class is used to instantiate a `Recordset` from applet parameters.

This class can be constructed two ways:

```
public AppletParamRecordset(Applet applet) throws Exception
```

or

```
public AppletParamRecordset(Applet applet, String strName) throws Exception
```

The second allows you to specify multiple named `Recordsets` within your template, and refer to them by using separate `AppletParamRecordset` instances.

The methods for this class are all inherited from the `Recordset` class; there are none specific to this class.

ColdFusion 3.1 Graphlet Set - Setup

) The following .class files must exist in the directory where they are to be used:

AreaChart.class
BarChart.class
item.class
LineChart.class
MultiBarChart.class
multibarchart3d.class
PieChart.class

) The following .class files must exist in the following folder:

allaire\dcf\recordset

in the folder where the graphlets are to be used:

appletparamrecordset.class
query.class
queryerrorexception.class
Recordset.class

Java is case-sensitive, so they must be named exactly how they're shown. The .class files do not support placement into a JAR, nor do they support being placed in folders in the directory where they are to be called from, with the exception of the dcf files.

Any other configuration for the graphlet set will not work. Once a web page calling a graphlet works, however, if frames are not used then it will also work when called from a URL as well as a UNC.