

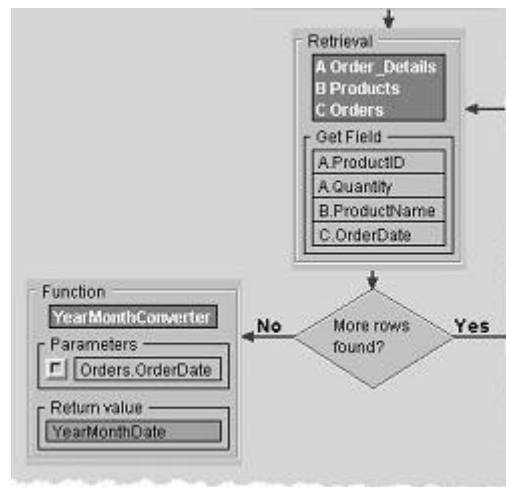
## Tutorial 11 - Building the ProductSalesStats Report Procedure - part II

The ProductSalesStats1 report, created in the Tutorial 10, has one logical flaw: it sums up the product shipment quantities for every date there were shipments, showing the bar values for every such date. But what we really want is the quantity subtotals for every month. In other words, if there were shipments on July 1, 2004 and July 20, 2004, the report will show two bars, but we need just one bar with all quantities for the month of July, 2004.

Scribe provides means to do additional processing of data by accumulating it in the Array, and then sorting/grouping it. The Array can be thought of as an equivalent of a temporary database table, except that it "lives" in the volatile memory rather than on the hard disk.

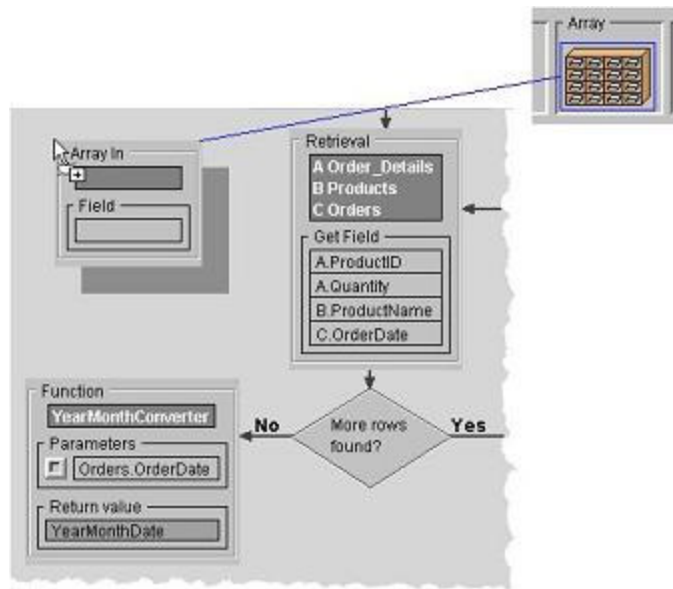
The Array component is always used in pairs: the first instance of this component is used to accumulate data in memory, and the second instance dispenses it back to the calling procedure. This will be further clarified in the example report that we will create in this Tutorial.

Open the ProductSalesStats1 report and add a User Function component to it; set the component to YearMonthConverter - the User Function that we created in Tutorial 8 (in the Main Resources tab of the Palette you will have to select it from the tree node called "User functions"). Set the input parameter ("Date") to Orders.OrderDate, and drag the *New variable* into its return slot; rename "var1" to "YearMonthDate". You should now see something like this in your Workspace:



As you recall from the Tutorial 8, this function converts a date with an arbitrary day to its first-day-of-month equivalent.

Now we will bring in the Array component:

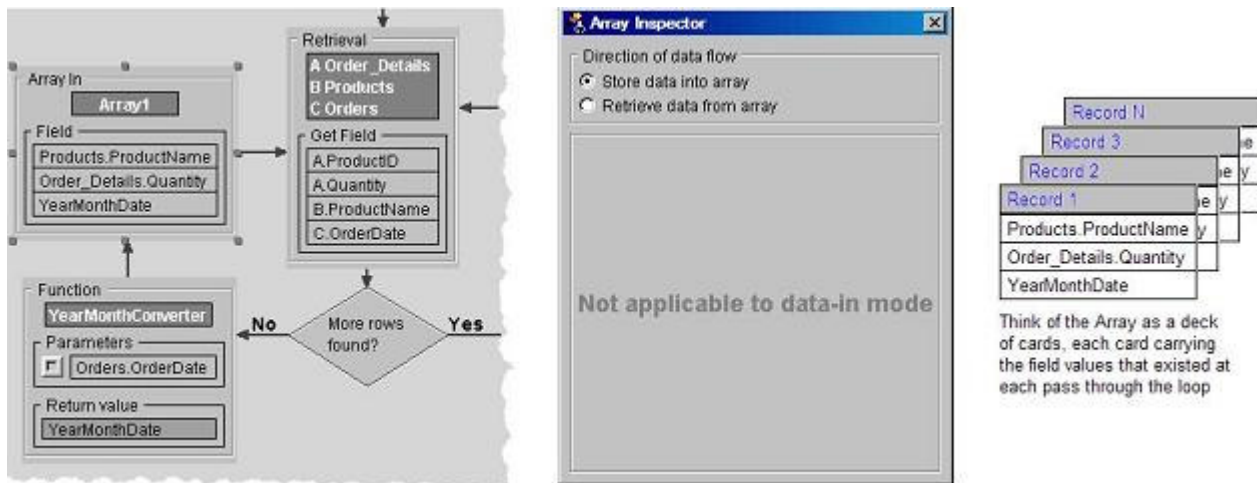


Once the Array component lands in the Workspace, it gets assigned a name. For the first Array component the assigned name is "Array1". It can be changed by editing it in the Palette (see Tutorial 4). The Array names are located in the Name column, under the newly added section "Array".

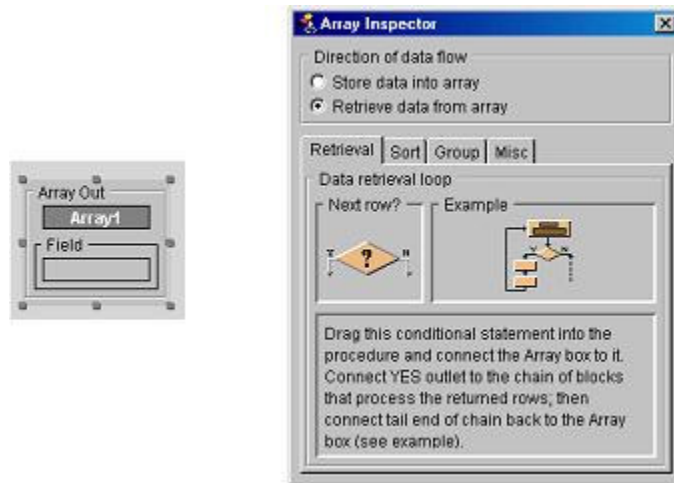
Resize the Array component by dragging down the bottom control point until it has three fields. Set these fields to:

- Products.ProductName
- Order\_Details.Quantity
- YearMonthDate

Note that the Array Inspector's data flow direction option is set to "Store data into array". This is a default setting for the first instance of the Array component in the flowchart. This setting means that as the Array component is executed in a loop, the values of variables in its fields are added to the internal storage.

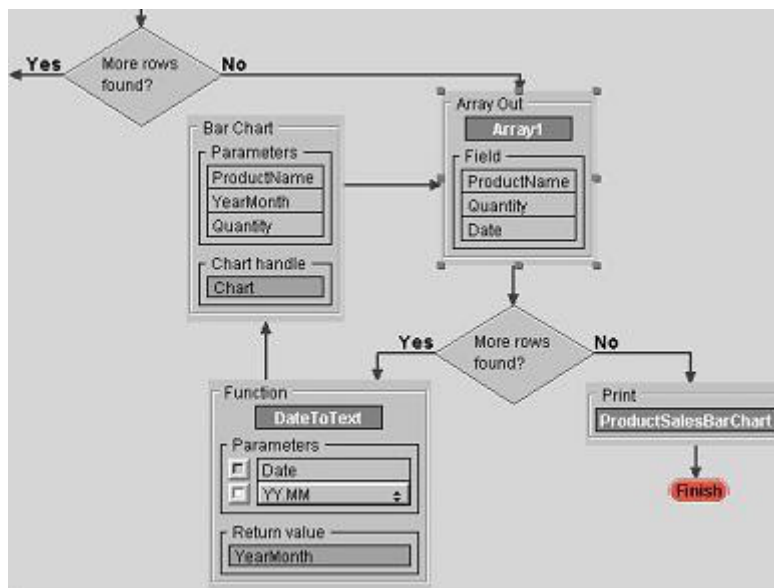


With the receiving end of the Array set, we can now turn our attention to the dispensing part of the Array. Drag in the Array component from the Palette - this would be the second instance of this component. Since there already is the Array component named "Array1", and it is marked as "Array In", the newly loaded Array is automatically set as "Array Out", and its Inspector is set accordingly:



You may notice some similarity between this Inspector and the one for the Retrieval component. Both provide a "Next row?" component that can be dragged into the Workspace, and both offer extended processing of the returned rows (sorting and grouping). This is because the "Array Out" variety of the Array acts much like an in-memory database table, and like the database Retrieval, returns the rows of fields stored in it, in a loop.

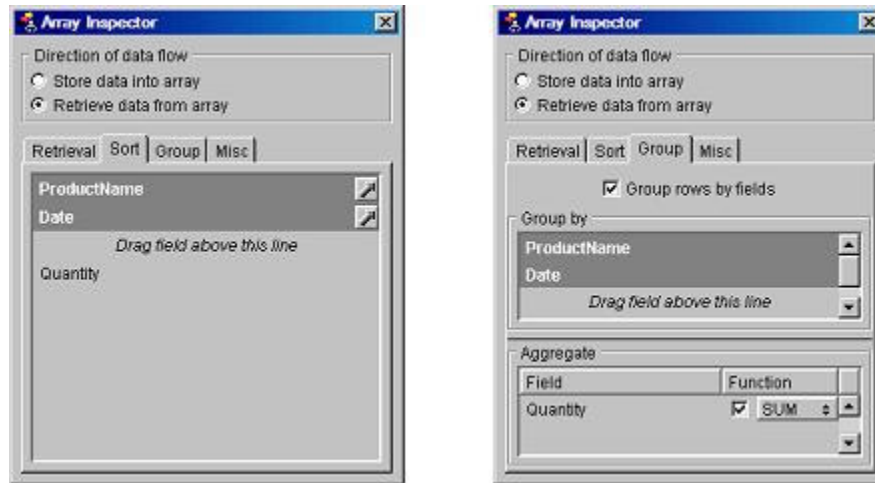
The "Array Out" component returns the same number of fields as its "Array In" counterpart, and in the same order. Variables receiving the values from the "Array Out" do not have to be the same as the variables used to move the data into Array. In other words, we can put in new variables that will automatically get the same data type as the original variables. With this in mind, we will resize the "Array Out" component to have three fields, and put the *New variable* in them. Every time we drag in the *New variable*, it becomes var1, var2 etc. After renaming them to more appropriate mnemonic names("ProductName", "Quantity" and "Date"), and organizing a loop with the "Next row?" component, we should have the following flowchart:



The Date variable, extracted from the Array Out component, is passed to the DateToText function that converts it to YY.MM format and places the result into YearMonth variable (note that this date format must be added to the list of supported formats prior to bringing in the DateToText function using the [Date Format Editor](#)). Then the ProductName and Quantity variables, extracted from the Array Out component, along with the newly created YearMonth variable, are passed to the Bar Chart component for creating the bar chart.

Extraction of rows from the Array component can be refined by sorting and grouping the rows and fields. This is done via the Inspector, and is quite similar to the respective facilities in the Database Retrieval Inspector. To

achieve the desired appearance of the Bar Chart, we sort the Array content by the ProductName and Date, and group by the same fields, aggregating the Quantity field as SUM:



#### Note on aggregation of fields in Arrays:

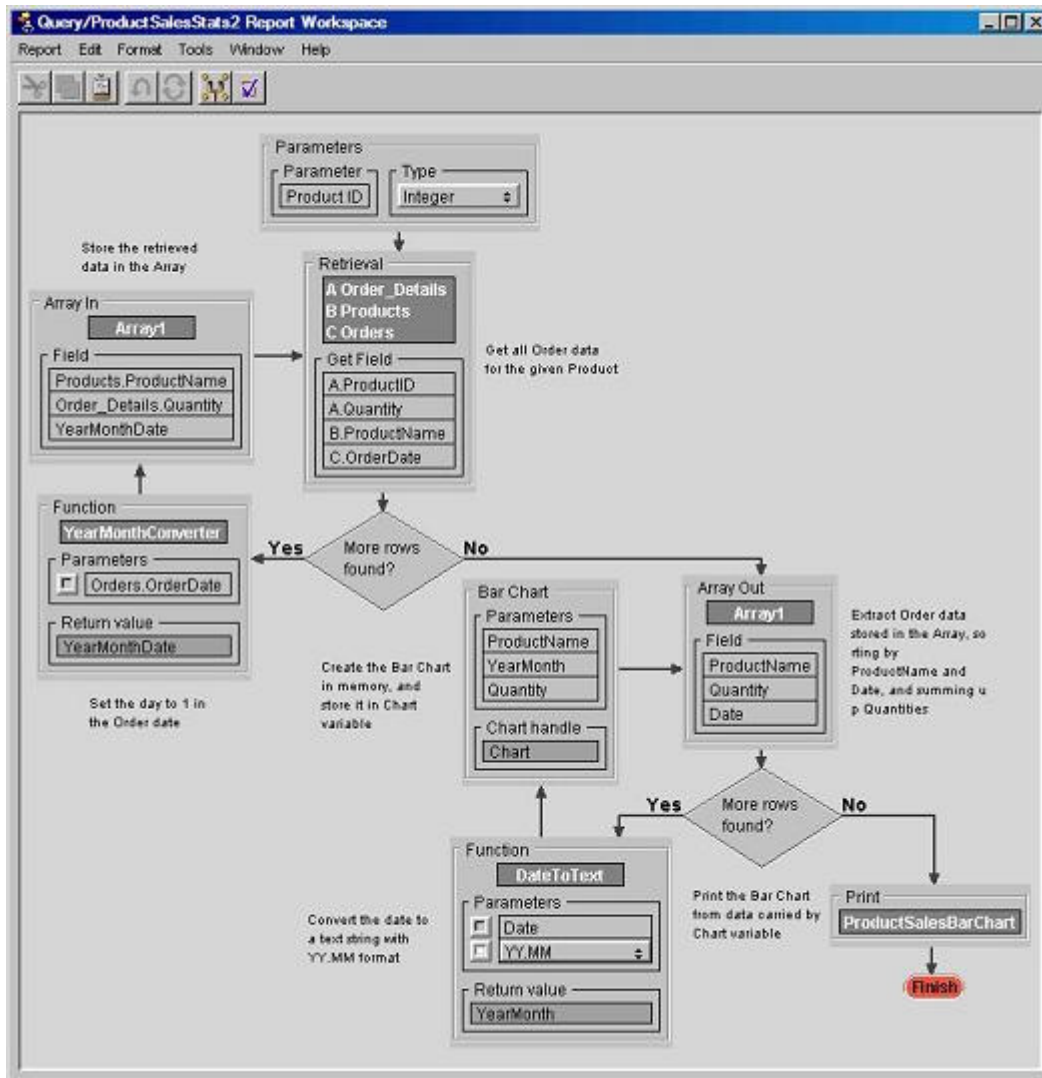
There is a difference between aggregation of fields in Array and Retrieval component Inspectors. The Retrieval Inspector has to follow the requirements of the SQL's GROUP BY clause which mandates using the aggregate functions on all fields listed in the SELECT and not in the GROUP BY. In other words, any field not used in grouping must be aggregated in one way or another.

The Array object, while following the general idea of the SQL-based grouping, is not bound by the SQL restrictions and affords a more liberal treatment of the non-grouping fields. In Array Out, if the field is not included in the aggregation function, it will be set to the value stored in the first row. This may or may not be useful (depending on the particular situation), but at least the user is not obligated to aggregate the field that does not need to be aggregated - hence the checkbox to let the user choose.

If none of the non-grouping fields is selected for aggregation (in other words, the grouping loses its meaning), then an error message is displayed.

Arrays can also be passed into and out of the lower-level procedures. See Tutorial 18 for detailed coverage of the use of Array components.

The complete Report Procedure is shown in the figure below:



Running the report with Product ID = 1 creates the following Bar Chart:



Save the Report Procedure as Query/ProductSalesStats2.