crystal decisions™

Crystal Reports

Advanced Reporting Techniques using Arrays

Overview

This document is intended to assist you to understand, build, and manipulate Arrays in the Crystal Reports Designer and the Seagate Info Report Designer.

To better understand Arrays you should be familiar with manual running totals, and variable scopes. There are technical briefs written on both of these topics and can be downloaded from our website.

For more information on manual running totals search for Scr_running_total.zip on our support site at http://support.crystaldecisions.com/downloads.

For more information on variable scopes search for Scr_variablescopes.pdf on our support site at http://support.crystaldecisions.com/docs.

Contents

Introduction

In Crystal Reports (CR), Arrays are ordered lists of values that are all of the same data type. Data values in Arrays are also known as elements.

To create an Array in CR use the Array function that is available in CR formula editor. In CR, an Array that contains several elements can be used to manipulate data to allow for more advanced reporting options.

Sample Reports needed

Before you proceed reading this document, you will need to download the following sample reports from Crystal Decisions support site at:

http://support.crystaldecisions.com/downloads

The file name to search for is CR_Arrays_Samples.zip.

This Winzip file contains the reports listed in this document. These sample reports will help you better understand how Arrays work. Most of the sample code contained in this technical brief is based on these sample reports.

CR_Dynamic_Array_Builder_Crystal_Syntax.rpt	Illustrates how to dynamically assign values from a report to an Array.
CR_Pass_Values_From_Main_Report.rpt	Illustrates how to pass an Array created in a subreport to a main report and display the contents of that Array in the main report.
CR8_Looping_Through_Multiple_Value_Parameter.rpt	Illustrates how to loop through a multiple value parameter in the record selection formula.
CR_Manual_Running_Array.rpt	Illustrates how to use the elements in an Array to create a 'Sum' in a formula
CR_Index_not_store_and_fetch.rpt	Illustrates using Arrays to create an Index for a report.
CR_Multiple_Arrays_Dynamically_Populated.rpt	Illustrates how to use build multiple arrays when the elements exceed 1,000.

Details about Arrays

This section will describe the elements of Arrays, assigning values to Arrays, and using Arrays.

Limitations of Array Elements

In different versions of CR there are limitations to the amount of elements allowed in each Array.

- In CR version 6 the limit of elements allowed in the Array are 100 elements.

- In CR version 7 the limit of elements allowed in the Array are 250 elements.
- In CR version 8 the limit of elements allowed in the Array are 1000 elements.

Assigning values to Arrays

A simple string Array looks like the following:

[“car”, “bus”, “taxi”, “goat”, “boat”]

A simple numeric Array looks like the following:

[1, 2, 3, 4, 5]

In CR versions 6 and 7, you can only declare the Array in a formula and reference it. You cannot change the elements of an Array as you can in CR version 8.

For example:

```
//@MyArray
//assigning values to an Array in Crystal Reports version
//6, and 7
Numbervar Array MyArray := [1, 2, 3, 4, 5];
0
```

Create a new formula similar to the following to reference values of an Array:

```
//@ReferenceArray
//referencing the values of elements in the
//variable MyArray
whileprintingrecords;
Numbervar Array MyArray;
If {database.field} = condition1
       Then MyArray [2]
              Else MyArray [4]
```

This formula will only return values that meet the condition. For example, if the database field meets condition1 then the second element of the Array will display. Otherwise, the fourth element will display.

CR version 8 gives you the ability to assign values to an Array.

To assign values to elements in the Array, create a formula similar to the following:

```
//@ChangeArray
//changing the values of elements in the variable MyArray
whileprintingrecords;
Numbervar Array MyArray;
```

```
If {database.field} = condition1
Then MyArray [2] := 50
Else MyArray [4] := 83
```

This formula replaces elements in the Array based on the condition. Such as, if the database field equals the condition then the second element in the Array will change the value to 50. Otherwise, the fourth element will change to the value of 83.

Declaring Numeric Arrays

To declare a numeric Array in CR, create a formula similar to the following:

```
//@myArray
//declaring an Array with five elements
Numbervar Array MyArray := [1, 2, 3, 4, 5];
```

To assign a new value to one particular element of the Array in CR version 8, create a formula similar to the following:

```
//@myArray
//Assigning values to one element
Numbervar Array MyArray := [1, 2, 3, 4, 5];
MyArray [3] := 25
```

The element 3 was reassigned with the value 25.

Only elements that exist in the declared Array may be reassigned.

For example, you attempt to assign a sixth element value in an Array that only has five elements declared will result in error.

For example:

```
//@myArray
//Assigning a value to one element not declared
//in the Array
Numbervar Array MyArray := [1, 2, 3, 4, 5];
MyArray [6] := 25
```

You have not declared the Array to have 6 elements therefore you cannot assign it a value.

NOTE	To increase the number of elements within an Array, you can use either the Redim or Redim Preserve functions. These functions are available in the Crystal Reports formula editor using either Crystal Syntax or Basic Syntax.

Array Specific Functions

In CR version 8, you can increase the number of elements in an array after the array has been created and populated with data. You can choose to preserve or not to preserve the data that was previously assigned to positions within the array. Depending on what you want to do will determine the actual function to use. Also, there is a function that counts the total number of elements existing within the Array.

These functions are:

- Redim x[n]
- Redim Preserve x[n]
- Ubound [x]

Array Specific Functions in Detail

`Redim x[n]`

Using this function will re-dimension the array x to size n, where x is an array and n is a positive whole number specifying the new size of n.

For example:

```
//@AddElements
//creates additional elements in array without preserving
//previous values held
StringVar array x := ["a", "bb", "ccc"];
Redim x [4];
//now x = ["", "", "", ""]
```

The Array will now contain four elements, all of which are blank strings. The previous string values held within the original positions have all been reassigned the value of a blank string. This will not preserve the data that was previously assigned a position within the array.

`Redim Preserve x[n]`

Using this function will re-dimension the array x to size n, while preserving the initial values in x. The array is x and n is a positive whole number specifying the new size of n.

Example Formula:

```
//@AddElements
//creates additional elements in array while preserving
//previous values held
StringVar array x := ["a", "bb", "ccc"];
Redim Preserve x [4];
//now x = ["a", "bb", "ccc", ""]
```

The Array will now contain four elements. All elements previously populated maintained their original values while the new element has been assigned the default value for the Array's data type. This will preserve the data that was previously assigned a position within the array.

Ubound(x)

This function returns a number containing the largest available subscript for the given array.

For example:

```
//CountElements
NumberVar Array x;
Redim x[10];
UBound(x)
//Returns 10
```

This function is commonly used to obtain the size of an array before using a looping control mechanism to systematically manipulate elements of the array.

Using Basic Syntax to Create Arrays

When variables are declared using Basic Syntax, the structure of declaration and value assignment is slightly different than from using Crystal Syntax.

For Example:

```
//BasicSyntaxArray
//Formula assigns an additional position to the array while
//maintaining previous values stored in other positions
dim myarray(1) as number
counter = counter + 1
myarray = array(counter)
Redim preserve myarray(counter)
myarray(counter) = {Customer.Customer ID}
//Adds a value to the new position and displays it
formula = myarray(counter)
```

Using Arrays

In CR you are limited to using one value per variable declared. You are able to store up to 1000 separate values into a single Array. This allows you to pass many values at one time to a subreport. By passing values to a subreport, you can dynamically manipulate the values assigned to variables. In addition, this allows you to sum the elements in an Array within the subreport, after the array has been systematically manipulated.

Arrays

The section goes through each sample report in detail. The sample reports provided describe possible uses for Arrays. In this section, you will find details on how to:

- create an Array that loops through a multiple value parameter in the record selection formula.
- dynamically assign values from a report to an Array.
- pass an Array created in a subreport to a main report and display the contents of that Array in the main report.
- use the elements in an Array to create a 'sum' in a formula.
- use Arrays to create an index for a report.
- use Arrays to create a 'Top N' report that can be sorted by a field other than the one for 'Top N'.
- use Arrays to suppress blank subreports without generating any white space.

Using Arrays to eliminate errors

A multiple value parameter is stored as an Array. You have created a parameter that allows you to enter multiple discrete strings. If you enter spaces in the parameter unintentionally and then search for those values stored in the database, a complete record set will not bet returned. This is because the values stored in the database do not have spaces, and do not match your parameter entry.

Viewing your record selection, the formula generated is as follows:

```
{MyDatabase.Myfield} IN {?Parameter}
```

This formula is not allowing for spaces in the parameter name. Therefore, the report skips records in the database without spaces.

To return the complete record set, including the parameter entry containing a zero, you can modify the record selection formula to read similar to the following:

```
numbervar counter;
numbervar positionCount:=count({?test});
stringvar Array NewArray;
while positionCount >= counter do
(
counter:=counter + 1;
redim preserve NewArray[counter];
NewArray [counter] := trim({?test}[counter])
);
{Customer.Customer Name} in NewArray
```

Note	The code provided in this example is commented thoroughly in the sample report called CR8_Looping_Through_Multiple_Value_Parameter.rpt contained in the Winzip file, CR_Arrays_Samples.zip.

The purpose of this formula is to:

- Create a counter
- Create an endpoint for the loop
- Create a storage Array

The formula counts how many items have been entered into the multiple-value parameter. Then it loops through all of the selections entered in the parameter, trims each one, and assigns the trimmed entry to an element in the new Array. Now, the record selection is based on the contents of the New Array.

Note	If your database is large, and you create a record selection similar to this example, CR will be forced to bring back all of the records from the database and filter them on the client side. This can significantly impact the performance of your report.

Creating Arrays Dynamically

In order for you to successfully workaround building Arrays in CR you must be familiar with building Arrays dynamically. In other words, by dynamically building an Array you can change the value of elements contained in the Array to reflect the actual values contained in your database.

In order to dynamically create an Array, it is recommended that you are familiar with manual running totals; shared variables; control structures such as CASE statements, IF THEN ELSE statements, looping, etc.

Building an Array Dynamically

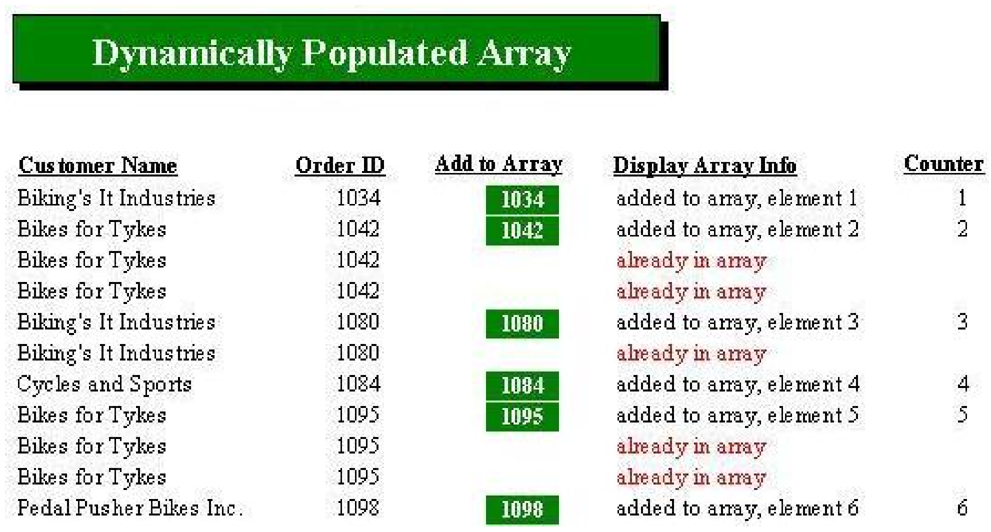
The following example formula verifies if a value of a database field is contained in the array. If the value is not contained in the array, the formula performs the following actions:

- The counter increments
- The size of the array increments
- The value of the database field is added to the array

Example Formula:

```
//@DynamicArrayBuilder

whileprintingrecords;

numbervar array MyArray;

numbervar Counter;

//The line below ensures that only new values are added to
//the array.

if not({MyDatabase.ValueForArray} in MyArray) then
```

```
//The line below is activated if the field value in not
//already contained in the array.  The counter increments
//by one.

(Counter := Counter + 1;

//The line below ensures that the size of the array does
//not exceed 1000 values.  An array can contain a maximum
//of 1000 values.

if Counter <= 1000

//Redim Preserve is function that changes the size of the
//array according to the new value of Counter.  Also, the
//existing values in the array are not removed or changed.
//The array now has space for a new value.

then (Redim Preserve MyArray[Counter];


//The new value is added to the newly created space in the
//array.

MyArray[Counter] := {MyDatabase.ValueForArray}));
```



Dynamically Populated Array

Customer Name	Order ID	Add to Array	Display Array Info	Counter
Biking's It Industries	1034	1034	added to array, element 1	1
Bikes for Tykes	1042	1042	added to array, element 2	2
Bikes for Tykes	1042		already in array	
Bikes for Tykes	1042		already in array	
Biking's It Industries	1080	1080	added to array, element 3	3
Biking's It Industries	1080		already in array	
Cycles and Sports	1084	1084	added to array, element 4	4
Bikes for Tykes	1095	1095	added to array, element 5	5
Bikes for Tykes	1095		already in array	
Bikes for Tykes	1095		already in array	
Pedal Pusher Bikes Inc.	1098	1098	added to array, element 6	6

Figure 1 - Only new values are added to the array. The number of values in the array is increased and the counter is incremented by one. The formula displays the new array values.

Note	The code provided in this example is commented thoroughly in the sample report called CR_Dynamic_Array_Builder_Crystal_Syntax.rpt contained in the Winzip file, CR_Arrays_Samples.zip.

Building Multiple Arrays

The following formula example illustrates how to build multiple arrays when the size of the array exceeds 1,000 elements and verifies if a value of a database field is contained in the any of the arrays. The formula performs the following actions:

- Dynamically resizes and populates the arrays
- Adds only new elements to the arrays
- Increments the size of the arrays
- Adds values to the next array if the current array reaches 1,000 elements
- Resets the arrays on the Group Footer level

Example Formula:

```
//@Array Generation, Crystal Syntax

whileprintingrecords;

//In this example, up to 3 arrays are populated
//dynamically.  If groups have more than 3000 unique
//records, more arrays can be added.

numbervar array idarrayCS;

numbervar array  idarrayCSII;

numbervar array  idarrayCSIII;

//rt1CS is a counter that will be used to assign the
//element number (array position) of a value in an array.

numbervar rtICS;

//The code below resets the array on the group level.
//Arrays are reset to have 1 element which equals 0...an
//array must have at least one element and this is why the
//array's size is set to 1 element.

if onfirstrecord or {Customer.Country} <>
previous({Customer.Country}) then

(redim idarrayCS[1];

redim idarrayCSII[1];

redim idarrayCSIII[1];

rtICS := 0);

//The code below verifies to see if the Order ID field is
//in any of the above-declared arrays.  If it is, it is not
//added again and rt1 is not incremented.

if not({Orders.Order ID} in idarrayCS or {Orders.Order ID}
in idarrayCSII or {Orders.Order ID} in idarrayCSIII) then

(

rtICS := rtICS + 1;

//Redim means that the size of the array (the number of
//elements) is changed to the value at the right.
//Preserve means that all elements in the array that are
//redimmed remain in the array.

if rtICS <= 1000 then (Redim Preserve idarrayCS[rtICS];
idarrayCS[rtICS] := {Orders.Order ID});

if rtICS in 1001 to 2000 then (Redim Preserve
idarrayCSII[rtICS-1000]; idarrayCSII[rtICS-1000] :=
{Orders.Order ID});

if rtICS in 2001 to 3000 then (Redim Preserve
idarrayCSIII[rtICS-2000];idarrayCSIII[rtICS-2000] :=
{Orders.Order ID});

);

if rtICS <= 1000 then idarrayCS[rtICS]

else if rtICS in 1001 to 2000 then idarrayCSII[rtICS-1000]

else if rtICS in 2001 to 3000 then idarrayCSIII[rtICS-2000]
```

Note	The code provided for this example is commented thoroughly in the sample report called CR_Multiple_Arrays_Dynamically_Populated.rpt contained in the Winzip file, CR_Arrays_Samples.zip. To view the code of the above formula in Crystal syntax or in Basic syntax, edit the applicable @Array Generation formula. Also, there is a manual running total formula, @Manual RT, in the Details section that indicates how many elements per group are added to all of the arrays.

Passing the Array from Main Report to Subreport

Many of the workarounds involve building Arrays in a subreport and calling them from the main report.

It is possible to share an Array just like you can share a variable. The following syntax enables you to share the Array:

```
//sharing the Array

Shared StringVar Array MyArray := ["Todd", "Jason", "Dirk",
"Jamie", "Aron"];
```

Note	The code provided in this example is commented thoroughly in the sample report called CR_Pass_Values_From_Main_Report.rpt contained in the Winzip file, CR_Arrays_Samples.zip.

Summing the Array

You can sum the elements of an Array without using a looping formula.

If you have a numeric Array with 10 elements and want to find out the sum of the first five elements, create a formula similar to the following:

```
//@SumFirstFiveElements

//summarizes the first five positions in the Array

Whileprintingrecords;

numbervar Array MyArray;

sum (MyArray[1 to 5])
```

For example, you want to see the total Sums of Last Year's Sales for various companies. You want to know how the top third, middle third, and bottom third performed last year. You have already dynamically built an Array and named it MyArray.

The following code allows you to view the top, middle, and bottom third of the Sum of Last Year's Sales for various companies:

```
{@FirstThirdTotal}
whileprintingrecords;
numbervar Array MyArray;
numbervar counter;
numbervar thirds;
thirds:=truncate((counter / 3), 0);
sum (MyArray[1 to thirds])

{@SecondThirdTotal}
whileprintingrecords;
numbervar Array MyArray;
numbervar counter;
numbervar thirds;
thirds:=truncate((counter / 3), 0);
sum (MyArray[(thirds + 1) to (thirds * 2)])


{@LastThirdTotal}
whileprintingrecords;
numbervar Array MyArray;
numbervar counter;
numbervar thirds;
thirds:=truncate((counter / 3), 0);
sum (MyArray[(thirds * 2 + 1) to counter])
```

Note	The code provided in this example is commented thoroughly in the sample report called CR_Manual_Running_Array.rpt contained in the Winzip file, CR_Arrays_Samples.zip.

Sorting the Array manually

If you want to sort the elements of an Array manually, then you would need to create a custom looping formula. To sort the elements of an Array within CR, you must use a double For Loop. Create a formula similar to the following:

Note	CR has a limit of 30,000 loop evaluations per formula. Only Arrays that have a maximum of 150 elements should be sorted using this method.

To sort the elements of a numeric Array, create a formula similar to the following and call it @Sort_Array:

1. Create a formula similar to the following:

```
// This formula tests to see if the element of the Array
//is greater than the next element.
//If the current element is greater than the next element,
//their positions are switched using another variable
//to temporarily store the value.
WhilePrintingRecords;
NumberVar Array Array_Name;
NumberVar counter1;
Numbervar counter2;
Numbervar temp;
Numbervar Array_Size := count(Array_Name);
for counter1 := Array_Size to 1 step -1  do
(
    for counter2 := 1 to counter1 - 1 do
    (
        if Array_Name[ counter2] >
             Array_Name[ counter2 + 1] then
        (
            temp := Array_Name[counter2];
            Array_Name[counter2] :=
             Array_Name[counter2 + 1];
            Array_Name[counter2 + 1] := temp;
        )
    );
);
```

2. Replace the Array_Name variable with the variable name of your Array.
3. Insert the formula into a section in your report. Ensure that you insert the formula in the section after the section that is used to populate your Array.

The same logic can be used to sort the elements of a string Array:

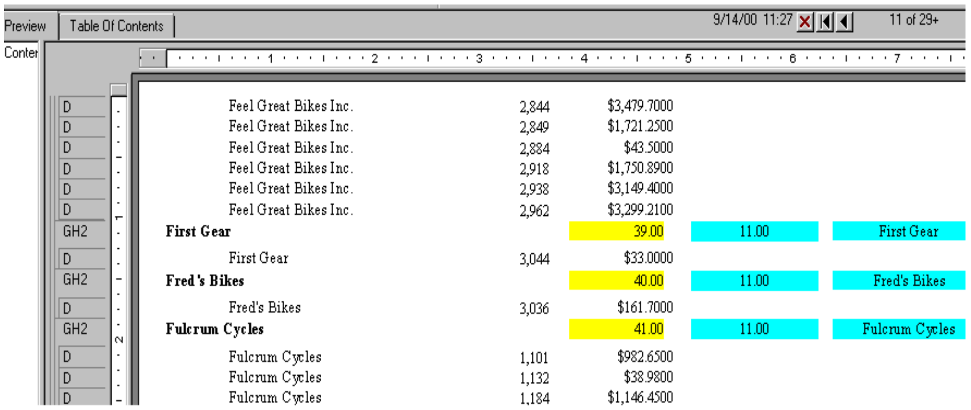
1. Create a new formula called @Sort_Array.
2. Cut and paste the following formula into the formula editor:

```
// This formula tests to see if the element of the Array
//is greater than the next element.
//If the current element is greater than the next element,
//their positions are switched using another variable
//to temporarily store the value.
WhilePrintingRecords;
Stringvar Array Array_Name;
NumberVar counter1;
Numbervar counter2;
Stringvar  temp;
Numbervar Array_Size := count(Array_Name);
for counter1 := Array_Size to 1 step -1  do
(
    for counter2 := 1 to counter1 - 1 do
    (
        if Strcmp( Array_Name[ counter2],
             Array_Name[ counter2 + 1]) > 0 then
        (
            temp := Array_Name[counter2];
            Array_Name[counter2] :=
             Array_Name[counter2 + 1];
            Array_Name[counter2 + 1] := temp;
        )
    );
);
```

3. Replace the Array_Name variable with the variable name of your Array.
4. Insert the formula into a section in your report. Ensure that you insert the formula in the section after the section, which is used to populate your Array.

Index Report

You can build an Index in CR using an Array. CR does not have any internal capabilities to create Indexes.



D	Feel Great Bikes Inc.	2,844	$3,479.7000			
D	Feel Great Bikes Inc.	2,849	$1,721.2500			
D	Feel Great Bikes Inc.	2,884	$43.5000			
D	Feel Great Bikes Inc.	2,918	$1,750.8900			
D	Feel Great Bikes Inc.	2,938	$3,149.4000			
D	Feel Great Bikes Inc.	2,962	$3,299.2100			
GH2	**First Gear**			39.00	11.00	First Gear
D	First Gear	3,044	$33.0000			
GH2	**Fred's Bikes**			40.00	11.00	Fred's Bikes
D	Fred's Bikes	3,036	$161.7000			
GH2	**Fulcrum Cycles**			41.00	11.00	Fulcrum Cycles
D	Fulcrum Cycles	1,101	$982.6500			
D	Fulcrum Cycles	1,132	$38.9800			
D	Fulcrum Cycles	1,184	$1,146.4500			

Figure 2 – The Index being built using an Array

To create an index using an Array, complete the following steps:

1. Dynamically create two Arrays at the Group Header level. One of the Arrays must contain the Group Name the other Array must contain the Page Number that corresponds with the Group.
2. Preview the report. You will see the two Arrays with all of the values necessary to build the Index. You will also see the counter that will tell you how many elements are contained in the Array.
3. Insert a subreport in the main Report Footer.
4. Share the Arrays and the counter to extract the fields from the two Arrays.

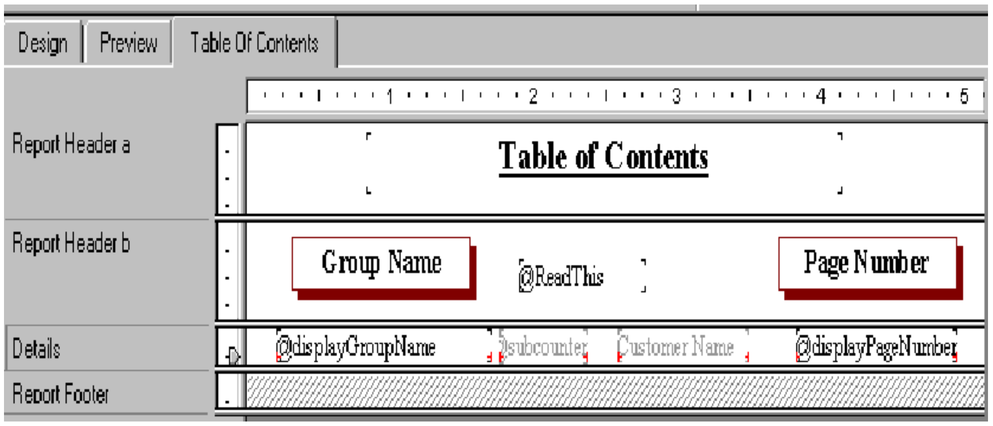


Figure 3 – Table of Contents subreport

The subreport needs a database field placed in the Details section. There must be enough details in our subreport to accommodate the number of elements in the Array.

For example:

Use the CustomerName field and place in the Details section because it produces enough details lines to support the number of elements in the Array.

From the subreport, complete the following steps to display the elements of the Array:

1. 1. Create a manual running total for the subreport and call it @subcounter. This counter will be used to display each element from the Array in turn.
2. Create two formulas. The first formula will display the Group Name and the second formula will display the Page Number.

```
//@displayGroupName
//This will display the Group Name
whileprintingrecords;
numbervar subcounter;
shared stringvar Array GroupNameArray;
GroupNameArray[subcounter]


{@displayPageNumber}
//This will display the Page Number
whileprintingrecords;
numbervar subcounter;
shared numbervar Array PageNumberArray;
PageNumberArray[subcounter]
```

At each detail line, the two formulas will display the element that is located at the position indicated by subcounter. The subcounter is incremented at each line. This allows the two formulas to display the elements at the position of the subcounter.

The Customer Name field that is driving the details section has more records then there are Array elements. The Index contains unnecessary information. In order to eliminate the extra information, and only have as many details lines display that are necessary for the two Arrays, you must suppress and conditionally suppress sections.

1. From 'Format', select 'Section'.
2. Suppress all of the sections except the Report Header and the Details section.

At the Details section, create a conditional suppression formula similar to the following to eliminate unnecessary details lines:

```
//to eliminate unnecessary details lines
Whileprintingrecords;
Shared numbervar counter;
Numbervar subcounter;
If subcounter > counter then true
```

This ensures that CR will generate as many details lines as there are elements in the Array because the Array is by definition as long as Counter.

3. Suppress the Customer Name field so you only see the Index.

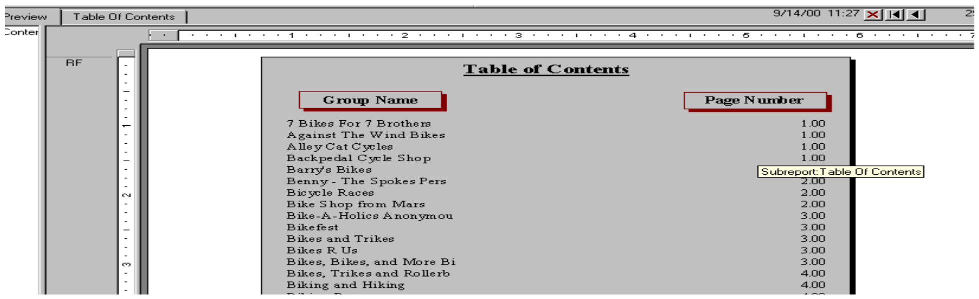


Table of Contents

Group Name	Page Number
7 Bikes For 7 Brothers	1.00
Against The Wind Bikes	1.00
Alley Cat Cycles	1.00
Backpedal Cycle Shop	1.00
Barry's Bikes	[illegible]
Benny - The Spokes Pers	2.00
Bicycle Races	2.00
Bike Shop from Mars	2.00
Bike-A-Holics Anonymou	3.00
Bikefest	3.00
Bikes and Trikes	3.00
Bikes R Us	3.00
Bikes, Bikes, and More Bi	3.00
Bikes, Trikes and Rollerb	4.00
Biking and Hiking	4.00

Figure 4 – An Index created using an Array

Note	The code provided in this example is commented thoroughly in the sample report called CR_Index_not_store_and_fetch.rpt contained in the Winzip file, CR_Arrays_Samples.zip.

Contacting Crystal Decisions for Technical Support

We recommend that you refer to the product documentation and that you visit our Technical Support web site for more resources.

Self-serve Support:

http://support.crystaldecisions.com/

Email Support:

http://support.crystaldecisions.com/support/answers.asp

Telephone Support:

http://www.crystaldecisions.com/contact/support.asp