R&R with XML

A Liveware Publishing, Inc. White Paper
March 14, 2003
Christian A. Strasser
CTO – Liveware Publishing, Inc.

Introduction

XML has become pervasive. Like its cousin, EDI, many companies are looking to it to provide a consistent, portable method of integrating data between disparate systems both within and between companies.

At its simplest, XML is a structured way of organizing data for easy transport via the Internet. An XML document is plain text and consists of a number of tags (element descriptors) and the actual data (elements) themselves.

A sample XML file is shown below:

```
<?xml version="1.0"?>
       <whiteboxes>
              <company>
                     <name>ACCES I/O Products Inc.  </name>
                     <address>  San Diego, CA </address>
                     <phone>  800-326-1649 </phone>
                     <url>  www.accesioproducts.com </url>
                     <specialty>   </specialty>
                     <support>  Toll-free technical support </support>
                     <products>  Barebones systems </products>
              </company>

              <company>
                     <name>Advantec Computer Systems LLC  </name>
                     <address>  Hopkinton, MA </address>
                     <phone>  877-696-4848 </phone>
                     <url>  www.advanteccomputer.com </url>
                     <specialty> Custom-white box computer systems </specialty>
                     <support>  Toll-free technical support </support>
                     <products>  Barebones systems </products>
              </company>

</whiteboxes>
```

Source: VAR Business 500 for 2000

This example shows the data associated with two suppliers of "White Boxes." These are generic, non name brand computers. The information contained includes the companies' addresses, phone numbers and specialty areas.

The general trend over the years has been to streamline database storage and access to maintain the bare minimum amount of data and the quickest path to it. By its nature, XML breaks away from this: It is a very verbose specification and an extremely inefficient way to store and represent data. There are advantages to it, however:

- An XML document is readable by a person without a computer.
- Like HTML, it is "stateless." This means that an XML document stands on its own and doesn't require some knowledge of where it came from or where it's going

- It is extensible. This means that the document can easily change to respond to unforeseen requirements in representing the data.
- It is self-contained (and –defining)1. This is because an XML document can include an entire Schema (a schema is a structured way of identifying the parts of and rules for updating a database record).

The example shown above is a simple two record data table. In traditional database structure (table with rows and columns), it would be represented like this:

Companies						
Name	**Address**	**Phone**	**url**	**Specialty**	**Support**	**Products**
ACCES I/O Products Inc.	San Diego, CA	800-326-1649	www.accesioproducts.com	.NULL.	Toll-free technical support	Barebones systems
Advantec Computer Systems, LLC	Hopkinton, MA	877-696-4848	www.advanteccomputer.com	Custom-white box computer systems	Toll-free technical support	Barebones systems

Database: Whiteboxes

It should be apparent from the XML sample above, that data is represented hierarchically. That is, the definition of each element cascades downward from general to specific. Thus, the list of whitebox producers includes companies. Each company is described by seven attributes. To illustrate this further, consider this example.

You are receiving data from these white box producers and determine that bursting the address into its separate components would help with marketing and support. After a little analysis, you conclude that each address could be defined as two discrete parts, the city and the state code. With a traditional database, you'd have to redefine the table and create a process for collecting and storing the data in separate pieces. With XML, you only have to identify new tags and write the transformation needed to split the old address into its separate parts.

The data shown above would now include:

```
…record 1…
<address>
      <city> San Diego </city>
      <state> CA </state>
</address>

…record 2…
<address>
      <city> Hopkinton </city>
      <state> MA </state>
</address>
```

This illustrates the extensible and self-defining nature of the language.

Database vendors have begun to address the need to store and produce XML documents. Oracle, Microsoft, Software AG and others have updated their database products to work with native XML. As time goes on, XML will become more common, perhaps becoming the basic means of capturing and storing data of all types.

The inefficiency in the specification can be addressed in a number of ways. One would be to take advantage of its text-based nature and apply standard file compression algorithms to the data. Another would be to create a structure that "knows" that the data is XML, but stores the information in a more compact / indexed fashion.

XML Reporting

As with any database format, someone will eventually want to present the stored information as a nicely formatted, human-readable document. R&R can handle this very well. To build reports using R&R, you'll need the following:

1. A properly formed XML document
2. An ODBC driver (we recommend those developed by Data Direct Technologies, www.datadirecttechnologies.com)
3. A copy of R&R SQL Edition (standard with version 10)

Generally speaking, XML documents represent relatively small sets of data, although nothing in the definition or specifications of the language require that this be the case. This comes about because of the way that the parsers work (either by operating on the entire document in memory or by processing it incrementally)2.

They can also become very complex, effectively becoming an alternative database in themselves, with additional documents defining an overall schema, normalized, associated tables using keys to link back to a "master" document, etc. The samples we'll present here will be relatively short for simplicity. However, if the ODBC driver being used can support it, R&R will make use of any features passed back.

For the examples here, we'll use the White Boxes database shown in part above. The actual database consists of twenty-eight different companies in a number of states and countries. While not especially large, the dataset is big enough to illustrate the concepts needed.

Well-Formed

The XML document shown in the introduction contains a subset of the data in the entire White Boxes database. It is a well-formed, simple XML data source.

Being well formed requires a number of constraints be upheld. Many of these are beyond the scope of this document (see the World-Wide Web Consortium's website for the full Recommendation at http://www.w3.org/TR/REC-xml.

Generally, however, the requirements include:

- Lnguage version declaration at the beginning of the document

  ```
  <?xml version="1.0"?>
  ```

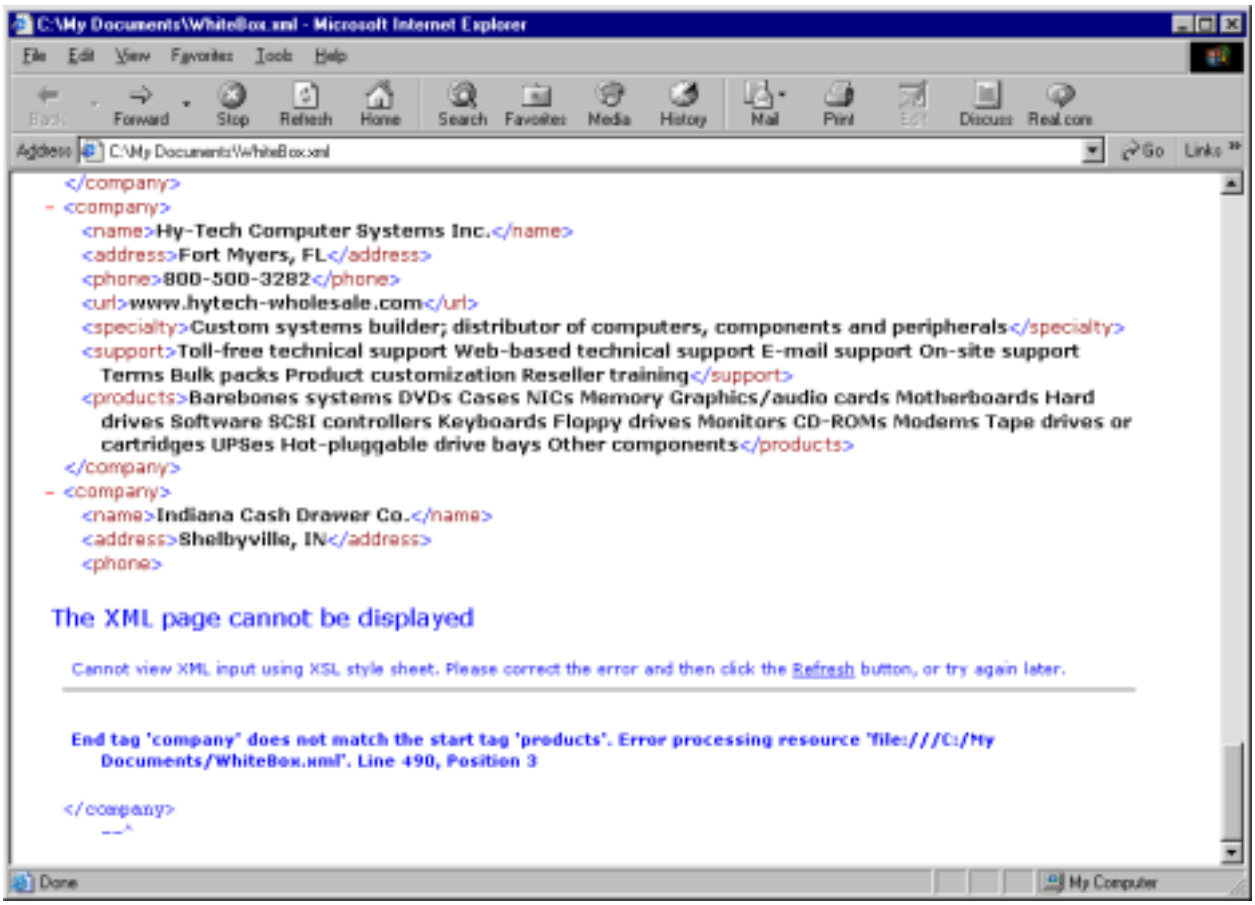
- The names used in the start and end-tags must be matched. Note that they do not need to be contiguous. That is, interim nested additional elements can be displayed within a higher-level element so long as they all terminate. This can be shown by the syntax:

  ```
  <element tag>Element</element tag>
  ```

- No attributes can use the angle bracket in it.

Additional rules apply to validity, comments, defining legal selection values, etc. Generally, the specification defines all the rules needed to create a full-fledged database in the XML format.

Because the document is interpreted, the area where the document breaks down may not be initially visible. When this occurs, the browser will display an error indicating the problem as shown here:



Specifically, the data was not terminated properly with an ending element matching with a starting element. Note though, that additional tags can be dropped in wherever necessary with no processing errors. For example, if a particular dataset came through with a more detailed address, it would be processed without any hiccups:

```
<address>
    1231 South Peach Blossom Lane
    <city>Norcross</city>
    <state>GA</state>
</address>
```

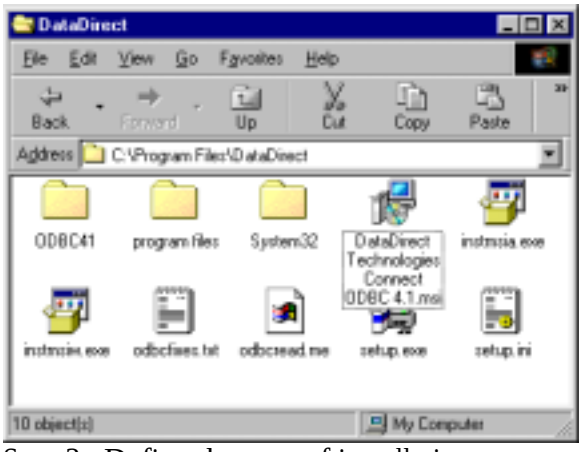
Configuring ODBC

For this paper, we will use version 4.1 of DataDirect's ODBC driver. The actual file and version is: IVXML18.DLL dated 2/15/2002. If you don't currently have R&R Version 10 with the new drivers, samples of them can be downloaded from DataDirect's website and installed on a local

system for test purposes. The installer takes you through the process of selecting one of a number of available drivers.

ODBC Installation and Configuration

Manually configuring the XML driver first consists of executing the wizard to install it on your PC. Here, the icon shown in the window at right is double-clicked. This brings up the wizards which take you step-by-step through the installation:



Step 1: Acknowledge the Wizard and click [Next]

Step 2: Specify the Maintenance or initial configuration of your drivers. Click [Next]

Step 3: Define the type of installation you want. For this example, we used a Single Driver / Desktop installation. Click [Next]

Step 4: Select your options. DataDirect will create basic data sources for you and replace existing drivers if you so choose.

Step 5: Confirm or change the installation directory to place the DataDirect files. Click [Next]

Step 6: Select the ODBC driver (XML in this case) that you wish to install and the sub-options for it. Click [Next]

Step 7: Enter your identification information as well as your serial number and product key. Click [Next]

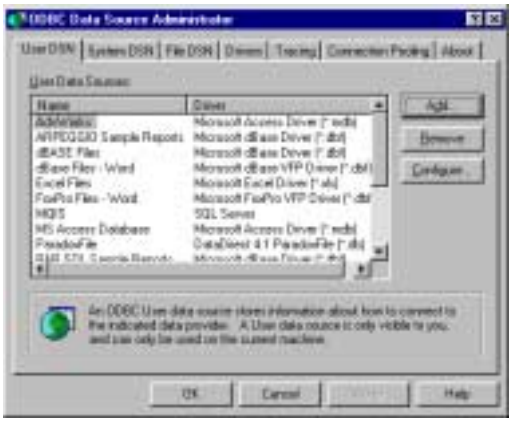
Step 8 (not shown): Start the process, click [Next] to continue.

Step 9 (not shown): Click [Finish] to complete the driver installation.

Define Your Data Source

Step 1: In Windows 98, Click on Start / Settings / Control Panel. Double-click on "ODBC Data Sources." For other Windows systems, the sequence is similar, but the ODBC configuration applet may be beneath another option.

Step 2: Determine if you want a User or



System Data Source. User DSNs exist only for a particular person / user who logs onto a specific computer system. System DSNs exist for the computer system regardless of who logs onto it. For this example, we'll create a User DSN called "XML." Click OK.

Step 3: Select the appropriate ODBC driver.

In the example here, the DataDirect driver is the first one listed. Click [Finish] to begin the specific definitions for the particular data you wish to connect to.

Configure Your Data Source

Step 1: Name your data source (here, we've called it simply, "XML"). Provide a description for it.

Step 2: Click the [Add] button to identify the locations of the various files that will make up your complete data set. You have the option of specifying an entire folder, a discrete XML document or a portion of an HTML document. Here, we've specified the file WhiteBox.XML. Advanced options is left alone.

Step 3: In most cases, you should be able to avoid special configuration issues, such as row hints or driver-specific options. To verify that your data works, click on "Test Connect."

Step 4: Once the connection succeeds, click on the[Apply] button followed by [Close]. Your datasource has been created and is available for use within R&R.

R&R SQL Edition Version 10

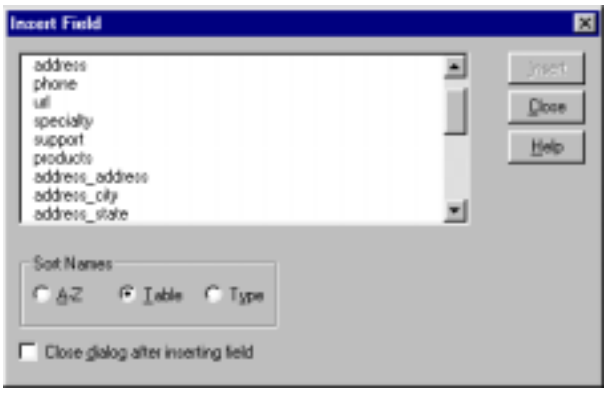
With the introduction of Data Direct Technologies' ODBC drivers, R&R can now access XML data sources. The requirement is that the XML drivers be installed on the server and/or PC that will run the report(s). With appropriate ODBC drivers, earlier versions of R&R could access XML data as well, but it was up to the individual to procure their own drivers.

Starting a report with an XML datasource is the same as beginning any other type of report. That's one of the key benefits of ODBC: It hides the mechanics of connecting to your data so that the applications see a consistent "face" for interactions.

From the menu, select File | New. Select "Blank Report" from the popup window and select the "XML" datasource from the list. The various individual parts that were identified in the Step 2 of the Configuration portion, will be represented as distinct master tables to select (see figure at right).

In this case, we're going to select "whiteboxes" as our Master Table. After this, we're in the familiar report designer.

Inserting fields allows us to place the different data elements on the report as with any standard database. These fields are dropped onto the layout as with any other type of database. Below is a picture of the "Insert Field" dialog. You can see that the fields shown in the list (i.e. address, phone, url, etc.) correspond to the named pairs of tags in the XML document, such as <phone></phone>. Of special interest is the set of fields at the bottom of the list, including address_city and address_state. These correspond to the expansion of the address tags mentioned earlier, to include the breakout of the address into its component elements. The address_address field is equivalent to the address field, but the records without the city and state broken out have no data for these fields.



The products field is a character field that contains freeform text. Thus, calculations may be defined that pull out the individual product elements for each of the vendors. Each of the products listed is separated by two spaces. Using this information, we can define a series of calculations to extract them individually.

Calculation Name	Expression	Used In	Comments
TrimProd01	LTrim(Products)	Prod01End Prod01	Removes excess space from the left side of the field; returns a character field
Prod01End	IIf(at(" ", TrimProd01) <> 0, at(" ", TrimProd01) - 1, len(TrimProd01))	Prod01 TrimProd02	Identifies the location, within TrimProd01, of the first occurrence of two spaces; returns a numeric field The IIF ensures that if only one product is in the list then the position where that product ends is returned
Prod01	left(TrimProd01, Prod01End)	LAYOUT	Displays the first product indicated in the list; returns a character field
TrimProd02	substr(TrimProd01, Prod01End + 3, len(TrimProd01) - Prod01End)	Prod02End Prod02	Removes excess space from the beginning of the list of remaining products, starting from where the first product ended; returns a character field
Prod02End	IIf(at(" ", TrimProd02) <> 0, at(" ", TrimProd02), len(TrimProd02))	Prod02	Identifies the location, within TrimProd02, of the first occurrence of two spaces; returns a numeric field The IIF ensures that ifi there are only two products in the list, then the position where the second product ends is returned
Prod02	left(TrimProd02, Prod02End)	LAYOUT	Displays the second product indicated in the list; returns a character field

Additional products can be extracted from the text field by applying similar calculations through the remainder of the report (i.e. to get Prod03, Prod04, etc.).

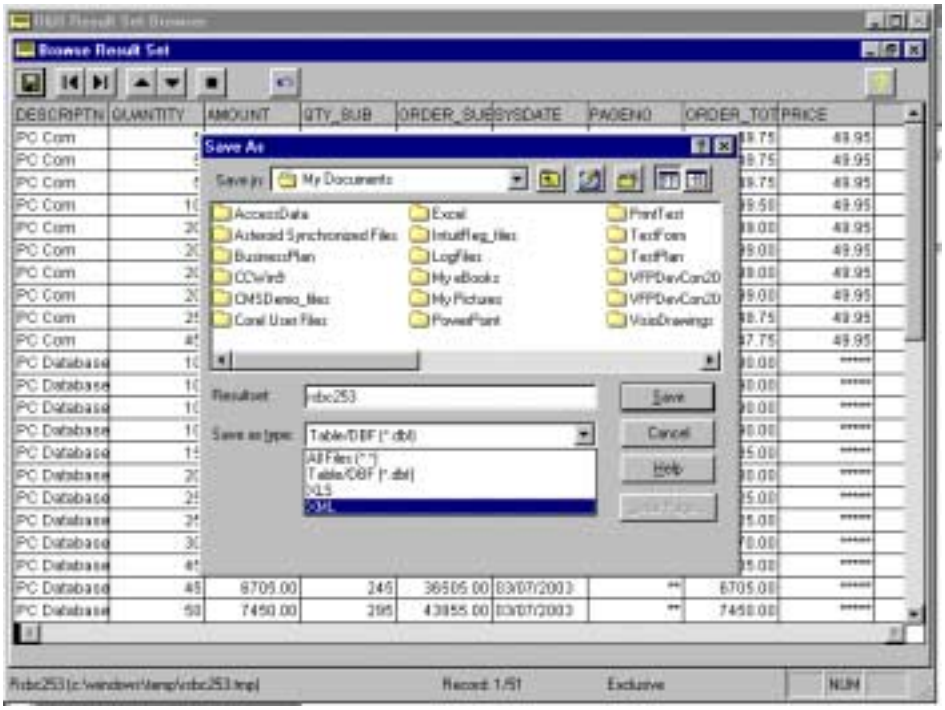
One important point to note is that all the base fields identified within the XML file are all character. This is irrespective of schema definitions or other constructs that might attempt to put some sort of typing onto a value. The result is that the report developer must recast all fields into their semantic types. Some of the necessary calculations are shown in the following table:

Field Value	Semantic Type	Conversion	Result
“The Computer Store”	Text	NONE	“The Computer Store”
“05/17/2000”	Date	CTOD(“05/17/2000”)	{05/17/2000}
“1500.34”	Numeric	VAL(“1500.34”)	1500.34
“True”	Boolean	Iif(FieldValue = “True”, .T., .F.)	.T.

Using R&R to Create XML Documents

For the very brave, R&R could always be used to generate XML, just as it could generate HTML: by defining a report in such a way that tags were created at the appropriate positions within the report and exporting the results to a text file with the necessary extension. This was true of both the Xbase and the SQL editions.

The latest build of R&R Version 10 includes a revised Resultset Browser that simplifies this process greatly. Now, when the Resultset is displayed, the user can select XML as the "Save type" and create a well-formed XML file from the data displayed (see graphic).



The resulting XML file will have the data shown in the grid as well as the schema describing the data at the top of the file (see appendix for a partial sample of the file created).

Handcrafted XML

Using the XML option to save data this way will be sufficient for many organizations. However, it is likely that others will need to be more precise with their XML data creation. This could be for several reasons:

- They want to burst the data being produced so that different vendors / customers / contacts receive only their portion of the data
- They want to precisely control the fields and order of the data being created so that the data conforms to some external specification, and the Resultset Browser export is too broad
- They need to identify other information within the schema that isn't included in the default export created by Resultset Browser
- They need to separate the schema from the data and Resultset Browser automatically includes both together

The answer in all of these cases is to define a report that will generate all of the necessary XML. This section will go over some of the basics to generate a rudimentary (but usable) report. We'll stick with the SQL Edition, but use the sample data as the basis for our report.

We'll imagine a business situation where we want to notify our customers about their orders and provide them with confirmations. Typically in this situation, we'd need a few things:

1. The Company Name
2. The Customer Contact Name
3. The Customer Contact E-mail Address
4. What was Ordered
5. When it was Ordered
6. What they Owe
7. How it'll be Shipped

The sample data delivered with R&R is sufficient for defining a report with this information. Issues include: There is no e-mail address stored, so we'll have to derive one.

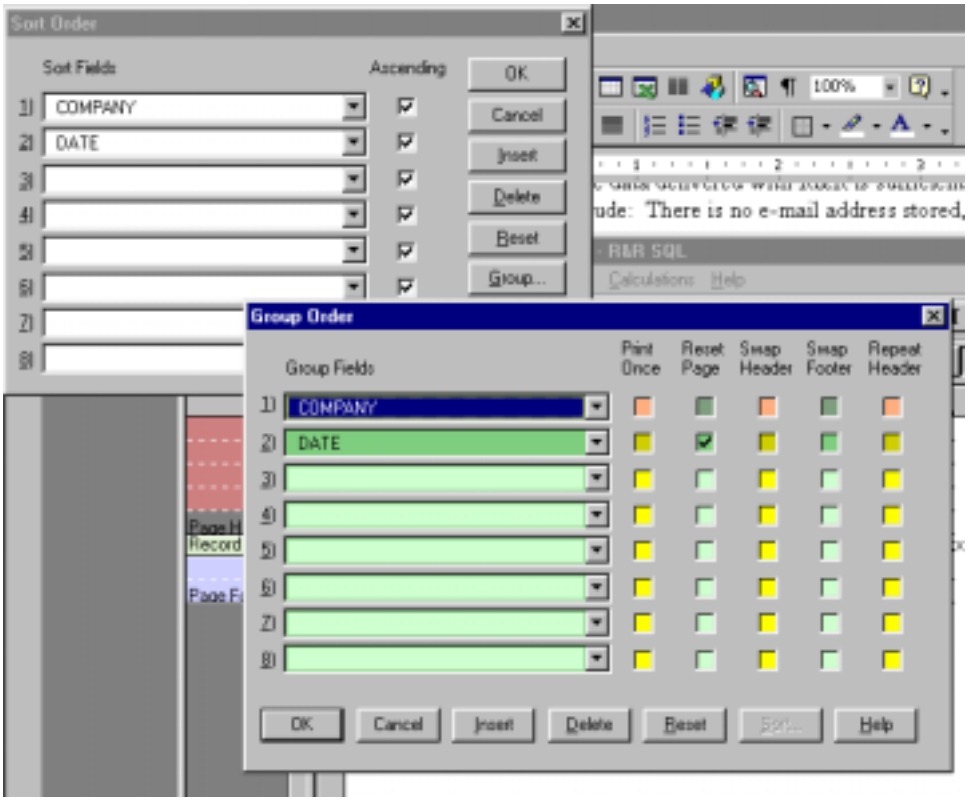
The fields required to supply the above data to each customer includes:
rrcust.company
rrcust.last_name
rrcust.first_name
rrorders.date
rrorders.shiptype
rritems.quantity
rrprices.descriptn
rrprices.price

Additionally, we'll need to create a few new fields, such as e-mail address, extended amount and total amount for the entire order.

The report begins with selecting the Master Table. For simplicity, rrcust will be chosen. This is because all other data naturally flows from the customer. Additional joins will be defined as follows:

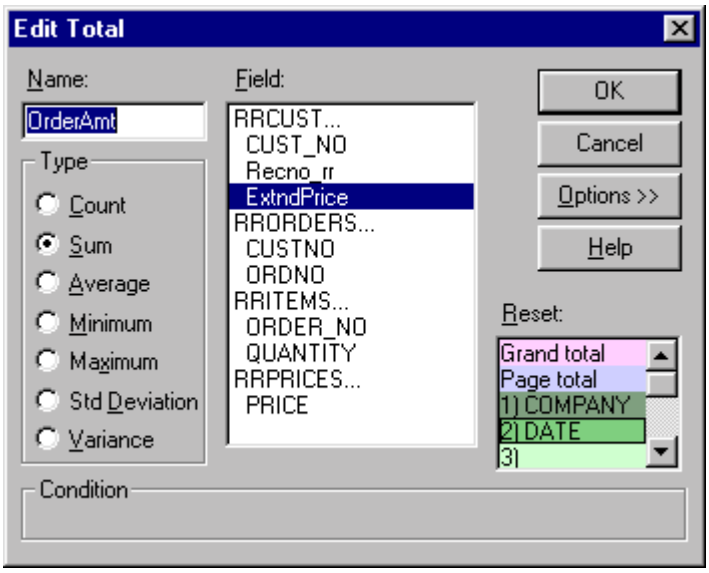
```
rrcust.cust_no ➔    rrorders.custno
                    rrorders.ordno ➔    rritems.order_no
                                        rritems.product_no ➔ rrprices.prod_code
```

To burst a report, we need to define a page-reset grouping level. We have a couple of choices here. The first is to use the customer code / company name and deliver any orders that occur within our criteria to them. The second is to create a second level for date and send individual confirmations for each separate order. We'll implement this latter method for the report. This report is set to be sorted / grouped according to the screen below. Note that the grouping level corresponds to the sorting level. For this report, this is fine. However, in practice, you'd want to ensure that grouping occurred on a unique field so that unintended consolidation didn't occur (i.e. two people with the name "Smith" having their information grouped together).



By setting the Date field to be the page reset level, we can be sure that each burst will generate a report subset unique to each customer, but containing only the information contained for that particular order. Note again that it could be possible for companies to order multiple times within a day. For that situation, you'd replace DATE with the order number field.

Next we'll create the calculated and total fields for determining the order amount. The first field, we'll call "ExtndPrice." It will represent the extended price (quantity x price). The formula is very simple: `rritems.quantity * rrprices.price`. The second field will be the total of the extended prices for each separate date (order). The total field defined is "OrderAmt" and its screen is shown at right:



Finally, we'll create an e-mail address to use for this example. We'll do it by using parts of the person's name and their company name. This calculation will simply be called "email" and defined as

lower(Left(FIRST_NAME, 1) - LAST_NAME - "@" - StrRep(COMPANY, " ", "") - ".com"). What this calculation does is combine the first letter of the customer's first name and join it to their last name. This, in turn, is linked to the company name (with the spaces stripped out) by the "@" symbol. Finally, ".com" is put at the end. The "-" operations are a shorthand way to accomplish an RTrim (i.e. to remove spaces from the right side of a data field).

At this point, we've defined all elements that we need for our confirmation document. Now it just needs to be formatted as an XML document.

As indicated before, we must create a well-formed XML document. The first requirement here is that it begin with the appropriate declaration. The minimal header is:

```
<?xml version="1.0"?>
```

This must appear at the start of each document that we send. This implies that we need either a page header, or a group header corresponding to the page reset level that we're using for our burst. The recommended method is to use the group header, since it only prints once per group. A page header could cause problems if you had data that extended over multiple pages. Insert a group header for the Date field, and type the header shown above at the left margin. Also, delete the existing page headers and footers automatically created when you started R&R.

The next step is defining a name for the document's data contents as well as a hierarchy for the data to follow (in practice, these requirements would be spelled out in advance with your various customers. R&R allows you to generate multiple extracts for your customers depending on their specifications using formulas, line logic and other features). This name is equivalent to the table. We'll call it "OrderInfo." Thus, at the start and end of each customer's XML document, we need to define the tags:

```
<OrderInfo>
</OrderInfo>
```

The detail information will be placed between these two tags. We'll place them onto the report by typing them on the layout, rather than creating calculations. Again, since these only need to be included once per grouping level, we'll add another group header on Date as well as a group footer. Insert the header below the XML header line and the footer below the line with the total field, OrderAmt (if present). Because XML ignores spaces, we can indent these two tags a little so that the resulting extract is easier to read. Be very careful as you define your report. Remember that XML is case sensitive and an XML parser will reject your file if you use "different" tags (such as <OrderInfo> and </Orderinfo>).

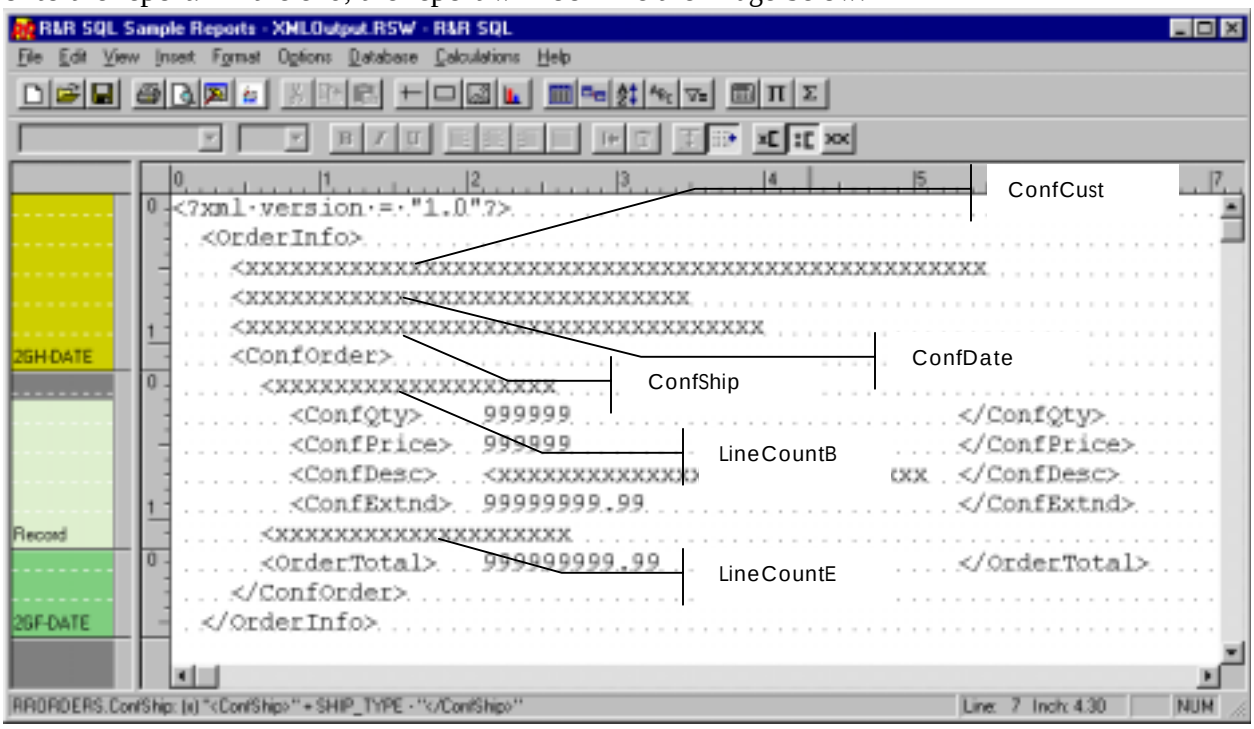
Now we get into the more interesting part: Defining the logical structure of the data. We have to ask first, what changes from record to record and what remains static? We can draw some conclusions based on our table linking hierarchy:

1. Each confirmation can only go to a single customer
2. Each customer record can have one or more orders (although we're assuming only one order per day)

3. Each order record can contain one or more items (in practice, it must contain at least one item, or it's not an order)
4. Each item record must include a single price record (this is a business rule from our particular sample data. In actuality, a company could have multiple prices for an item for many reasons: price changing over time, price unique to a particular customer, price adjusted seasonally, etc.)

From this, we know that each confirmation will contain only one element from rrcust and rrorders. To represent this, we'll have to add a few more group headers for the date field. Then, we'll define calculated fields for the data elements needed on each. These will be shown below.

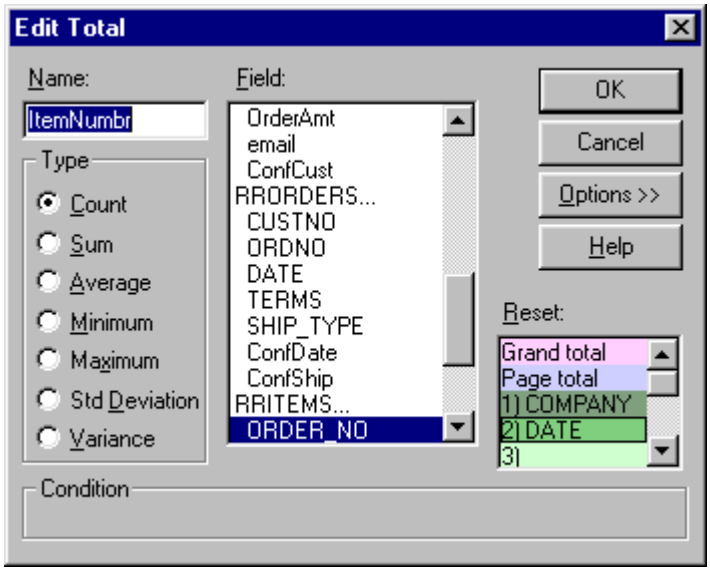
To provide a fairly broad-based set of example, the report will create XML in two ways: 1) by defining calculated fields that place the tags around the data field and 2) by dropping the tags directly onto the report. In the end, the report will look like the image below:



1. Define the tags to hold the company field
    a. Name = ConfCust
    b. Expression = "<ConfCust>" + COMPANY - "</ConfCust>"
2. Define the tags to hold the date field
    a. Name = ConfDate
    b. Expression = "<ConfDate>" + DtoC(DATE) - "</ConfDate>"
3. Define the tags to hold the shipping information
    a. Name = ConfShip
    b. Expression = "<ConfShip>" + SHIP_TYPE - "</ConfShip>"

Drop each of these expressions onto their own header band-lines in the order shown beneath the initial <OrderInfo> tag. Closing out the group header is the tag defining the actual order data. This tag is called <ConfOrder>.

The actual order data is included on the report's record lines. However, because orders can have multiple line items, we need a way to distinguish sets of order values from each other. That is, we don't want, for instance, "quantity" to be included multiple times within a single tag-set without some unique identifier setting them off. This would be an ambiguous situation and could cause issues with the receiving program that has to interpret what's being transmitted.



To resolve this, we'll have R&R automatically create a set of Line Item tags to distinguish each individual line of the order. We'll create a total field called "ItemNumbr" and define it as shown in the dialog box at right. This total field will be used within the calculated field which defines the tags for the line items. We need two of them (one for the start and one for the end). They are defined as shown below:

1. Line Item Start Tag
    a. Name = LineCountB
    b. Expression = "<LineItem" - LTrim(Str(ItemNumbr)) - ">"
2. Line Item End Tag
    a. Name = LineCountE
    b. Expression = "</LineItem" - LTrim(Str(ItemNumbr)) - ">"

As shown on the screen image above, these are placed on record band-lines and "sandwich" the other items that show the detail.

Finally, we'll place the data items that represent the actual things ordered. These will go on individual record band-lines on the report. For illustrative purposes, we will lay these onto the report without first defining a calculated field. This will show that there are multiple ways to accomplish a task within R&R. The table below shows the database field and the XML tags used to bracket it:

Database Field	XML Begin Tag	XML End Tag
rritems.quantity	<ConfQty>	</ConfQty>
rrprices.price	<ConfPrice>	</ConfPrice>
rrprices.descriptn	<ConfDesc>	</ConfDesc>
rrcust.extndprice	<ConfExtnd>	</ConfExtnd>

As indicated on the screen print above, these can be simply dropped onto the report layout. Each of the items following the XML Begin Tag on the individual record band-lines has been set to "auto-trim." This means that the field / text shown will butt directly against the field / text to its

immediate left. This eliminates unwanted spaces from the data, and saves the trouble of lining up fields to begin exactly where the text preceding it ends.

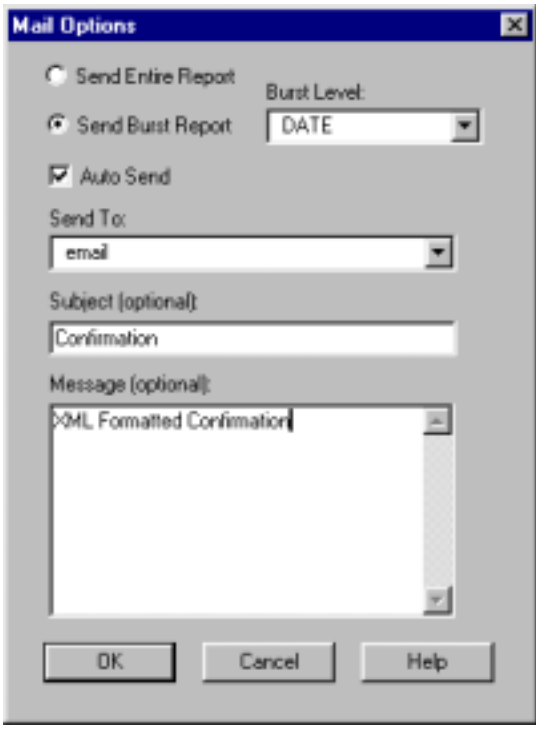
Generating the XML File

We're almost ready to generate our e-mailed XML confirmations to our customers. The remaining steps will clean up formatting issues so that the data generated is free from extraneous characters or formatting.

1. Create a "Generic / Text Only" printer (GTOP)
2. Remove margins from the report
3. Export a text file with MAPI settings for bursting

Because we plan on exporting the entire file, not simply a single band, we must select "text" as our export format, not "text data." However, the printer definition used for the report influences the output somewhat, by allowing or requiring margins / spacing around the "page." Defining a GTOP eliminates these constraints.

To define a GTOP, click on Start / Settings / Printers. Double-click the "Add Printer" icon, select "Next." Define the printer as "Local," select "Next." Under manufacturers, scroll down until you see "Generic" listed. Select it and then, making sure that "Generic / Text Only" is highlighted, select "Next." Define the printer port to be LPT1:, select "Next." Name the printer if desired, do not set it as your default printer. Select "Next." Don't bother with printing a test page, select "Finish."



In your report, go to the Menu and select File / Print. Select the new printer. Choose to preview the report and then close the dialog (click "cancel"). You'll probably notice that your fields have been moved somewhat around the layout to reflect the default font and size for the GTOP. Rearrange things as necessary. When complete, return to the Menu and select File / Page Setup. On the dialog, change all margins to 0. Then click OK.

Finally, define the MAPI settings and generate the exports (note that to burst, you must have a MAPI compatible e-mail program installed on your PC. Outlook / Outlook Express, Eudora and others are some examples.

To burst the documents, go to the Menu and select File / Export / Text. Click on the "Send via MAPI" checkbox and then click the "Mail Options" button. You'll see a dialog box. Complete it as shown above. Then click OK.

Back at the Export dialog, click the “Edit” button. Define the output file / location for the temporary document (be sure to explicitly give it an XML extension). Then click “Export.” R&R will run the report and place each individual document into your outbox as an e-mail with the subject / message defined above and the specific XML document as an attachment. Some e-mail programs might be configured to have you confirm each message before it gets added to your outbox as a safeguard against rogue programs hijacking your address book.

Conclusion

From this paper, it should be apparent that R&R is extremely versatile and can be stretched to accomplish tasks that it isn’t explicitly designed to do. This is a recurring theme throughout R&R, however: Rather than layer discrete features and functions on top of a structure, R&R provides a platform with the capability to handle any number of situations. It might require some cleverness on the end-users part, but it is that person who, in the end, provides value within their company.

Appendix A: XML Saved Data Format from Resultset Browser

```
<?xml version = "1.0" encoding="Windows-1252" standalone="yes"?>
<VFPData>
  <xsd:schema id="VFPData" xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:msdata="urn:schemas-microsoft-com:xml-msdata">
    <xsd:element name="VFPData" msdata:IsDataSet="true">
      <xsd:complexType>
        <xsd:choice maxOccurs="unbounded">
          <xsd:element name="rsbc253">
            <xsd:complexType>
              <xsd:sequence>
                <xsd:element name="descriptn">
                  <xsd:simpleType>
                    <xsd:restriction base="xsd:string">
                      <xsd:maxLength value="20"/>
                    </xsd:restriction>
                  </xsd:simpleType>
                </xsd:element>
                <xsd:element name="quantity">
                  <xsd:simpleType>
                    <xsd:restriction base="xsd:decimal">
                      <xsd:totalDigits value="3"/>
                      <xsd:fractionDigits value="0"/>
                    </xsd:restriction>
                  </xsd:simpleType>
                </xsd:element>
                <xsd:element name="amount">
                  <xsd:simpleType>
                    <xsd:restriction base="xsd:decimal">
                      <xsd:totalDigits value="8"/>
                      <xsd:fractionDigits value="2"/>
                    </xsd:restriction>
                  </xsd:simpleType>
                </xsd:element>
                                .
                                .
                                .
  <rsbc253>
    <descriptn>PC Com</descriptn>
    <quantity>20</quantity>
    <amount>999.00</amount>
    <qty_sub>45</qty_sub>
    <order_sub>2247.75</order_sub>
    <sysdate>2003-03-07</sysdate>
    <pageno>**</pageno>
    <order_tot>999.00</order_tot>
    <price>49.95</price>
  </rsbc253>
  <rsbc253>
    <descriptn>PC Com</descriptn>
    <quantity>20</quantity>
    <amount>999.00</amount>
    <qty_sub>65</qty_sub>
    <order_sub>3246.75</order_sub>
    <sysdate>2003-03-07</sysdate>
    <pageno>**</pageno>
    <order_tot>999.00</order_tot>
    <price>49.95</price>
  </rsbc253>
                                .
                                .
                                .
</VFPData>
```

1. searchwebservices.com via WhatIs.com (www.whatis.com) definition of XML
2. http://gethelp.devx.com/techtips/xml_pro/10min/10min0200/10min0200.asp, "**Maintain Large Databases With XML Servers",** *Kurt Cagle*