crystal decisions™

Crystal Reports

The UflStore Function in Crystal Reports versions 5.x and 6.x

Overview

This document explains how to correctly pass data between subreports and a main report using UflStore and its related functions in Crystal Reports versions 5.x & 6.x. Understanding how this new function works will help prevent you from running into easily avoidable errors.

If you use Crystal Reports version 7.x and want to pass information between subreports and a main report, please go to:

http://support.crystaldecisions.com/docs and search for SCR7_VariableScopes.PDF, which is a technical brief on *Variable declarations in Crystal Reports version 7*.

Contents

What is UflStore?

Since the introduction of subreports in Crystal Reports version 5.0, customers have wanted a way to pass information between a main report and subreports. UflStore.dll addresses this issue by creating a global memory variable; a variable whose contents are available to both the main report and any subreport contained within it.

Practical uses for UflStore are numerous and most of them relate to subreports. If you have created a subreport and would like to use the information that it generates in the main report, then UflStore is the way to do it. The exciting thing is that the information stored is available not only to the main report, but to all subreports contained within it. Here are a few ideas:

- Create a Grand Total that adds all totals from the subreport and main report. This is a good way to combine totals that come from non-related databases.

Create an index for your main report. With a good working knowledge of your database you can create an index that will display which pages certain information (groups, totals etc) is printed on.

An overview of variables

Standard variables in Crystal Reports

Crystal Reports (CR) allows you to use variables in formulas. While programmers need no introduction to variables, non-programmers may find the following discussion helpful.

- Unlike a constant value, which is fixed and unchanging, a variable can be repeatedly assigned different values.
- A variable is like a container that can hold one value at a time. You assign a value to a variable, and the variable maintains that value until you later assign a new value. Then the variable maintains the new value until you later assign a newer value, etc.
- When using a variable in a formula, the formula looks to the variable and uses its current value in calculating the formula result. If the variable value changes, the formula looks to the new value and uses that to calculate a new result.

Declaring the variable

CR requires you to declare all variables prior to using them. When you declare a variable, you tell the program:

- The name you intend to use for the variable.
- The type of data you want the variable to hold.

The program uses this information to set aside a piece of memory for receiving and storing the values that are assigned to the variable.

NOTE	If you declare a variable with the same name and data type in two or more formulas, the formulas share the same variable. Thus, if one formula sets the value of the variable, the variable in the second (and additional formulas) reflects the change.

Variable name

You can name the variable anything you wish with the following qualifications:

- The variable name must not exceed 254 characters.
- The variable cannot have the same name as a CR operator or built-in function.

As a general rule, it's best to keep the variable name short, easy to remember, and unique (not so similar to another variable name as to cause confusion).

Data type

The data type of a variable determines the type of data that can be stored as a value in that variable. With Crystal Reports, you can create a variable with any of the following available data types:

- Number (100000).
- Currency ($30,000.00).
- Boolean (TRUE).
- Date (January 1, 1997).
- String ('Hello').
- Time (11:59:01).
- DateTime (97/01/01 11:59:01 P.M.).

Example

@Variable

```
NumberVar Total;
Total:= {database field} + Total
```

- The name of this variable is Total.
- The variable data type is Number.
- The value of the Total variable is {database field} + Total.

This means that the value stored in Total is equal to the value of {database field} for the current record, plus the last calculated value of Total. This is an incremental, or running total, which increases by the value of {database field} for each new record.

UflStore vs. standard memory variables

The UflStore functions operate slightly differently than standard memory variables.

- First of all, UflStore declares the variable and assigns a value to it in one function:

```
StoreNumberVar("Name", 1000)
```

    - "Name" declares the variable name (since variable names are strings, they are case sensitive).
    - 1000 is the value assigned to the variable Name.
- Secondly, variables can be dynamically named; no longer do they have to be hard coded at the time of report creation.

Using UflStore

Using UflStore is a two-step process:

1. You first need to store the value of a variable into memory.
2. You can then fetch the value of the variable out of memory.

Since Crystal Reports processes a report from top to bottom, you must make sure that the Store formula is evaluated before the Fetch formula. If the program attempts to Fetch a variable that has yet to be stored in memory, the message "Variable not found" will be displayed.

For example, suppose that a value is to be passed from a subreport to the main report. You must ensure that the subreport (containing the Store formula) is placed in a section, which lies physically above the section that contains the Fetch formula.

If there is only one instance of a subreport in the main report, put the subreport—containing the Store formula—in the Report Header section of the report. You can retrieve the stored variable value by placing the Fetch formula in any section beneath the Report Header.

Examples

Simple usage of Store and Fetch

Using Store and Fetch involves three formulas. The following example describes passing a value from a subreport to the main report; however, Store and Fetch can also be used to pass a value from the main report to a subreport.

1. Create the first formula in the main report. This formula assigns a zero value to a variable. The variable will eventually contain the value that needs to be passed from the subreport to the main report.

```
@Initialize

WhilePrintingRecords;
StoreNumberVar ("VarName", 0)
```

Place this formula in the main report, in a section above the section that contains the subreport.

2. In the subreport, create the Store formula:

```
@Store

WhilePrintingRecords;
StoreNumberVar("VarName", {NumberField})
```

3. In the main report, create the Fetch formula, which will retrieve the value stored in the subreport:

```
@Fetch

WhilePrintingRecords;
FetchNumberVar ("VarName")
```

Place this formula on your main report, in a section below the section containing the subreport.

4. You can now use this value in other formulas in your main report, provided the formulas are placed in sections beneath the subreport containing the Store function.

   For example:

```
@NewFormula

If @Fetch <= 1000
then
"Below or equal 1000"
```

Create a Manual Running Total using Store and Fetch

By using UFLStore to pass values between the main report and the subreport, you can do calculations such as manual running totals (MRT) that require data from both the main report and the subreport.

The following example outlines a simple MRT using store and fetch in Crystal Reports 5 or 6. The MRT takes a field from the main report and performs a running total on it in a subreport.

1. In the main report, create a formula that assigns a value to a variable based on the value of the field you want to total. This formula will also fetch the value of the subreport variable and add it to the main report variable.

```
//@store

// assigns the value of variable

//'a' and fetches

//the value of variable 'b'

//from the subreport in

//preparation for the next

//instance of the calculation.

whileprintingrecords;

if previousisnull ({table.number field})
```

```
then storenumbervar("a",0)
else  storenumbervar("a",fetchnumbervar("b"))
```

2. Place @store in the main report in a section above the subreport.
3. Suppress @store so it is not visible on the main report.
4. In the subreport, create a formula that assigns a value to the subreport variable which increments the running total based on the value of the main report variable plus the field you need totaled.

```
//@fetch
//stores the value of variable 'b'
//and fetches the value of variable 'a' from the main
report.
//it then adds the next value to the running total.
whileprintingrecords;
storenumbervar("b",fetchnumbervar("a")+ {table.number
field})
```

The running total is now incremented in the subreport based on the value of the field in your main report.

NOTE	Refresh your report with each new change to your formulas. If you view the report by clicking on the Preview tab, the Store functions are not re-evaluated. Therefore, the Fetch functions will return only the last value stored in memory.

Advanced usage of Store and Fetch: Dynamically naming variables

UflStore allows you to dynamically name your variables at runtime. When using standard memory variables, you must hard-code the name of the variable when creating the report.

There are a number of methods that can be used to dynamically name a variable at runtime:

A Numeric example

1. Creating the Store formula

Create the following formula and insert it into the Details section of your subreport.

```
@Store

Whileprintingrecords;

//specify the evaluation time of this formula as
//'Whileprintingrecords' to ensure reliability

//create a Number variable, "Counter"
NumberVar Counter;

//add 1 to the last calculated value for Counter
Counter:= Counter + 1;

//dynamically create many variables and store the

//values of each Customer's Last Year's Sales into each
//respective variable

StoreNumberVar(ToText(Counter,0,""),{Customer.LastYear'sSal
es})

// Note: The ToText() function is formatting the number as
a string without any decimal places or thousands separator
```

The @Store formula is naming variables dynamically. First, a variable is created that matches the value of Counter. If Counter = 1 then a variable named of "1" will be created and the value of the current {Customer.Last Year'sSales} will be placed into it. On the next pass through the database records, 1 will be added to the Counter variable so Counter will now equal 2. A new variable named "2" will be created and the next value of {Customer.Last Year'sSales}will be placed into it, and so on.

If we place @Store in the Details section of a report, then every time the formula is evaluated, the actual "Store variable" name that is created is dependant on the value of the Counter variable. For each record, a different value amount ({Customer.LastYear'sSales}) is stored for each variable.

For example:

- Record #1, Last Year's Sales = $1050:

@Store is evaluated, Counter is incremented to 1, so the Store is in effect doing this— Store("1", 1050)

- Record #2, Last Year's Sales = $4000:

@Store is evaluated, Counter is incremented to 2, so the Store is in effect doing this—Store("2", 4000)

- Record #3, Last Year's Sales = $2500:

@Store is evaluated, Counter is incremented to 3, so the Store is in effect doing this— Store("3", 2500)

- and so on.

Now that we have created all these variables in memory and they are storing values in our subreport, how do we get them to display in our main report? To do this, we need to use the Fetch function to create variables dynamically.

2. Creating the Fetch formula

Put the following Fetch formula in the Details section of your main report.

```
@Fetch

NumberVar Counter2;
Counter2 := Counter2 + 1;
FetchNumberVar(ToText(Counter2,0,""))
```

- *Record #1:* @Fetch is evaluated, Counter2 is incremented to 1, so the formula outputs whatever is stored in variable "1"—1050.
- *Record #2:* @Fetch is evaluated, Counter2 is incremented to 2, so the formula outputs whatever is stored in variable "2"—4000.
- *Record #3:* @Fetch is evaluated, Counter2 is incremented to 3, so the formula outputs whatever is stored in variable "3"—2500.

NOTE	Of course the counter variable name can be called anything. Call it Counter3, or Count6 or X if you like. It is not the *name* of the counter that is going to be used in the naming of the Store-Fetch variables, it is the *value* of that counter.

A String example

1. @Store2

```
StoreStringVar({DatabaseTable.CustomerName},"Hello")
```

2. @Fetch2

```
FetchStringVar({DatabaseTable.CustomerName})
```

@Store2 will create a variable that matches the value from the database field {CustomerName}. If the value in the database field = Smith then a variable "Smith" will be created and the value "Hello" will be placed into it. If the database field = Jones then a variable "Jones" will be created and the value "Hello" will be placed into it, and so on.

Tips, tricks and limitations

- Standard variables in Crystal Reports are not case sensitive. However, variable names are case-sensitive when using UflStore. The variable names must match exactly when using Store and Fetch. For example:

  ```
  StoreNumberVar("GrandTotal", 1000) and
  FetchNumberVar("grandtotal")
  ```

Because GrandTotal and grandtotal do not match, Crystal Reports will produce the error message "Variable not found" when trying to evaluate the Fetch formula.

- Drilldown cannot be performed on a report that is using UflStore.
- When previewing the report, refresh the data for the report. If you view the report by clicking on the Preview tab, the Store functions are not re-evaluated. Therefore, Fetch functions will return only the last value stored in memory.

- When using Fetch we recommend that you assign them to standard memory variables before using them in calculations. For example:

```
@RunningTotal

    NumberVar Counter;
    NumberVar GrandTotal;

//assign to a standard memory variable
Counter := Fetch("SubReportTotal")

    GrandTotal := GrandTotal + Counter
```

- Do not suppress your subreport or the section that contains the subreport; the UflStore functions must be included on the report to evaluate properly. You can, however, suppress sections within the subreport.
- "Variable not found" is the most common error associated with UflStore. The following is an explanation of why this happens and how to prevent this:
    - The value that you are storing is null. The variable has not been stored in memory, therefore, it cannot be found.
    - The Fetch formula is placed before the Store formula. This will cause the same error due to the fact that the Store formula has not yet been evaluated.

To prevent these problems from occurring, we recommend that you:

3. Place the subreport (containing the Store formula) in its own section before the Fetch formula.
1. Initialize the variables that you are using in your `Store` and `Fetch` formulas in the Report Header of the main report. For example:

```
@Initialize

StoreNumberVar ("My Variable", 0);
```

By setting the variable to zero, you are storing it into memory. Crystal Reports will now be able to find this value.

2. On the File menu, select Report Options, and then select the Convert Null to Default check box.

This must be done on the report containing the Store formula. If this is the subreport, right-click on the subreport and select Edit to set the Report Options.

Syntax

Uf5Store (16-bit) and U25Store (32-bit) contain the following functions. You can find these functions in the Functions box of the Crystal Reports Formula Editor, under the Additional Functions folder.

Note that in all examples, 'X' denotes a String.

StoreNumberVar(X, Y)

Stores a Number, Y, to the variable X. If a formula field contains this function, it returns the numeric value Y.

```
StoreNumberVar("gTotal", 1000).
```

Returns: 1000

StoreStringVar(X, Y)

Stores a String value, Y, to the key X. If a formula field just contains this function, it returns a String value Y.

```
StoreStringVar("My Name", "Beanie")
```

Returns: Beanie

StoreDateVar(X, Y)

Stores a Date value, Y, to the key X. If a formula field just contains this function, it returns a Date value Y.

```
StoreDateVar("MyBDay", Date(1970,04,02))
```

Returns: April 2, 1970

StoreCurrencyVar(X, Y)

Stores a String value, Y, to the key X. If a formula field just contains this function, it returns a String value Y. This is the same as using SaveNumberVar ().

```
StoreCurrencyVar("gTotal", 1000)
```

Returns: $1,000.00

StoreBooleanVar(X, Y)

Stores a True/False value, Y, to the key X. If a formula field just contains this function, it will return a String value Y.

```
StoreBooleanVar("Gender Is Male", True)
```

Returns: True

FetchNumberVar(X)

Given the key X, this function will return a numeric value that is stored under this key.

```
FetchNumberVar("gTotal")
```

Returns: 1000

FetchStringVar(X)

Given the key X, this function will return a string value that is stored under this key.

```
FetchStringVar("My Name")
```

Returns: Beanie

FetchDateVar(X)

Given the key X, this function will return a date value that is stored under this key.

```
FetchDateVar("MyBDate")
```

Returns: April 2, 1970

FetchCurrencyVar(X)

Given the key X, this function will return a currency value that is stored under this key.

```
FetchCurrencyVar("gTotal")
```

Returns: $1,000.00

FetchBooleanVar(X)

Given the key X, this function will return a Boolean value that is stored under this key.

```
FetchBooleanVar("Gender is Male")
```

Returns: True

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