Making Quick Reports Compatible with Excel

By Yvan Ayaay, Technical Support Engineer, 4D Inc.

TN 06-08

Introduction

The Quick Report is one of the tools in 4D that allows you to generate reports. One of the report formats that can be generated is an HTML file type. The template used to generate this type of output contains tags that can also be used to generate an XML type output. With an XML output, it can easily be imported into XML-supporting applications, one of which is Microsoft Excel. In this technote, it will be illustrated how to generate an XML output with Quick Report that will be compatible with Microsoft Excel (Excel 2003).

Abstract

The output type generated by Quick Report can be to a printer, to a text file, to an HTML file, to a 4D View or to a 4D Chart. When setting the destination into an HTML file, a template is used to construct the report in HTML format. This template uses set of tags to process the data in order to retain a layout close to the original report or to adapt to a custom HTML layout. The tags used in this template can somehow be manipulated not only to generate HTML file but to generate an XML file as well. Microsoft Excel (Excel 2003) supports XML. It can parse custom XML that follow a certain schema and it can also open XML Spreadsheet, which is its own XML format.

In this technote, the feature of the Quick Report to set the destination to an HTML file will first be discussed. The HTML template used for this feature and some of the functions of this template's tags will be described. Next, it will be shown how these sets of template tags can be used to generate an XML output with Quick Report commands. Two XML formats that are compatible with Microsoft Excel (2003) will be generated. One is the custom XML format and the other is the XML Spreadsheet format (also known as SpreadsheetML) which is Excel's own XML format.

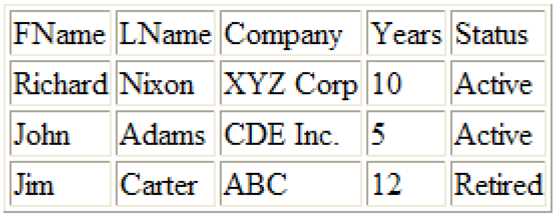
Quick Report Output: HTML File

A report can be generated using the Quick Report editor or programmatically using Quick Report commands. The destination of the output can be to a printer, a text file, an HTML file, a 4D View or 4D Chart. When the output is set to an HTML file, an HTML template that consists of tags is used to build an HTML file. A default HTML template is used but a customized template can be created and used to generate a customized HTML layout.

Below is a sample code that procedurally constructs a Quick Report and generates an HTML file output:

```
`Method: GenerateHTMLFile
C_TEXT(MyTemplate;MyValTex;$path)
C_LONGINT(QRArea)
$path:="myQR.html"
QRArea:=QR New offscreen area ` Create a QR offscreen area
QR SET REPORT TABLE(QRArea;Table(->[CustomerInfo])) `Set Customer table as the current table for
QR area
QR INSERT COLUMN(QRArea;1;->[CustomerInfo]FName) `Insert FName field as 1st column
QR INSERT COLUMN(QRArea;2;->[CustomerInfo]LName) `Insert Lname as 2nd column
QR INSERT COLUMN(QRArea;3;->[CustomerInfo]Company) `Insert Company as 3rd column
QR INSERT COLUMN(QRArea;4;->[CustomerInfo]Years) `Insert Years as 4th column
QR INSERT COLUMN(QRArea;5;->[CustomerInfo]Status) `Insert Status as 5th column
ARRAY REAL($aColumns;1)
$aColumns{1}:=1 `Array for column to order.
ARRAY REAL($aOrder;1)
$aOrder{1}:=-1 ` Array for sort order. Negative 1 for descending and positive 1 for ascnding.
QR SET SORTS(QRArea;$aColumns;$aOrder) ` Set the sort order based on column
QR SET DESTINATION(QRArea;qr HTML file ;$path) ` Set destination to HTML FILE
ALL RECORDS([Customer])
QR RUN(QRArea) `Executes Quick Report.
QR DELETE OFFSCREEN AREA(QRArea) ` Delete QR offscreen area
```

The HTML file that is generated will display data as shown below:



FName	LName	Company	Years	Status
Richard	Nixon	XYZ Corp	10	Active
John	Adams	CDE Inc.	5	Active
Jim	Carter	ABC	12	Retired

The code above uses the default HTML template to create the HTML output. The QR SET HTML TEMPLATE command (http://www.4d.com/docs/CMU/CMU00745.HTM) can be used to set a different or customized HTML template that will be used to generate the Quick Report output. This customized template stored in a text variable passed as parameter to the QR SET HTML TEMPLATE command uses set of tags to handle data in order to construct a customized HTML layout. Moreover, these tags can be manipulated to generate an XML format Quick Report output as well.

HTML Template Tags

Below are some of the HTML template tags used in the later examples and their description:

<!--#4DQRheader--> ... <!--/#4DQRheader--> - The contents included between these tags come from the column titles.

<!--#4DQRrow--> ... <!--/#4DQRrow--> - The contents included between these tags are repeated for each data row.

<!--#4DQRcol--> ... <!--/#4DQRcol--> - The contents included between these tags are repeated for each data column within a row. The order of the columns follows the order in the report. With this tag, a column number (n) can also be specified <4DQRcol;n--> ... <!--/#4DQRcol;n--> to insert data from a certain column number. For instance, <!--#4DQRcol;2--> ... <!--/#4DQRcol;2--> will insert data from the second column.

<!--#4DQRdata--> - This tag will be replaced by the current data for the current cell.

These set of tags can be manipulated to construct an HTML template that will generate an HTML file of a certain display format. To illustrate how to use these tags, a simple table with data as shown below will be considered.

CustomerID	*Name*	*City*
111	John Smith	Seattle
112	Ben Gordon	San Jose

The CustomerID, Name, and City are the column titles of the report. To retrieve these titles, the tags can be used like below:

```
<HTML>
<!--#4DQRHeader-->
      <!--#4DQRCol-->
          <!--#4DQRData-->
       <!--/#4DQRCol-->
<!--/#4DQRHeader-->
</HTML>
```

This should display an HTML result like this:

```
CustomerID
Name
City
```

And to retrieve the data per row in columns, the tags can be used in the order below:

```
<HTML>
<!--#4DQRRow-->
   <!--#4DQRCol-->
     <!--#4DQRData-->
    <!--/#4DQRCol-->
<!--/#4DQRRow-->
<HTML>
```

The HTML output for these tags should look like below when displayed:

```
112
Ben Gordon
```

```
San Jose

111
John Smith
Seattle
```

The HTML template tags can also be manipulated to generate an XML format output. The output can then be easily imported into Microsoft Excel (2003). The next section will discuss how to do this.

Generating an XML Quick Report Output

Microsoft Excel (tested using Office Excel 2003) provides two different kinds of XML functionality: it allows building a spreadsheet from a basic XML file and it allows opening a spreadsheet saved in Excel's own XML format known as XML Spreadsheet. In this section, it will be illustrated how to generate a template that will allow you to build a basic XML file that can be parsed by Microsoft Excel and how to construct a template that will create an XML Spreadsheet or SpreadsheetML that can automatically be opened by MS Excel.

Generating a Basic XML file

Microsoft Excel (2003) can open an XML document that conform to a schema and import data from it. XML (Extensible Markup Language) is a data exchange standard language (for more information about XML, check Technote # 03-48: XML – An Introduction to Extensible Markup Language). Many applications other than Microsoft Excel support this format as well. Generating the output of the Quick Report to be of this type makes it easy to export data into Excel. (***note***: Microsoft Excel 2003 supports custom XML. Check the version of Excel that you're using to see if it is compatible with basic XML.)

The set of HTML tags described in the above section can be used in a customized HTML template to generate a Quick Report output structured in XML format. The process of generating the quick report programmatically will be the same using the HTML File type as destination. The output format depends on the structure of the template used. Thus, most of the customization will be done in the template to generate a well-structured XML file.

A report in XML for the simple Customer table with records as shown below will be generated with Quick Report to show this process.

CustomerID	*Name*	*City*
111	John Smith	Seattle
112	Ben Gordon	San Jose

Below is a sample XML representation of the above data:

```
<?xml version="1.0"  encoding="ISO-8859-1"?>
<Customer>
   <Customer>
      <CustomerID>
           112
      </CustomerID>
      <Name>
           Ben Gordon
      </Name>
      <City>
           San Jose
      </City>
   </Customer>
   <Customer>
      <CustomerID>
           111
      </CustomerID>
      <Name>
           John Smith
      </Name>
      <City>
           Seattle
      </City>
   </Customer>
</Customer>
```

To generate a quick report for the above records for the Customer table, the SampleQR method as shown below can be executed.

```
 `Method:  SampleQR
 `Description:  This method illustrates how to generate a quick report in XML format.
C_TEXT(MyTemplate;MyValText;$path)
C_LONGINT(QRArea)
$path:="XMbasic.xml"
QRArea:=QR New offscreen area `  Create a QR offscreen area
QR SET REPORT TABLE(QRArea;Table(->[Customer])) `Set  Customer table as the current table for QR area
QR INSERT COLUMN(QRArea;1;->[Customer]CustomerID) `Insert  CustomerID as 1st column
QR INSERT COLUMN(QRArea;2;->[Customer]Name) `Insert  Name as 2nd column
QR INSERT COLUMN(QRArea;3;->[Customer]City) `  Insert City as 3rd column
ARRAY REAL($aColumns;1)
$aColumns{1}:=1 `Array  for column to order.
ARRAY REAL($aOrder;1)
$aOrder{1}:=-1 `  Array for sort order. Negative 1 for descending and positive 1 for ascending.
QR SET SORTS(QRArea;$aColumns;$aOrder) `  Set the sort order based on column and in descending order.
QR SET DESTINATION(QRArea;qr HTML file;$path) `  Set destination to HTML File
MyTemplate:=ConstructBasic_XMLTemplate  (QRArea) `  Customize HTML template
QR SET HTML TEMPLATE(QRArea;MyTemplate)
ALL RECORDS([Customer])
```

```
QR RUN(QRArea) `Executes Quick Report.
QR DELETE OFFSCREEN AREA(QRArea) ` Delete QR offscreen area
```

As you can see in the above code, a quick report is programmatically created with the Quick Report destination set to an HTML file. A customized template is set as the current HTML template using the QR SET HTML TEMPLATE command. The text variable “MyTemplate” contains the customized template to structure the XML file output. The ConstructBasic_XMLTemplate method as shown below builds and returns this customized template for the Quick Report area passed to it as parameter.

```
  `Method: ConstructBasic_XMLTemplate
  `Description: This method constructs an HTML template that will generate an XML structured output.
 ` It takes the Quick Report area reference as parameter and returns the customized template
` as text.
C_TEXT($0;$tempText)
C_LONGINT($1;$QRarea;curPos)
C_LONGINT($numCols;$tableNum;$i;$hide;$rep;$size;$numFields;$fieldNum)
C_TEXT($ColName;$obj;$format;$title;$newLine;$fieldobj)

$QRarea:=$1 `Quick Report area reference
$numCols:=QR Count columns($QRarea) ` Count columns in QR area.
$tableNum:=QR Get report table($QRarea) `Get table number of table used in QR area.

$newLine:=Char(13)+Char(10)
$tempText:="" ` template text variable
$tempText:=$tempText+"<?xml version=\"1.0\" encoding=\"<!--#4DQRCharSet-->\"?>"
  `use table name as the root element
$tempText:=$tempText+$newLine+"<"+Replace string(Table name($tableNum);" ";"_")+">"
$tempText:=$tempText+Char(13)+"    <!--#4DQRRow-->"
  `use table name as tag for each row.
$tempText:=$tempText+Char(13)+"    <"+Replace string(Table name($tableNum);" ";"_")+">"

  `Based on the number of columns, insert column names as tags with data for the column in between.
  `This will be repeated per row.
For ($i;1;$numCols)
      `Get information about the column in QR.
    QR GET INFO COLUMN($QRarea;$i;$title;$obj;$hide;$size;$rep;$format)
    $ColName:=$title
    ` retrieve column at a specific column number
    $tempText:=$tempText+Char(13)+"        <!--#4DQRCol;"+String($i)+"-->"
    ` insert column name as tag per column
    $tempText:=$tempText+Char(13)+"        <"+Replace string($ColName;" ";"_")+">"
    ` insert data in the column tag
    $tempText:=$tempText+Char(13)+"            <!--#4DQRData-->"
    $tempText:=$tempText+Char(13)+"        </"+Replace string($ColName;" ";"_")+">"
    $tempText:=$tempText+Char(13)+"        <!--/#4DQRCol;"+String($i)+"-->"
End for
` insert matching close tags.
$tempText:=$tempText+Char(13)+"    </"+Replace string(Table name($tableNum);" ";"_")+">"
$tempText:=$tempText+Char(13)+"    <!--/#4DQRRow-->"
```

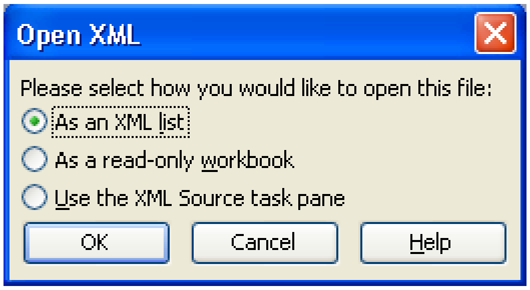
```
$tempText:=$tempText+$newLine+"</"+Replace string(Table name($tableNum);" ";"_")+">"

$0:=$tempText
```

The XML representation of the Customers table data represents each row of data per column as set of child elements. The column names are used as element tags and are repeated for each row of data.

In the ConstructBasic_XMLTemplate method, the XML encoding is first inserted into the template text variable. Then, the name of the table is used as main root element tag. Since each row is enclosed in child elements, the column names and its data are placed within <!--#4DQRrow--> ... <!--/#4DQRrow--> tags so it will be repeated per row. In the said method, the number of columns in the Quick Report area is retrieved to get the column names of the columns which are inserted as tags. This is done in a loop for each column. Within the column tags, the data for each specific column is retrieved by using the <4DQRcol;n--> ... <!--/#4DQRcol;n--> and <!--#4DQRdata--> tags. Each tag is paired with its matching closing tag. The contents within the <!--#4DQRrow--> ... <!--/#4DQRrow--> tags is repeated for each row.

When the SampleQR method is executed, an XML file is generated. When this file is opened with Excel, a dialog as shown below will be displayed. This is usually the case when opening a basic XML document. This dialog does not come up when opening an XML Spreadsheet.



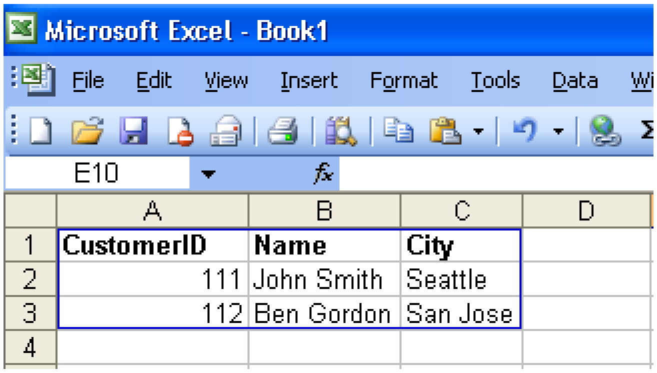
Please select how you would like to open this file:

● As an XML list

○ As a read-only workbook

○ Use the XML Source task pane

When you choose the “As an XML list” option, the data is imported as shown below:



	A	B	C	D
1	CustomerID	Name	City	
2	111	John Smith	Seattle	
3	112	Ben Gordon	San Jose	
4				

Generating an XML Spreadsheet

Starting with Microsoft Excel XP, a new XML functionality has been introduced in Excel with the use of SpreadsheetML or XML Spreadsheets. This is Excel’s own XML format.

This allows Excel to open this type of document automatically just like it opens .xls or excel files. No parsing like in basic XML above is being done when this type of document is opened.

The simple Customer table with its data as shown below will once again be used to illustrate how to generate this type of XML format.

CustomerID	*Name*	*City*
111	John Smith	Seattle
112	Ben Gordon	San Jose

Below is a sample XML Spreadsheet for the above Customer table data.

```
<?xml version="1.0"?>
<?mso-application progid="Excel.Sheet"?>
<Workbook xmlns="urn:schemas-microsoft-com:office:spreadsheet"
 xmlns:o="urn:schemas-microsoft-com:office:office"
 xmlns:x="urn:schemas-microsoft-com:office:excel"
 xmlns:ss="urn:schemas-microsoft-com:office:spreadsheet"
  xmlns:html="http://www.w3.org/TR/REC-html40">
 <DocumentProperties xmlns="urn:schemas-microsoft-com:office:office">
   <Author>IVAN</Author>
  <LastAuthor>Yvan Ayaay</LastAuthor>
   <Created>2006-02-20T06:13:35Z</Created>
  <Company>NONE</Company>
   <Version>11.5606</Version>
  </DocumentProperties>
 <OfficeDocumentSettings xmlns="urn:schemas-microsoft-com:office:office">
  <DownloadComponents/>
  <LocationOfComponents HRef="file:///D:\F__\"/>
  </OfficeDocumentSettings>
 <ExcelWorkbook xmlns="urn:schemas-microsoft-com:office:excel">
   <WindowHeight>9210</WindowHeight>
   <WindowWidth>15195</WindowWidth>
  <WindowTopX>0</WindowTopX>
  <WindowTopY>30</WindowTopY>
   <ProtectStructure>False</ProtectStructure>
   <ProtectWindows>False</ProtectWindows>
  </ExcelWorkbook>
 <Styles>
  <Style ss:ID="Default" ss:Name="Normal">
   <Alignment ss:Vertical="Bottom"/>
   <Borders/>
   <Font/>
   <Interior/>
   <NumberFormat/>
   <Protection/>
   </Style>
  </Styles>
 <Worksheet ss:Name="Customer">
  <Table ss:ExpandedColumnCount="3" ss:ExpandedRowCount="3" x:FullColumns="1"
   x:FullRows="1">
   <Column ss:AutoFitWidth="0" ss:Width="56.25"/>
  <Row>
   <Cell><Data ss:Type="String">CustomerID</Data></Cell>
   <Cell><Data ss:Type="String">Name</Data></Cell>
   <Cell><Data ss:Type="String">City</Data></Cell>
   </Row>
   <Row>
   <Cell><Data ss:Type="Number">112</Data></Cell>
   <Cell><Data ss:Type="String">Ben Gordon</Data></Cell>
   <Cell><Data ss:Type="String">San Jose</Data></Cell>
   </Row>
   <Row>
   <Cell><Data ss:Type="Number">111</Data></Cell>
   <Cell><Data ss:Type="String">John Smith</Data></Cell>
   <Cell><Data ss:Type="String">Seattle</Data></Cell>
   </Row>
  </Table>
  <WorksheetOptions xmlns="urn:schemas-microsoft-com:office:excel">
   <Selected/>
   <ProtectObjects>False</ProtectObjects>
   <ProtectScenarios>False</ProtectScenarios>
  </WorksheetOptions>
  </Worksheet>
</Workbook>
```

As you can see, the spreadsheet begins with XML declaration and processing information followed by the metadata about the document. The content between the Worksheet elements is where the data is manifested. The HTML Template tags will be manipulated to dynamically insert data as they should appear within the Worksheet tags.

To generate an XML Spreadsheet output, the SampleQR method in the above section could still be used but instead of calling ConstructBasic_XMLTemplate method to construct the template, the ConstructXMLSpreadSheet method is called:

```
`Method: SampleQR
……..
QR SET DESTINATION(QRArea;qr HTML file ;$path)  ` Set destination to HTML File
MyTemplate:=ConstructXMLSpreadSheet (QRArea)  ` Customize HTML template
QR SET HTML TEMPLATE(QRArea;MyTemplate)
…………..
```

The ConstructXMLSpreadSheet method, as shown below, creates an HTML template that builds an XML SpreadSheet output.

```
 `Method: ConstructXMLSpreadSheet
 `Description: This method creates a template that builds an XML Spreadsheet output. The template is
returned as a text variable.

C_TEXT($0;$tempText)
C_LONGINT($1;$QRarea;curPos;$numRecords)
C_LONGINT($numCols;$numRows;$tableNum;$i;$hide;$rep;$size;$numFields;$fieldNum)
C_TEXT($ColName;$obj;$format;$title;$newLine;$fieldobj;mytype;$metadataTemplate)
C_POINTER($tablePtr)

$QRarea:=$1
$numCols:=QR Count columns($QRarea) ` Count columns in QR
$tableNum:=QR Get report table($QRarea) `Get table number of current table in QR
$numFields:=Count fields($tableNum) `Count number of fields in table
$tablePtr:=Table($tableNum)
$numRecords:=Records in selection($tablePtr->) `Get number of records in selection

ARRAY TEXT(asFields;Count fields($tableNum))
For ($vlField;1;Size of array(asFields)) ` get all fieldnames of the fields in the table and save in array
        asFields{$vlField}:=Field name($tableNum;$vlField)
End for
  `insert text in template text variable starting with Worksheet element tag
$tempText:="<Worksheet ss:Name=\""+Replace string(Table name($tableNum);" ";"_")+"\">"
  `declare number of columns for the worksheet
$tempText:=$tempText+Char(13)+" <Table ss:ExpandedColumnCount=\""+String($numCols)+"\"
ss:ExpandedRowCount=\""+String($numRecords+1)+"\" x:FullColumns=\"1\" x:FullRows=\"1\">"
  `Insert the column titles
$tempText:=$tempText+Char(13)+"<!--#4DQRHeader--><Row><!--#4DQRCol--><Cell><Data
ss:Type=\"String\"><!--#4DQRData-"+"-></Data></Cell><!--/#4DQRCol--></Row><!--/#4DQRHeader-->"
$tempText:=$tempText+Char(13)+"<!--#4DQRRow--><Row>"
  `Insert column data for each row.
For ($i;1;$numCols)
        QR GET INFO COLUMN($QRarea;$i;$title;$obj;$hide;$size;$rep;$format) ` get column
information
        curPos:=Position("]";$obj)
        $fieldobj:=Substring($obj;curPos+1) `$fieldobj contains field name
        $fieldNum:=Find in array(asFields;$fieldobj) `find field number
        GET FIELD PROPERTIES($tableNum;$fieldNum;$fieldType) `determine the type of field
        mytype:=GetType ($fieldType) `GetType function returns the type of data of the field type.
        $ColName:=$title ` title of the column
        $tempText:=$tempText+Char(13)+"<!--#4DQRCol;"+String($i)+"-->" ` content of column at
certain column number will be retrieved
        $tempText:=$tempText+Char(13)+"<Cell><Data ss:Type=\""+mytype+"\">" ` declare data type
        $tempText:=$tempText+Char(13)+"<!--#4DQRData-->" ` insert data in column
        $tempText:=$tempText+Char(13)+"</Data></Cell> <!--/#4DQRCol;"+String($i)+"-->" `insert
matching closing tags
End for
  `insert the closing tags of the lower part of the XMLSpreadsheet
$tempText:=$tempText+Char(13)+"</Row><!--/#4DQRRow-->"
$tempText:=$tempText+Char(13)+"</Table>"
$tempText:=$tempText+Char(13)+"<WorksheetOptions xmlns=\"urn:schemas-microsoft-
com:office:excel\">"
$tempText:=$tempText+Char(13)+"<Selected/>"
$tempText:=$tempText+Char(13)+"<ProtectObjects>False</ProtectObjects>"
$tempText:=$tempText+Char(13)+"<ProtectScenarios>False</ProtectScenarios>"
$tempText:=$tempText+Char(13)+"</WorksheetOptions>"
$tempText:=$tempText+Char(13)+"</Worksheet>"
$tempText:=$tempText+Char(13)+"</Workbook>"
```

```
$metadataTemplate:=ReadTemp ("XMLtemp2.txt") `ReadTemp reads a file that contains a template for the upper part of the XML Spreadsheet.
$0:=$metadataTemplate+$tempText ` Concatenate the upper part and the lower part of the XML Spreadsheet to be returned as template.
```

In the ConstructXMLSpreadsheet method, the content between the WorkSheet elements of the XML Spreadsheet is dynamically created. The metadata portion (upper portion from the top of the sample XML spreadsheet down to the top of the WorkSheet element tag) of the XML Spreadsheet is stored in a file and is retrieved and concatenated with the constructed lower portion at the bottom of the method. The content between the WorkSheet elements tags as shown below is where the data (including the column titles) are enclosed.

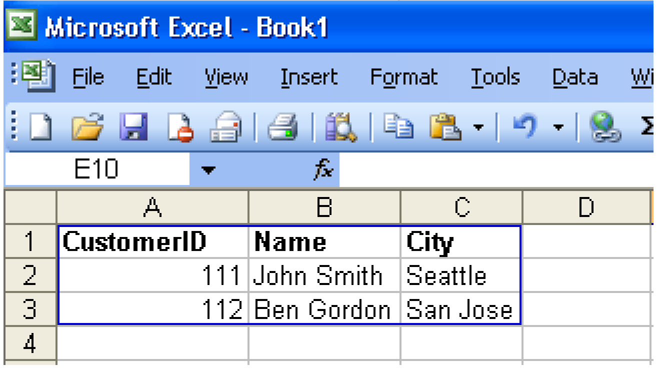
```
……………………
<Worksheet ss:Name="Customer">
  <Table ss:ExpandedColumnCount="3" ss:ExpandedRowCount="3" x:FullColumns="1"
   x:FullRows="1">
   <Column ss:AutoFitWidth="0" ss:Width="56.25"/>
   <Row>
    <Cell><Data ss:Type="String">CustomerID</Data></Cell>
    <Cell><Data ss:Type="String">Name</Data></Cell>
    <Cell><Data ss:Type="String">City</Data></Cell>
   </Row>
   <Row>
    <Cell><Data ss:Type="Number">112</Data></Cell>
    <Cell><Data ss:Type="String">Ben Gordon</Data></Cell>
    <Cell><Data ss:Type="String">San Jose</Data></Cell>
   </Row>
   <Row>
    <Cell><Data ss:Type="Number">111</Data></Cell>
    <Cell><Data ss:Type="String">John Smith</Data></Cell>
    <Cell><Data ss:Type="String">Seattle</Data></Cell>
   </Row>
  </Table>
…………………………………………
………………………………………
```

As you can see, the first row in the worksheet contains the column titles. In the ConstructXMLSpreadsheet, after inserting the Worksheet and Table tags to the template text variable, the header tag <!--#4DQRheader--> ... <!--/#4DQRheader--> is inserted. The content within the header tags allows you to retrieve column titles. To retrieve the title, the <!--#4DQRData--> tag is used. And to get all the column titles, this tag is enclosed within the <!--#4DQRcol--> ... <!--/#4DQRcol--> tags. Since the title is enclosed in the Cell and Data tags as you can see in the worksheet section, the Cell and Data tags and their matching closing tags are also enclosed in the column tag which is enclosed in the header tag. In this row, the column titles are repeated as cells within these tags.

After the first row in the worksheet, the data per column is listed per row. Depending on the type of the data, the data is declared in the Data tag. In order to construct the

data as they appear in the worksheet, the type of data should be dynamically detected. To do this in the ConstructXMLSpreadsheet method, the field number of column is initially retrieved to get its property. And based on this, a function is called (GetType function) that will return the data type either as "Number" or "String".
To traverse all the columns of the Quick Report, the number of columns in QR area is first retrieved and then each column is accessed within a loop as shown in the ConstructXMLSpreadsheet method. This is done after inserting the row for the column title. Within the loop, information about the column is retrieved that will allow you to retrieve the field number and check the field property. The GetType function is called to get the type of field.
In the said method, in order to retrieve information for each row of data, the for loop is enclosed within the <!--#4DQRrow--> ... <!--/#4DQRrow--> tags which will repeat the contents within it for every row. Using <!--#4DQRData--> tag within this tag and enclosing it within <4DQRcol;n--> ... <!--/#4DQRcol;n > tags allows you to retrieve each data per particular column (n). Enclosing the Cell and Data tags and their matching closing tags within the column tags will repeat these tags per data of each column. The type of data is declared dynamically as shown in the ConstructXMLSpreadsheet method.
After the insertion of the tags to retrieve the data in the ConstructXMLSpreadsheet method, the matching closing tags for the Worksheet tags are inserted. At this point, the template text variable contains the lower part of the XML Spreadsheet. The file that contains the upper part is read from a file to a text variable and then concatenated with the lower part. The entire template is then passed as the returned text in the method.

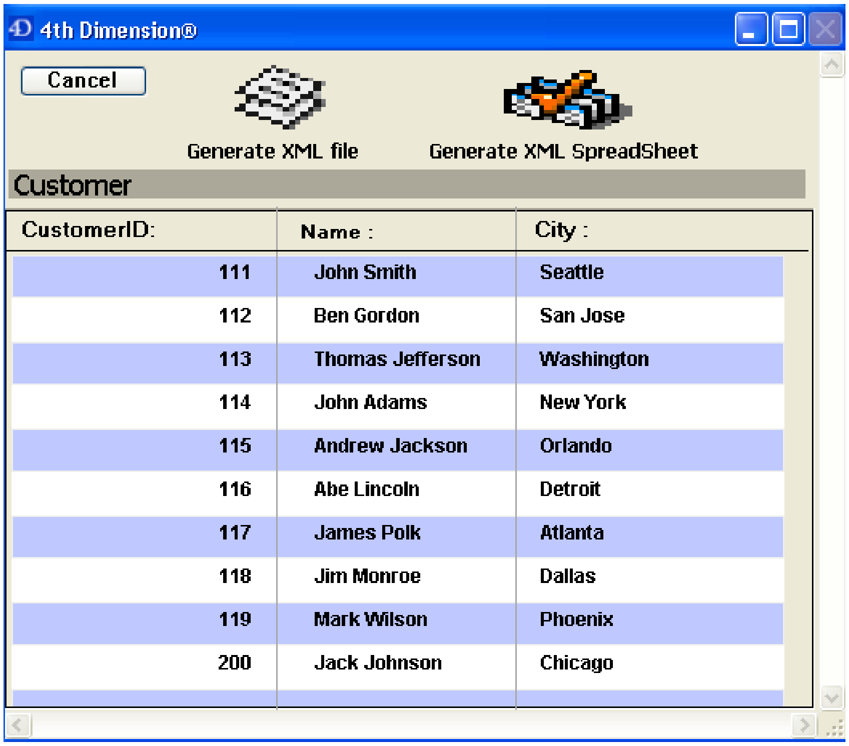
When the sampleQR method is executed using this template, an XML spreadsheet will be created. It can be automatically be opened by Microsoft Excel that looks like the output as shown below. No prompt is displayed when this file is opened as it is treated like an .xls or excel file. (Again, this has been tested using Microsoft Excel 2003).



	A	B	C	D
1	CustomerID	Name	City	
2	111	John Smith	Seattle	
3	112	Ben Gordon	San Jose	
4				

Sample Database

The sample database that comes with this technote shows the two methods in generating an XML output that will be compatible with Microsoft Excel. When the database is first loaded, a dialog as shown below is displayed:



CustomerID:	Name :	City :
111	John Smith	Seattle
112	Ben Gordon	San Jose
113	Thomas Jefferson	Washington
114	John Adams	New York
115	Andrew Jackson	Orlando
116	Abe Lincoln	Detroit
117	James Polk	Atlanta
118	Jim Monroe	Dallas
119	Mark Wilson	Phoenix
200	Jack Johnson	Chicago

When you click on the “Generate XML file” button , the list of records is generated as an XML file. And when you click the “Generate XML SpreadSheet” button, the list is generated as an XML Spreadsheet. The name of the generated files and their location are displayed in an Alert dialog after the click. The Quick Report (done programmatically) is used to generate these types of output.

Conclusion

The Quick Report with the output destination set to HTML file allows you to generate an XML output that can be read by Microsoft Excel (2003). Excel 2003 supports standard XML and its own XML format known as XML SpreadSheet. You can generate an output that Excel supports with Quick Report by customizing HTML template used when the destination of QR output is set to HTML file. This template is consisting of tags that make it possible to dynamically construct an XML output.