Web-Based Reporting with R&R

A Liveware Publishing, Inc. White Paper

26 April 2001
Christian A. Strasser
CTO – Liveware Publishing, Inc.

Introduction

R&R is an extremely versatile tool. Both the xBase and SQL versions of the product provide a large number of ways to communicate database information. From licensing and installing individual copies of the report designer to integrating the ActiveX component into a custom application, the choices available are extremely robust and rich.

One of the biggest opportunities to appear in recent years, is reporting over the Internet. There are many uses for this method of distribution. One is Electronic Statement Presentation (ESP), a key feature provided by many organizations to their customers which lets them login and display their specific account data through their browser. Another is providing broad distribution of reports to a variety of people. Certain jobs, uses and locations of people needing reports make it impractical for them to get access to their reports in traditional ways. These people can take advantage of their Internet browser and a connection to the Internet to access dynamic, parameterized reports from wherever they are.

This White Paper will describe the requirements and the tools needed to prototype a web-based reporting application using R&R as the engine to create the reports. Many topics will be covered, including web-server configuration, web-page construction, cgi execution and proper setup of the reports themselves to deliver the requested results.

Note that this paper will focus on a Windows-based approach to the problem *on the server*. This is because R&R is a Windows application and will not run on other platforms. This solution revolves around R&R sessions being controlled and executed as a server-side application. Note however, that the database does not have to be a Windows application (for the R&R SQL only), nor do the clients have to be Windows. This is because ODBC calls can be made to UNIX systems, mainframes, Macintoshes, etc. And because the results of the request to R&R are pure HTML, any browser on any operating system can interpret the results.

Although a lot of information is presented here, don't be intimidated. We have found that an R&R-based web report server can be implemented with only a few days' worth of effort. Most of the time spent will go into designing the front-end for the browser and in building the reports.

Overview

As noted above, there are several components to a web-based reporting solution which must all work together to deliver the result. These are:

- A front-end HTML page to allow selection of the desired report and to specify the parameters to send into the report.
- A properly-configured web server that has a directory for storing and executing cgi scripts, a place to save the reports and the results and appropriate rights to those areas. Any package will do. Typically, most Windows NT networks will use Microsoft's IIS. This paper will show the configuration for Microsoft's Personal Web Server product which comes bundled with Windows98. Undoubtedly, other web servers will work and will probably be quite similar in practice.
- An executable program stored on the web server which is responsible for 1) accepting the input from the web page, 2) parsing the input stream into its component pieces (i.e. individual parameters and the report being requested), 3) defining the right property settings in the R&R ActiveX control, 4) executing the report and 5) feeding the HTML output file back to the requestor. This program is known as the "CGI Script." Typically, it will be written in Visual Basic or Visual C. Both of these languages produces relatively lightweight code for rapidly executing a server-side process. Code for this is shown in Appendix 2.
- The R&R ActiveX runtime control (either or both of the xBase and SQL controls). This requires that a copy of R&R be installed on each server that will be responsible for feeding reports back to requesters. This ensures that the proper DLLs and Registry settings are copied to and made on the server.
- The set of R&R reports that will be executed to provide the information back to the person requesting the report. These reports can be either .RRWs or .RSWs, and must be defined with the appropriate RIPARAM calculated fields to accept parameters from the input source.
- Accessible data from the server's point of view. That is, the client does not have to be able to directly access the data used in the report, but the server does. This can be accomplished by having an ODBC connection with the proper information defined on the server (SQL), or having the specific .dbf files visible and mapped somehow from the server (xBase).

In brief, the process is this: the client's browser accesses a web-page which contains a means to select a report and provide parameters into the report to control the output or query in some fashion. When the "submit" button is pressed, it instructs the web server to execute a named CGI script program which parses the input stream passed to it by the web-page. The R&R ActiveX Component is part of this CGI, and based on the report passed and the parameters parsed, is executed with the appropriate properties set.

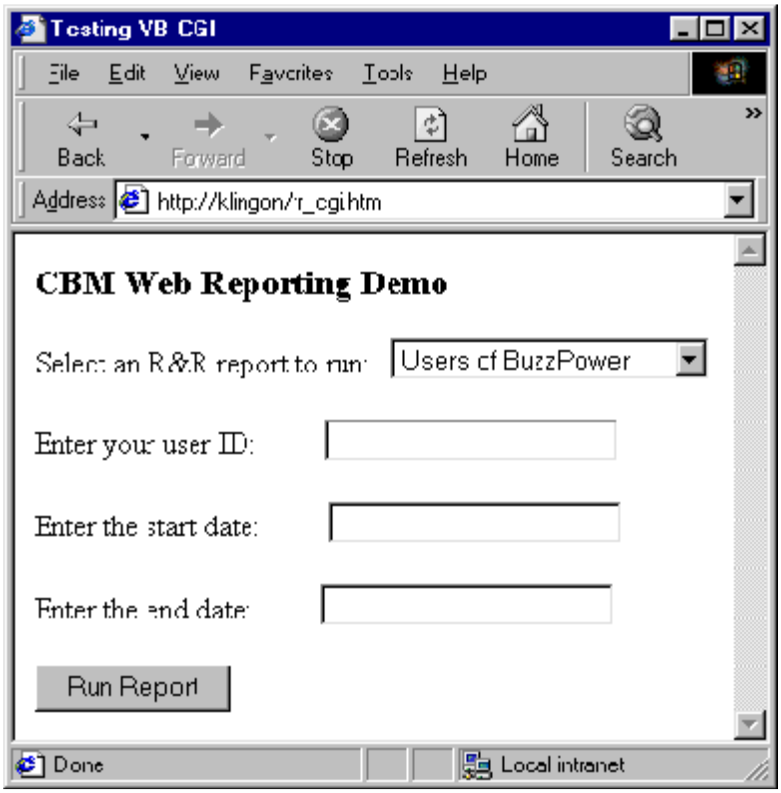
HTML

The purpose of the HTML page or pages is to accept the input for the report and allow parameters to be collected and passed to the server. This becomes the user interface to the reporting application. Thus, the developer has to decide the appropriate controls to pick the report and define the parameters that make sense for the particular report selected. This could be a simple one-page combo-box and a few input boxes as shown in the example here, or as complex as a set of linked pages, each of which allows a different set of parameters along with graphics, instructions, etc.

The "action" section of the page defines the name of the program and where it is stored that will be executed when the "Run Report" button is pressed. In this case, The program is called rr_cgi.exe and it is stored in the cgi-bin section of the web server. The rr_cgi.exe program will be explained in its own section of this white paper.

```
<HTML>
<HEAD>
<TITLE>Sample VB CGI</TITLE>
</HEAD>

<BODY>
<H3>CBM Web Reporting Demo</H3>
<P></P>
<P></P>
<FORM action="http://klingon/cgi-bin/cgi-bin/rr_cgi.exe" method="post">
Select an R&amp;amp;R report to run:  
<SELECT NAME="report">
        <OPTION VALUE="users" selected>Users of BuzzPower</OPTION>
        <OPTION VALUE="lag">Posting Lag Time</OPTION>
        <OPTION VALUE="cbm01">CBM Test Report 1</OPTION>
        <OPTION VALUE="cbm02">CBM Sales Report</OPTION>
</SELECT>
<P></P>Enter your user ID:        
<INPUT id=logid name=userid>
<P></P>Enter the start date:        
<INPUT id=begdate name=begdte>
<P></P>Enter the end date:        
<INPUT id=enddate name=enddte>
<BR>
<P></P><P></P>
<INPUT TYPE="submit" VALUE="Run Report">
</FORM>
</BODY>
</HTML>
```

The various reports available for this simple example are enumerated in the "select" portion of the web-page. In this case, each is explicitly listed for selection within a drop-down combo box. The "value" item specifies the name of the report that will be submitted to the rr_cgi.exe program, while the text following it describes the report name that is visible to the person browsing the page. Thus, if one picks the report shown as "CBM Test Report 1," the submission action will pass "cbm01" as part of the input stream to the rr_cgi.exe program



As shown in the screen shot at right, the user may submit additional information besides the specific report to run. This example included a user-id for validation and restriction of the results available from the database as well as start and end dates to allow reporting over specific, arbitrary periods (month, year, quarter, etc.).

These three additional values will be passed to the rr_cgi.exe program as the parameters "userid," "begdte," and "enddte."

As indicated above, one can easily imagine more extensive and automatic means of selecting reports for execution. The names of reports could be stored in a database that the web-page queries and displays dynamically as reports are added/deleted. The parameters as well could be variable depending on the specific report in question. The user could also be directed to a variety of web pages via hot links that would explain the report and the various parameters available. On this example, the page was stored as a standalone within a browsable list on the home page.

Web Server

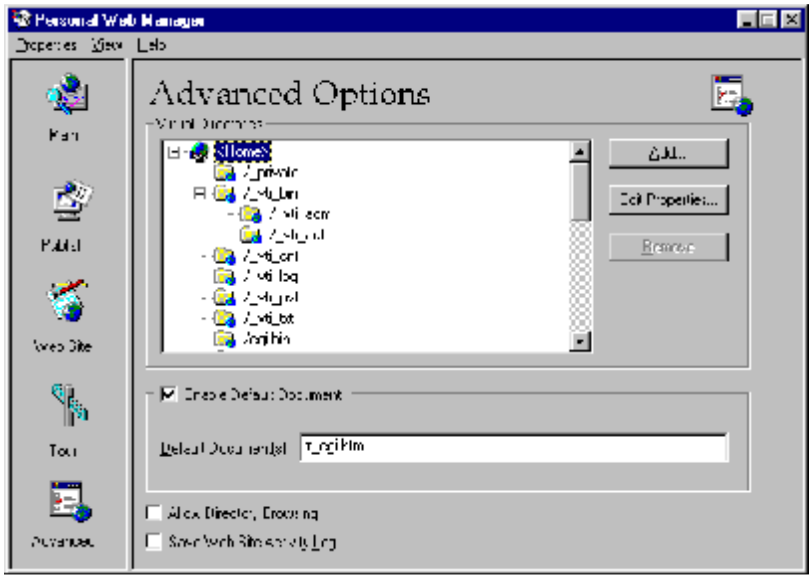
The sample application shown here has been setup using Microsoft's Personal Web Server (PWS) under Windows98(tm). We have also configured a similar application on IIS running under Windows NT(tm) as well. All configuration information will be presented in the context of PWS. A full treatment and description of PWS is beyond the scope of this white paper.

Run PWS by selecting Start | Programs | Accessories | Internet Tools | Personal Web Server | Personal Web Manager. Note that unless you have added this component to your Windows98 system, it will not be available on this path.

PWS must be started to enable serving of web pages to a client. On the "Main" tab, click on the "Start" button.

Certain settings to enable CGI execution and placement of the basic information needed for the system should also be configured at this time.

Advanced Options - Home

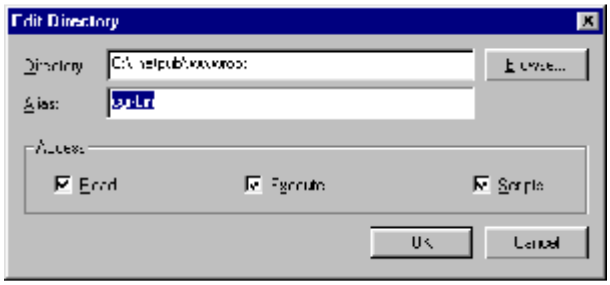


In the example given at right, the default document is the one shown in the HTML section above. In a live application, the default document would be your home page and one of the links would eventually lead down to the report selection page. The document shown here is stored in the <Home> directory.

Advanced Options- Scripts

To execute the CGI referenced in rr_cgi.htm, a /cgi-bin folder must be created under the <Home> folder (if it doesn't already). Click "Add" and

define the popup dialog as shown below. You need to define a "real" directory as well as a "virtual" directory (alias) for this section of your website. The cgi-bin folder must be tagged with all three "Access" options as shown at left to ensure that the program can be executed.



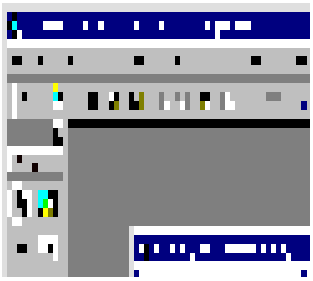
Advanced Options- Reports

Other folders must be created as well to ensure that the application will work. A "reports" virtual directory must be setup with at least "read" access. This folder is the place where your RRW files will be stored for access by the CGI. Beneath the "real" reports folder on your server system, you must define a folder to store the HTML files that will be generated by the CGI and served back to the client. In this example, the actual folder name on the server is: c:\inetpub\wwwroot\rr-repts\temphtmls. This folder is referenced within the VB program stored in the cgi-bin folder. It doesn't have to be a virtual web folder because it never gets accessed directly from a web client. However, there is no reason why a structure couldn't be created wherein individual clients' output could be stored and accessed for future retrieval. Security and timeliness of the data becomes a consideration at tthat point. Additionally, depending on volume, this folder could become very full and require maintenance to delete old .htm files.

Visual Basic Program

The key piece that ties all the components together is a relatively simple Visual Basic program that sits on the server and waits for a report request to come to it from the client's browser. This program incorporates the R&R ActiveX control to gather the parameters submitted from the web and pass them onto the reports that are executed on the server against the internal database.

The VB6 program requires some special variables and definitions that are needed to communicate with IIS and a the client. Microsoft's website offers a variety of useful technical documents to enable CGI execution. Our example relies heavily on KnowledgeBase Article Q239588. In it, the necessary initialization components for a VB CGI executable are defined. For convenience, the salient contents of this article are reprinted here with additional information needed for R&R (see next page).



Once the header information has been defined, the real work of the CGI can be done. Note that the project must be created by dropping the appropriate R&R ActiveX control onto a form in a VB project. A typical form is shown at right. The yellow icon is the R&R SQL ActiveX control and is named RSReport1, while the other icon is for xBase reports and named RRReport1.

The button is presented only as a place-holder for execution. It never actually gets "clicked" but is used in testing. This raises an important point: Obviously, on a server, no user input is possible. If any user-input dialog was displayed on the server, the client would perceive the application as "hung" because no action could be taken until someone at the server site refreshed the screen and closed the dialog.

The code attached to the Command1 button is shown at left. The only line that is necessary is the method to actually invoke the control. The example at left runs the SQL version. To execute the xBase, replace RSReport1.RunReport(1) with RRReport1.RunReport(1).

```
Option Explicit

Sub Command1_Click()
    RSReport1.RunReport (1) ' Run synchronously
End Sub

Private Sub Form_Load()

End Sub
```

```
Attribute VB_Name = "cgiMain"
'     *****************
'     *  R&R via CGI  *
'     *****************
' Note: Set the VB project options to use Sub Main as the startup form.
'
' Author:   Colin Strasser
'           September 8, 2000
'
' Based on code from Microsoft KnowledgeBase Article Q239588.
'--------------------------------------------------------------------

Option Explicit

Public Const STD_INPUT_HANDLE = -10&
Public Const STD_OUTPUT_HANDLE = -11&

Public Const CGI_AUTH_TYPE         As String = "AUTH_TYPE"
Public Const CGI_CONTENT_LENGTH    As String = "CONTENT_LENGTH"
Public Const CGI_CONTENT_TYPE      As String = "CONTENT_TYPE"
Public Const CGI_GATEWAY_INTERFACE As String = "GATEWAY_INTERFACE"
Public Const CGI_HTTP_ACCEPT       As String = "HTTP_ACCEPT"
Public Const CGI_HTTP_REFERER      As String = "HTTP_REFERER"
Public Const CGI_HTTP_USER_AGENT   As String = "HTTP_USER_AGENT"
Public Const CGI_PATH_INFO         As String = "PATH_INFO"
Public Const CGI_PATH_TRANSLATED   As String = "PATH_TRANSLATED"
Public Const CGI_QUERY_STRING      As String = "QUERY_STRING"
Public Const CGI_REMOTE_ADDR       As String = "REMOTE_ADDR"
Public Const CGI_REMOTE_HOST       As String = "REMOTE_HOST"
Public Const CGI_REMOTE_USER       As String = "REMOTE_USER"
Public Const CGI_REQUEST_METHOD    As String = "REQUEST_METHOD"
Public Const CGI_SCRIPT_NAME       As String = "SCRIPT_NAME"
Public Const CGI_SERVER_NAME       As String = "SERVER_NAME"
Public Const CGI_SERVER_PORT       As String = "SERVER_PORT"
Public Const CGI_SERVER_PROTOCOL   As String = "SERVER_PROTOCOL"
Public Const CGI_SERVER_SOFTWARE   As String = "SERVER_SOFTWARE"

Public Declare Function Sleep Lib "kernel32" _
(ByVal dwMilliseconds As Long) As Long

Public Declare Function stdin Lib "kernel32" Alias "GetStdHandle" _
(Optional ByVal Handletype As Long = STD_INPUT_HANDLE) As Long

Public Declare Function stdout Lib "kernel32" Alias "GetStdHandle" _
(Optional ByVal Handletype As Long = STD_OUTPUT_HANDLE) As Long

Public Declare Function ReadFile Lib "kernel32" _
(ByVal hFile As Long, ByVal lpBuffer As Any, ByVal nNumberOfBytesToRead As Long, _
lpNumberOfBytesRead As Long, Optional ByVal lpOverlapped As Long = 0&) As Long

Public Declare Function WriteFile Lib "kernel32" _
(ByVal hFile As Long, ByVal lpBuffer As Any, ByVal nNumberOfBytesToWrite As Long, _
```

The next section of the program defines the main() module. This contains

the initialization required to setup all the variables needed for the program to properly communicate with the webserver and through to R&R. Each of the important variables used in the program will be described below. Also shown here is the sample input stream typical of what would be sent to the program from the webpage shown above

```
sReadBuffer = "report=cbm01&userid=bearman&begdte=07%2F01%2F2001&enddte=07%2F31%2F2001"
```

```
Sub Main()
    Dim sReadBuffer As String
    Dim sWriteBuffer As String
    Dim lBytesRead As Long
    Dim lBytesWritten As Long
    Dim hStdIn As Long
    Dim hStdOut As Long

    Dim iPos1 As Integer
    Dim iPos2 As Integer
    Dim iPos3 As Integer
    Dim iPos4 As Integer

    Dim iAmp1 As Integer
    Dim iAmp2 As Integer
    Dim iAmp3 As Integer
    Dim iAmp4 As Integer

    Dim sArgName As String
    Dim sArgValue As String

    Dim sReportToRun As String
    Dim sUserID As String
    Dim sBegDte As String
    Dim sEndDte As String

    Dim iLP1 As Integer
    Dim iLP2 As Integer
    Dim iLP3 As Integer
    Dim iLP4 As Integer

    Dim sPm1 As String
    Dim sPm2 As String
    Dim sPm3 As String

    Const sReportPath = "C:\inetpub\wwwroot\rr-repts\"
    Const sTempPath = "c:\inetpub\wwwroot\rr-repts\TempHTMLs\"
    Const sParm1 = "report="
    Const sParm2 = "userid="
    Const sParm3 = "begdte="
    Const sParm4 = "enddte="

    Dim sFullOutputFileSpec As String
```

sReadBuffer
String variable that holds the input stream submitted from the parameters webpage shown earlier.

sWriteBuffer
String variable that holds the specially formatted text that is sent back out to the webserver after successful execution of the report

lBytesRead
Logical variable to indicate whether any data was transmitted from the client's browser

lBytesWritten
Logical variable that indicates whether any data was sent back to the browser

hStdIn
Special file handle variable used to enable reading from the console

hStdOut
Special file handle variable used to enable writing back to the console

iPos1 – iPos4
Integer variables needed to identify where the text stored in the various constant parameters (sParm1 – sParm4) lie within the text submitted through sReadBuffer

iAmp1 – iAmp4
The individual data elements submitted from the web browser are separated by ampersand "&"characters. These are integer variables used to indicate the

position of the ampersands in the sReadBuffer text. The data associated with each parameter begins immediately following the "=" sign and extends to just before the ampersand

iLP1 - iLP4
Integer variables that hold the total length of each of the four parameter constants identified above. These will be used to cull the name of the variable represented by each of the parameters (without the "=" sign)

sArgName
Temporary string variable needed to hold the name of the argument associated with each parameter (i.e. "report=" in sParm1 will be stored in sArgName as "report"

sArgValue
Temporary string variable needed to hold the value submitted for each individual element submitted through the webserver textstream. Referring to the sample input stream above, the value associated with the name "report" is "lag"

sReportToRun, sUserID, sBegDte, sEndDte
Each of these variables is specifically defined to be associated with one of the four values submitted through the input stream. This is important to note: All examples shown here are extremely hard-coded to expect four values to be submitted from the webserver with no variation as to name, order or type. Reports that had different requirements (i.e. different types of variables, more variables, etc.) would have to be accounted for by ensuring that every circumstance or situation of variables was handled. Liveware is developing a utility that simplifies this definition and administration on the server side.

sReportPath, sTempPath
These are constants that point to the physical directories on the server where the reports are stored and where the HTML output generated by R&R will be stored. These should correspond exactly to the pathnames described in the WebServer section of this document.

sFullOutputFileSpec
String variable used to indicate the entire path and filename where the temporary HTML export will be stored.

sOutputFileName
Optional string used to force all output to a single filename. This isn't recommended if any volume at all is expected on the website, since output would soon become confused and overwritten. Instead, a filename composed of the UserID (submitted in the input stream) appended with ".HTM" is a better choice, since each user will likely only be submitting and reviewing one report at a time.

sPm1 - sPm3
These are string variables that will correspond one-for-one with the values defined within the report in RIPARAM functions. The will consist of the character string defined in each RIPARAM followed by "=" and then the value submitted and parsed in the input stream (sUserID, sBegDte, sEndDte)

The Code

This section begins to detail the actual mechanics of the VB6 program. Each major section will be taken step-by-step and its function explained. Some pieces will have little meaning without a solid understanding of the

report specification being accessed. This will be addressed in the next section. A complete listing of the program is provided in the appendix at the end of this paper.

```
iPos2 = InStr(sReadBuffer, sParm2)
iAmp2 = InStr(iAmp1 + 1, sReadBuffer, "&")
sArgName = Left$(sReadBuffer, iLP2 - 1)
sArgValue = Mid$(sReadBuffer, iPos2 + iLP2, (iAmp2 - (iLP2 + iPos2)))
sUserID = Trim(sArgValue)

sFullOutputFileSpec = sTempPath & sUserID & ".HTM"
```

As noted in the variable definitions above, the input stream from the web client must be evaluated and parsed for the individual variables being transmitted. The code shown here is a typical fragment to do that.

The first action is to identify the position that the "userid=" item is located within the input stream (stored in constant sParm2) as well as the position of the second ampersand character. The first two lines of the fragment above accomplish this task. Once these locations are known, the program must extract the actual value from the stream that is associated with the element. This is done in the "sArgValue =" line. The data value is found via a complex set of calculations that define the start and end points of the value and "fence it in." Once identified, the value is stored into its specifically assigned variable (sUserID in this case) and can be used in other areas (as shown here to help create the temporary filename needed to store the HTML output).

Once all the variables are populated, the program takes that information and passes them to the R&R ActiveX control. One of these is needed for each report that can be executed from the client as defined on the web page shown previously. As described above, the sPmx variables are used to define the communication with the RIPARAM functions defined within the RRW file. In the code fragment here, the report definition has three RIPARAM functions to handle the username, beginning date and ending date for the report. The form defined in the VB program has been named generically as "Form1" although it could be anything that had more meaning.

```
If sReportToRun = "cbm01" Then
    sPm1 = "USERNAME=" & sUserID
    sPm2 = "BEGDATE=" & sBegDte
    sPm3 = "ENDDATE=" & sEndDte
    Form1.RSReport1.ReportName = sReportPath & "CBM-Test01.RSW"
    Form1.RSReport1.DataSource = "Vision"
    Form1.RSReport1.Username = "pwills"
    Form1.RSReport1.Password = "password"
    Form1.RSReport1.Parameters(0) = sPm1
    Form1.RSReport1.Parameters(1) = sPm2
    Form1.RSReport1.Parameters(2) = sPm3
End If
```

As described above, the R&R ActiveX control is called either RSReport1 or RRReport1. In the example here, the control is for the SQL version of the program. The reports used in these examples were developed against a Microsoft SQL Server(tm) database and the ODBC datasource defined as "Vision." This database also required a master userid and password, both of which were defined and passed to the report within the program.

The ReportName parameter passed to the control must exactly match one of the report files stored in the reports folder on the server. Finally, the ActiveX control can accept up to six parameters for a given report. This should be enough for most situations. In the event that more are needed, each parameter can be "overloaded," that is, created with two or more pieces of data that are parsed from within the report itself. In the fragment shown, three parameters are passed into the control. Note that parameters are numbered from 0 through 5.

```
Form1.RSReport1.CopiesToPrinter = 0
Form1.RSReport1.Destination = 13 ' HTML
Form1.RSReport1.ExportDestination = 2 ' File
Form1.RSReport1.PrintFileName = sFullOutputFileSpec
```

The remaining actions needed with the ActiveX control focus on defining the output destination and type. In the example shown here, the output will be sent to an HTML (Destination = 13) file (ExportDestination = 2) with filename defined as the temporary folder to store the output (PrintFileName = sFullOutputFileSpec). To be safe, the control sets the CopiesToPrinter to 0.

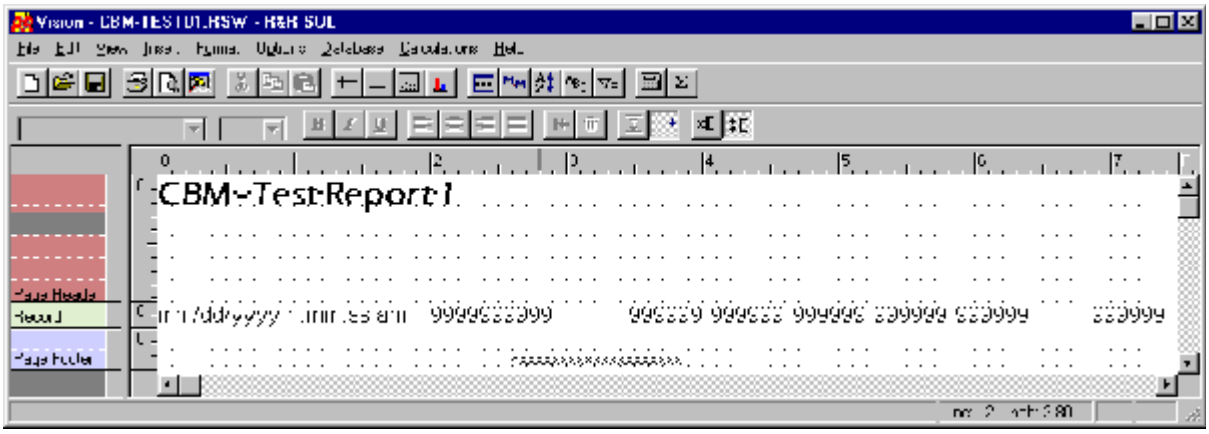
The last section of the program formats the HTTP stream for return back to the web client. The first set of text placed back into the sWriteBuffer variable ensures that the output will be interpreted as successful, valid HTML content for viewing within the browser. The most important part of the program is specified here: The call to RunReport(1) sets everything in motion. It opens the connection to the database, passes the login information & the parameters and finally exports the HTML output to the filename provided. The remaining code simply opens the channel back to the client and writes the data from the process one byte at a time.

```
    ' Construct and send response to the browser
    ' NB: We aren't doing enough error checking (we assume the
    '     report completed successfully
    sWriteBuffer = "HTTP/1.0 200 OK" & vbCrLf & "Content-Type: text/html" & _
                   vbCrLf & vbCrLf

    hStdOut = stdout()
    WriteFile hStdOut, sWriteBuffer, Len(sWriteBuffer) + 1, lBytesWritten

    Form1.RSReport1.RunReport (1) ' Run synchronously

    ' Output the report itself to the browser
    Dim c As String

    Open sFullOutputFileSpec For Input As #1
    Do While Not EOF(1)
        c = Input(1, #1)   ' Get one character.
        WriteFile hStdOut, c, 1, lBytesWritten
    Loop
    Close #1
    End
End Sub
```

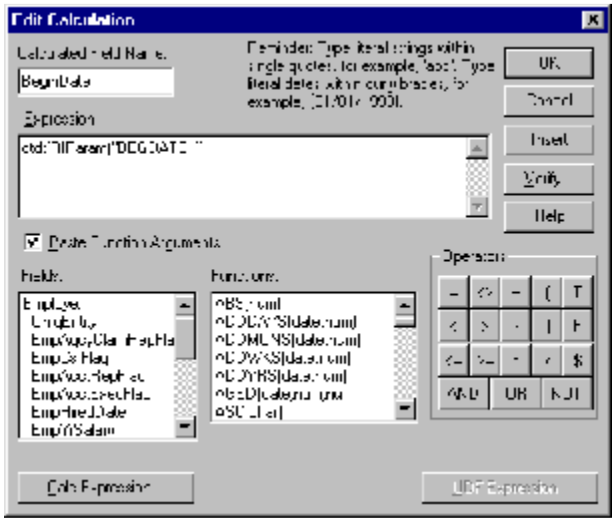
One final note regarding the choice of programming language: The sample shown here uses Visual Basic 6 to communicate with the server and IIS. Microsoft has made it relatively simple to press VB6 into this sort of role. However, other programming languages can be used to the same effect with differing degrees of difficulty. For example, by taking advantage of FoxISAPI (bundled with Microsoft's Visual FoxPro), VFP can be used in the CGI role. Additionally, C++, Delphi(tm) and any other Windows application development system able to support ActiveX controls and communicate with IIS will probably work. We leave it to individual developers to determine the most effective way to implement this capability in their most comfortable language / environment.

R&R Report

The R&R Report, "CBM-Test01.RSW," is a simple report designed specifically to highlight the interaction between it and the VB6 CGI program. Both must reside on the same network and have access to the datasource defined and passed into the control. The main designer screen is shown here.



Within the report, are three calculated fields, defining the RIPARAM elements needed to execute the report for the appropriate timeframe and for the proper user. One of these are shown here at the right. The other two define RIPARAM calculated fields for the ending date and the userid (see report specification in the appendix). Also note that each of the RIPARAM functions are natively stored and defined as text. A conversion to the appropriate data type is required before use.



In this example, the final step in the definition is the filter. Each of the calculations plays a role in controlling the output transferred back to the web client.

Also, not readily apparent in this example, but of potentially crucial importance are the formatting choices made for the report. Such considerations as lines, graphics, fonts, wrapping text, columns, etc. will make a big difference in how well laid-

out the resulting HTML output looks.

In general, pay attention to the following issues:

What we've concluded is that the complexity of the report matters a great deal. Relatively sparse reports with little in the way of extensive columns, fancy fonts, graphics and the like come out much better. When we've tried to use HTML with graphics and formatting intensive reports we thought the HTML output looked terrible.

Also, the choice of fonts seems to matter. Try to pick fonts that are fairly common and can be expected to be on most PCs (Times, Arial, Lucida, Tahoma).

Finally, don't attempt to mix word-wrapped text and single-field items on the same lines. For graphics, put the image on a single large free-form band and anchor it slightly above any text that is adjacent to it. Keep grouping bands and page resets to a minimum as well.

Conclusion

This paper has attempted to lay out the various components needed to implement an R&R based reporting solution over the web. The benefits of this include more widespread and easier access to reporting and the data stored in corporate systems. Reporting over the web opens up formerly insular data systems to customers, employees, suppliers, brokers, and more. The perception of higher value delivered to these constituents can enhance the overall relationship.

Probably the most important thing to take from this paper is the knowledge that R&R is capable of solving nearly every reporting problem you, your customers or your developers have. There are many other capabilities lurking beneath the surface waiting to be tapped. Talk to us about your most problematic reports. We're confident that R&R will deliver the results.

Liveware is available to assist in the creation and deployment of these systems where internal resources are not sufficient to implement them. As mentioned at the beginning of this paper, a web-based reporting system built on R&R can be developed and deployed in a matter of days. Visit our website at www.livewarepub.com or call us at (800) 936-6202 to learn more.

Appendix - Visual Basic Source Code

```
Attribute VB_Name = "cgiMain"
'     *****************
'     *  R&R via CGI  *
'     *****************
' Note: Set the VB project options to use Sub Main as the startup form.
'
' Author:   Colin Strasser
'           September 8, 2000
'
' Based on code from Microsoft KnowledgeBase Article Q239588.
'--------------------------------------------------------------------

Option Explicit

Public Const STD_INPUT_HANDLE = -10&
Public Const STD_OUTPUT_HANDLE = -11&

Public Const CGI_AUTH_TYPE         As String = "AUTH_TYPE"
Public Const CGI_CONTENT_LENGTH    As String = "CONTENT_LENGTH"
Public Const CGI_CONTENT_TYPE      As String = "CONTENT_TYPE"
Public Const CGI_GATEWAY_INTERFACE As String = "GATEWAY_INTERFACE"
Public Const CGI_HTTP_ACCEPT       As String = "HTTP_ACCEPT"
Public Const CGI_HTTP_REFERER      As String = "HTTP_REFERER"
Public Const CGI_HTTP_USER_AGENT   As String = "HTTP_USER_AGENT"
Public Const CGI_PATH_INFO         As String = "PATH_INFO"
Public Const CGI_PATH_TRANSLATED   As String = "PATH_TRANSLATED"
Public Const CGI_QUERY_STRING      As String = "QUERY_STRING"
Public Const CGI_REMOTE_ADDR       As String = "REMOTE_ADDR"
Public Const CGI_REMOTE_HOST       As String = "REMOTE_HOST"
Public Const CGI_REMOTE_USER       As String = "REMOTE_USER"
Public Const CGI_REQUEST_METHOD    As String = "REQUEST_METHOD"
Public Const CGI_SCRIPT_NAME       As String = "SCRIPT_NAME"
Public Const CGI_SERVER_NAME       As String = "SERVER_NAME"
Public Const CGI_SERVER_PORT       As String = "SERVER_PORT"
Public Const CGI_SERVER_PROTOCOL   As String = "SERVER_PROTOCOL"
Public Const CGI_SERVER_SOFTWARE   As String = "SERVER_SOFTWARE"

Public Declare Function Sleep Lib "kernel32" _
(ByVal dwMilliseconds As Long) As Long

Public Declare Function stdin Lib "kernel32" Alias "GetStdHandle" _
(Optional ByVal Handletype As Long = STD_INPUT_HANDLE) As Long

Public Declare Function stdout Lib "kernel32" Alias "GetStdHandle" _
(Optional ByVal Handletype As Long = STD_OUTPUT_HANDLE) As Long

Public Declare Function ReadFile Lib "kernel32" _
(ByVal hFile As Long, ByVal lpBuffer As Any, ByVal nNumberOfBytesToRead As Long, _
lpNumberOfBytesRead As Long, Optional ByVal lpOverlapped As Long = 0&) As Long

Public Declare Function WriteFile Lib "kernel32" _
(ByVal hFile As Long, ByVal lpBuffer As Any, ByVal nNumberOfBytesToWrite As Long, _
lpNumberOfBytesWritten As Long, Optional ByVal lpOverlapped As Long = 0&) As Long
```

```
Sub Main()
    Dim sReadBuffer As String
    Dim sWriteBuffer As String
    Dim lBytesRead As Long
    Dim lBytesWritten As Long
    Dim hStdIn As Long
    Dim hStdOut As Long
    Dim iPos1 As Integer
    Dim iPos2 As Integer
    Dim iPos3 As Integer
    Dim iPos4 As Integer
    Dim sArgName As String
    Dim sArgValue As String

    Dim sReportToRun As String
    Dim sUserID As String
    Dim sBegDte As String
    Dim sEndDte As String

    Dim sFullOutputFileSpec As String
    Dim sOutputFileName As String

    Dim iAmp1 As Integer
    Dim iAmp2 As Integer
    Dim iAmp3 As Integer
    Dim iAmp4 As Integer

    Dim iLP1 As Integer
    Dim iLP2 As Integer
    Dim iLP3 As Integer
    Dim iLP4 As Integer

    Dim sPm1 As String
    Dim sPm2 As String
    Dim sPm3 As String


    Const sReportPath = "C:\inetpub\wwwroot\rr-repts\"
    Const sTempPath = "c:\inetpub\wwwroot\rr-repts\TempHTMLs\"

    Const sParm1 = "report="
    Const sParm2 = "userid="
    Const sParm3 = "begdte="
    Const sParm4 = "enddte="

    iLP1 = Len(sParm1)
    iLP2 = Len(sParm2)
    iLP3 = Len(sParm3)
    iLP4 = Len(sParm4)

    sFullOutputFileSpec = sReportPath & sOutputFileName


    If Len(Environ$(CGI_CONTENT_LENGTH)) > 0 Then
        sReadBuffer = String$(CLng(Environ$(CGI_CONTENT_LENGTH)), 0)
```

```
End If

' Get STDIN handle.  The CGI protocol calls for reading GET request data
' through Stdin and writing output to Stdout.
hStdIn = stdin()
' Read what the user's browser passed as input
ReadFile hStdIn, sReadBuffer, Len(sReadBuffer), lBytesRead


' Find '=' in the name/value pair and parse the user's input into separate fields.
' In this example, there is only a single field and so we assume it's the one that
' specifies which of several possible .RSW files to use.
iPos1 = InStr(sReadBuffer, sParm1)
iAmp1 = InStr(sReadBuffer, "&")
sArgName = Left$(sReadBuffer, iLP1 - 1)
sArgValue = Mid$(sReadBuffer, iPos1 + iLP1, (iAmp1 - iLP1) - 1)
sReportToRun = Trim(sArgValue)

iPos2 = InStr(sReadBuffer, "userid=")
iAmp2 = InStr(iAmp1 + 1, sReadBuffer, "&")
sArgName = Left$(sReadBuffer, iLP2 - 1)
sArgValue = Mid$(sReadBuffer, iPos2 + iLP2, (iAmp2 - (iLP2 + iPos2)))
sUserID = Trim(sArgValue)

sFullOutputFileSpec = sTempPath & sUserID & ".HTM"

iPos3 = InStr(sReadBuffer, "begdte=")
iAmp3 = InStr(iAmp2 + 1, sReadBuffer, "&")
sArgName = Left$(sReadBuffer, iLP3 - 1)
sArgValue = Mid$(sReadBuffer, iPos3 + iLP3, (iAmp3 - (iLP3 + iPos3)))
sBegDte = Trim(Replace(sArgValue, "%2F", "/", 1, 2))

iPos4 = InStr(sReadBuffer, "enddte=")
iAmp4 = Len(sReadBuffer)
sArgName = Left$(sReadBuffer, iLP4 - 1)
sArgValue = Mid$(sReadBuffer, iPos4 + iLP4, (iAmp4 - (iLP4 + iPos4) + 1))
sEndDte = Trim(Replace(sArgValue, "%2F", "/", 1, 2))

If sUserID = "" Then
   sUserID = "YXATBS-99"
End If

If sBegDte = "" Then
   sBegDte = "01/01/" + Str(Year(Date))
End If

If sEndDte = "" Then
   sEndDte = "12/31/" + Str(Year(Date))
End If

If sReportToRun = "users" Then
    Form1.RSReport1.ReportName = sReportPath & "sample.RSW"
    Form1.RSReport1.DataSource = "Sample Access DB"
End If

If sReportToRun = "lag" Then
```

```
    sPm1 = "USERNAME=" & sUserID
    sPm2 = "BEGDATE=" & sBegDte
    sPm3 = "ENDDATE=" & sEndDte
    Form1.RSReport1.ReportName = sReportPath & "TimeLag.RSW"
    Form1.RSReport1.DataSource = "Sample Access DB"
    Form1.RSReport1.Parameters(0) = sPm1
    Form1.RSReport1.Parameters(1) = sPm2
    Form1.RSReport1.Parameters(2) = sPm3
End If

If sReportToRun = "cbm01" Then
    sPm1 = "USERNAME=" & sUserID
    sPm2 = "BEGDATE=" & sBegDte
    sPm3 = "ENDDATE=" & sEndDte
    Form1.RSReport1.ReportName = sReportPath & "CBM-Test01.RSW"
    Form1.RSReport1.DataSource = "Vision"
    Form1.RSReport1.Username = "pwills"
    Form1.RSReport1.Password = "password"
    Form1.RSReport1.Parameters(0) = sPm1
    Form1.RSReport1.Parameters(1) = sPm2
    Form1.RSReport1.Parameters(2) = sPm3
End If

If sReportToRun = "cbm02" Then
    sPm1 = "USERNAME=" & sUserID
    sPm2 = "BEGDATE=" & sBegDte
    sPm3 = "ENDDATE=" & sEndDte
    Form1.RSReport1.ReportName = sReportPath & "SalesReport.RSW"
    Form1.RSReport1.DataSource = "Vision"
    Form1.RSReport1.Username = "pwills"
    Form1.RSReport1.Password = "password"
    Form1.RSReport1.Parameters(0) = sPm1
    Form1.RSReport1.Parameters(1) = sPm2
    Form1.RSReport1.Parameters(2) = sPm3
End If


Form1.RSReport1.CopiesToPrinter = 0
Form1.RSReport1.Destination = 13 ' HTML
Form1.RSReport1.ExportDestination = 2 ' File
Form1.RSReport1.PrintFileName = sFullOutputFileSpec

' Construct and send response to the browser
' NB: We aren't doing enough error checking (we assume the
'     report completed successfully
sWriteBuffer = "HTTP/1.0 200 OK" & vbCrLf & "Content-Type: text/html" & _
               vbCrLf & vbCrLf

hStdOut = stdout()
WriteFile hStdOut, sWriteBuffer, Len(sWriteBuffer) + 1, lBytesWritten

Form1.RSReport1.RunReport (1) ' Run synchronously

' Output the report itself to the browser
Dim c As String
```

```
    Open sFullOutputFileSpec For Input As #1
    Do While Not EOF(1)
        c = Input(1, #1)   ' Get one character.
        WriteFile hStdOut, c, 1, lBytesWritten
    Loop
    Close #1
    End
End Sub
```

Appendix - Sample Report Specification

```
  R&R SQL Report Designer
  Version: 2.0 U00  Build: 8.1.25
  Win32 on Windows 95 (4.10)
   A
  Date: 04/25/2001     Time: 21:07

     Report Name: C:\Inetpub\wwwroot\rr-repts\CBM-Test01.RSW
         Date: 04/25/2001      Time: 20:58:38
     Report Version: 8.01


SELECT STATEMENT


     select
      `UniqEntity`, `EmpAgcyClaimRepFlag`, `EmpCsrFlag`,
       `EmpAcctRepFlag`, `EmpAcctExecFlag`, `EmpHiredDate`,
       `EmpYrSalary`, `UniqCdEmpInOutReason`
     from
     Employee



FORMAT INFORMATION

     Print Options
       Current printer: Generic / Text Only on FILE: (WINSPOOL.DRV)
       Number of report copies: 1
       Starting page number: 1
       Ending page number: 65535

       Print to File: c:\inetpub\wwwroot\spec.txt
       Collate copies: Yes
     Page Setup
       Page length: Letter 8 1/2 x 11 in
       Top margin: 0.50 inches
       Bottom margin: 0.50 inches
       Left margin: 0.50 inches
       Right margin: 0.50 inches
       Page orientation: Portrait
       Interline spacing: Yes
       Horizontal ruler spacing: 10
       Vertical ruler spacing: 10
     Record Layout
       Compress record/group lines? Yes
       Suppress record lines? No
       Headers/footers in summary? Yes
       Break record area? No
       Label type:
       Record order: Across
       Records across: 1
       Columns across: 1
```

Record width: 3.00 inches
Record height: 0.00 inches
Record copies: 1

DATABASE INFORMATION

Data Source: Vision
DS Driver: ODBCJT32.DLL

DS Driver DBMS: ACCESS Version: 04.00.0000
Master Table: Employee
(Alias: Employe)

FILTER INFORMATION

Include all records where EmpHiredDate is greater than or equal to
BeginDate and EmpHiredDate is less than or equal to EndDate and
UseIDChk is equal to 'bearman'

FIELD INFORMATION

Band	Line	Pos:In	Width	Field	Type	Format	Font	Att	Trm	Clr
Page H	1	0.00	19 Chr	CBM - Test Report 1	T	Left	1	B	N	Black
Page H	2	---------- Blank Line ----------								
Page H	3	---------- Blank Line ----------								
Page H	4	---------- Blank Line ----------								
Page H	5	---------- Blank Line ----------								
Record	6	0.00	21 Chr	Employe.EmpHiredDate	D	mm/dd/yyyy h:mm:ss am	1		N	
Record	6	1.90	10 Chr	Employe.UniqEntity	N	Fixed	1		N	Black
Record	6	3.40	6 Chr	Employe.EmpCsrFlag	N	Fixed	1		N	Black
Record	6	4.00	6 Chr	Employe.EmpAcctRepFl	N	Fixed	1		N	Black
Record	6	4.60	6 Chr	Employe.EmpAcctExecF	N	Fixed	1		N	Black
Record	6	5.20	6 Chr	Employe.EmpAgcyClaim	N	Fixed	1		N	Black
Record	6	5.80	6 Chr	Employe.EmpYrSalary	N	Fixed	1		N	Black
Record	6	6.80	6 Chr	Employe.UniqCdEmpInO	N	Fixed	1		N	Black
Page F	7	---------- Blank Line ----------								
Page F	8	2.60	21 Chr	Repname_rr	C	Left	1		N	Black

Font Names

1 Roman 10cpi 12.0

9 Calculated and 0 Total Fields

BeginDate (BeginDate) DateTime Calculated
ctdt(RIParam("BEGDATE"))
Date_rr (Date_rr) Date Calculated
Today's date
DATE()
EndDate (EndDate) DateTime Calculated
ctdt(RIParam("ENDDATE"))
Page_rr (Page_rr) Numeric Calculated

```
      Page number
      PAGENO()
   Query_rr  (Query_rr            )  Character      Calculated
      Current query expression
      QUERY()
   Recno_rr  (Recno_rr            )  Numeric        Calculated
      Report record number
      RECNO()
   Repname_rr(Repname_rr          )  Character      Calculated
      Current report name
      REPNAME()
   Time_rr   (Time_rr             )  Character      Calculated
      Current time
      TIME()
      Translation: {fn CURTIME()}
   UseIDChk  (UseIDChk            )  Character      Calculated
      RIParam("USERNAME")

      FIELD DATA TYPES
             Field         Rep type Database type
      -------------------- -------- -------------
      Employe.UniqEntity   Num      code 4
      Employe.EmpAgcyClaim Num      code 4
      Employe.EmpCsrFlag   Num      code 4
      Employe.EmpAcctRepFl Num      code 4
      Employe.EmpAcctExecF Num      code 4
      Employe.EmpHiredDate DateTime code 8
      Employe.EmpYrSalary  Num      code 4
      Employe.UniqCdEmpInO Num      code 4

LINE INFORMATION

   Conditional Lines:

      None

   Line Height Overrides:

      None

SORT/GROUP INFORMATION

      None

REGIONAL SETTINGS

      Decimal point: DOT
      Thousands: COMMA
      Currency: _ (Prefix)
      Short Date: mm/dd/yyyy
      True: T
      False: F
```