RPG Web Development — Getting Started

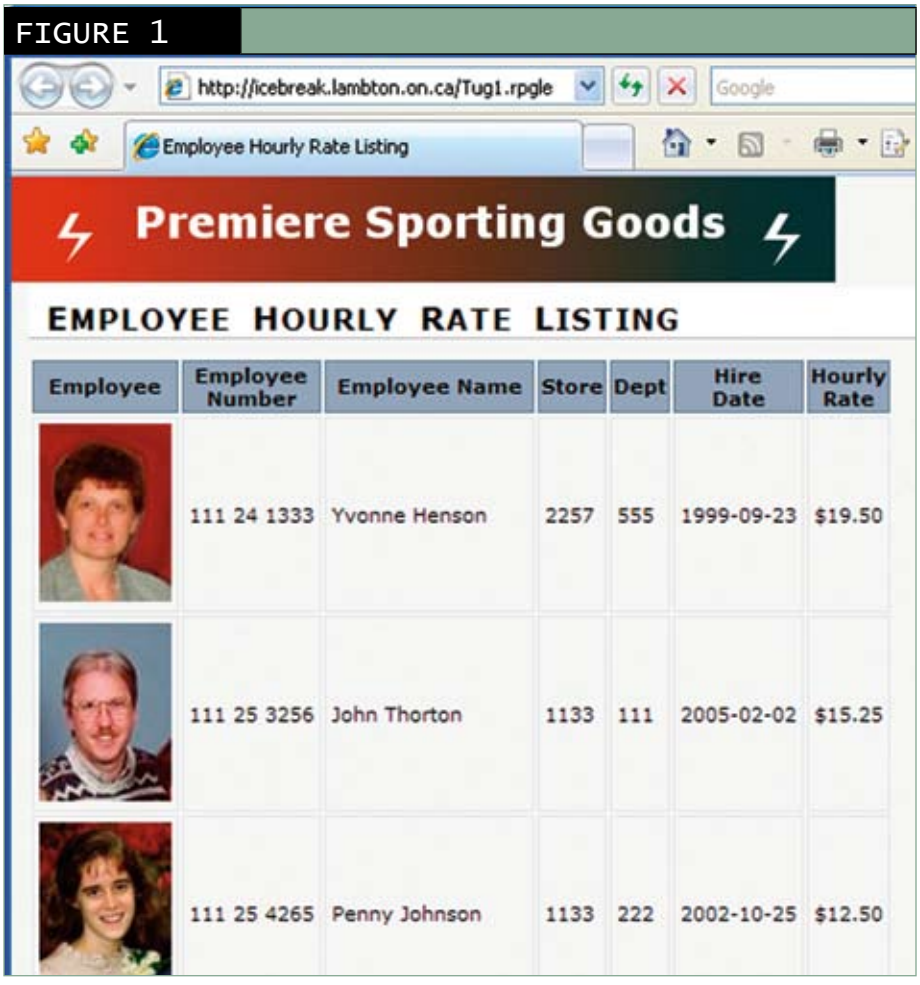
By Jim Cooper

Welcome to the wonderful and exciting world of RPG Web development. This article is focused on the basics of using RPG as a server-side scripting language to develop Web applications. Using RPG is just as easy and powerful as developing Web applications with any other language. More importantly, developing RPG Web applications does not require CGI, Java, JSPs, PHP, WebSphere, PASE, and complicated development tools.

In this article, an HTTP/application server technology called IceBreak is used that provides an environment for RPG server side programming. The basics of developing a simple RPG Web application are discussed that would be the equivalent of a green-screen load-all subfile. RPG can also be easy intergraded with technologies such as SQL, AJAX and Web services, which will be presented in future articles.

RPG is a business language that adapts well to the 3-tier model as it applies to Web applications. The data or first tier contains the database source; the middle tier or application server contains the business logic to process the data, and finally the third or presentation tier is the user interface where all the data is presented. RPG fits well into this model. In RPG Web applications, the data tier is the DB2 database stored on the System i. The middle tier contains the business logic developed in RPG that runs on the application server and the third tier is the user interface developed in HTML where the information is rendered to the browser. Why is it important to adhere to the 3-tier model? An application developed with this model becomes very flexible, and changes can be made at one tier without affecting the other two tiers.

FIGURE 1



This article was condensed from a larger lab tutorial. The full tutorial can be downloaded from the TUG Web site. In addition, tutorials and other RPG Web applications can be viewed at http://www.IceBreak4RPG.com.

You work for a small to medium size System i company and want to continue developing applications with RPG but want to migrate to Web applications in an easy to manage environment. The key words here are "easy to manage environment" because it has not been easy to develop RPG Web applications. In the past, RPG applications had to be combined with other technologies to run on the Web. Worst yet, RPG did not have a native HTTP/application server. Therefore, to develop new Web applications, it was necessary to reach outside the ILE environment and combine RPG with other technologies. Well, this is no longer required. Let us have a look at how easy it is to develop a native RPG Web application.

The Application

The human resource manager of Premiere Sporting Goods wants a Web application that displays a list of employees. In this application, all of the employee records are retrieved from the employee file and displayed in the browser as shown in **Figure 1**.

Developing an RPG Web Application

There are several steps in developing an RPG Web application:

1. Create an optional cascading style sheet (CSS).
2. Create the HTML user interface. Web pages should use the rules of XHTML, which is replacing HTML. For this article, HTML will imply XHTML. In green-screen applications, the user interface is a display file. In a browser application, the user interface is HTML.

FIGURE 2

```
<!--#tag="page_header"-->
<html>
<head>
  <title> Employee Hourly Rate Listing </title>
  <link rel="stylesheet" type="text/css"
  href="../Theme/Master.css" />
</head>
<body>
  <img src="./images/premierebanner.gif" />
  <h1> Employee Hourly Rate Listing </h1>

  <table>
    <tr>
    <th> Employee                          </th>
    <th> Employee <br /> Number            </th>
    <th> Employee Name                     </th>
    <th> Store                             </th>
    <th> Dept                              </th>
    <th> Hire     <br /> Date              </th>
    <th> Hourly   <br /> Rate              </th>
    </tr>

<!--#tag="table_row"-->
   <tr>
   <td> <img src="./images/<% = %char(employeeNo)
                              %>.jpg"> </img> </td>
   <td> <% = %editW(employeeNo  : '0   &  &    ')
                                          %> </td>
   <td> <% = %trim(firstName)
                + ' ' + %trim(lastName) %> </td>
   <td> <% = %char(storeNo) %>              </td>
   <td> <% = %char(deptNo) %>               </td>
   <td> <% = %char(hireDate) %>             </td>
   <td> <% = %editC(hourlyRate  : '3' : '$')
                                          %></td>
   </tr>

<!--#tag="finish_page"-->
  </table>
</body>
</html>
```

3. Create an RPG program that contains the business logic.
4. Compile the HTML file and RPG program into a program object (*PGM) that is stored in a library. This is similar to compiling a green-screen application. The only difference is that the user interface is an HTML file instead of a display file.
5. Debug the application.
6. Request the RPG application from a browser.

The RPG Web application in this example consists of three source files that are created using WDSc and saved in an IFS folder:

- The **CSS** (cascading style sheet) provides the rules for how different elements of the Web page are rendered to the browser.
- The **HTML file** contains the user interface or presentation layer. In this example, the HTML file is called TUG1.htm with the extension HTM, which identifies this file as an HTML file.
- The **RPG file** contains the business logic. In this example, the RPG program is called TUG1.rpgle with the extension RPGLE, which identifies it as an RPG Web application.

Cascading Style Sheet (CSS)

Cascading Style Sheets (CSS) is a simple mechanism for adding style (e.g. fonts, font sizes, colors, margins, and spacing) to Web files. Styles define how to display HTML elements. Normally, standard CSS files are created for an organization and all applications and Web pages share the same CSS. Usually, CSS files are created and used for several applications as a means of standardizing browser-based applications. Once the CSS becomes a company standard, there is very little need to modify it. A CSS is a very important component of a Web application, but unfortunately there is not enough space to discuss CSS in this article.

The HTML User Interface

The first step in developing an RPG Web application is to design and create the user interface using HTML. As mentioned, Web pages should use the rules of XHTML, which is replacing HTML. However, this article, does not discuss the rules of HTML. Instead, I will leave that for you to explore. The HTML file defines what is rendered to the browser and is similar to creating the display file for a green-screen application. The HTML user interface for this application is defined in the `TUG1.htm` file shown in **Figure 2**. Notice the link on line 5 to a CSS file called `Master.css`. The `Master.css` file is where the different styles are defined.

The code in **Figure 2** is a basic HTML example and is not discussed in detail here. There are many books that can be used to learn the basics

of HTML. The important thing to understand is that once a user interface template is created in HTML, it can be easily adopted to other applications. Once a company decides on a "user interface look," the same HTML files can be used repeatedly.

HTML tables are used to list items in a Web application in a similar manner as a subfile is used in a display file. In this application, a HTML table is used to list the employee records from the employee file. Each record is read from the employee file and a table row is inserted into the HTML table. Therefore, a table row is defined to describe how the data is to appear in the browser. Although one or more rows will appear in the table when it is rendered to the browser, only *one* table row is defined in the HTML file. This single table row defines the first line of one or more identical lines to be displayed in the table. As each record is read from the employee file, the RPG program uses the same table row definition in the HTML file to insert a new row into the HTML table.

FIGURE 3

```
<%
     FempPayTBL IF   E           K Disk    Rename(empPayTBL : empPayR)
     D  tag            S             50A   varying
      /free
       tag = 'page_header';
       exSR writeHTML;
       read empPayTBL;

       dow Not %EOF (empPayTBL);
         tag = 'table_row';
         exSR writeHTML;
         read empPayTBL;
       endDo;

       tag = 'finish_page';
       exSR writeHTML;
       *inLR = *ON;

       begSR writeHTML;
         select;
           when tag = 'page_header';
             %> <!--#include file="TUG1.htm" tag="page_header"--> <%

           when tag = 'table_row';
             %> <!--#include file="TUG1.htm" tag="table_row"-->   <%

           when tag = 'finish_page';
             %> <!--#include file="TUG1.htm" tag="finish_Page"--> <%
         endSL;
       endSR;
      /end-free
%>
```

The RPG Program

The RPG program for this application is shown in **Figure 3**. This program is the equivalent of a basic green-screen load-add subfile. Normally I would use a procedure in place of the subroutine but used a subroutine for those that have not embraced procedures. This program performs three steps to build the Web page. First, it includes the page header called page_header from the TUG1.htm file. Examine **Figure 2** and you will see that the code identified by `#tag="page_header"` includes the HTML code for the page header and the table header. Next, a `doW/endDo` loop is used to loop through the employee file and include a table row in the Web page for each employee. Thus, the program is building a table dynamically from the database file. Once the table is created, the program includes the HTML code to finish the Web page and render it to the browser.

RPG Extensions

When an RPG application contains a display file, there are special RPG operations, such as EXFMT, that are used to work with the display file. Likewise, in an RPG Web application, there are extensions built into the HTTP/application server that allows RPG to work with the HTML file. The basic extensions used in this application are discussed next.

(1) The #tag Anchor and #include Directive

The `#TAG` anchor in the HTML file in **Figure 2** is used in conjunction with the `#INCLUDE` directive in the RPG program in **Figure 3**. It is important to understand that the RPG program is

dynamically building a Web page. When processing the HTML code, the RPG program includes specific segments from the HTML file. The #TAG anchors in the HTML file separate the HTML code into the segments needed by the RPG program at the time of processing. In **Figure 2**, the HTML code contains three segments identified as "page_header", "table_row", and "finish_page". When the RPG program includes the HTML code for a specific tag, that HTML code is inserted into the Web page.

(2) Delimiters <% and %>

RPG programs contain code that is surrounded by the delimiters <% and %>. The code is executed on the server and can contain any expressions, statements, procedures, or operators valid for the RPG programming language. When an application is requested, the HTTP/application server needs to know what code is to be executed and what code is content that is to be rendered to the browser. RPG code enclosed by <%...%> is just executed, while expressions that include an equal sign, <% = ... %>, are evaluated and the result is emitted as content. Therefore, the statement <% = %trim(firstName) + ' ' + %trim(lastName) %> in **Figure 2** renders a concatenated employee name. The dynamic values are determined at runtime and are substituted at the evaluated (=) symbol. As a result, when the page is rendered to the browser, the dynamic values appear on the Web page.

In the RPG program in **Figure 3**, the delimiter <% is specified at the beginning of the RPG program and the delimiter %> is specified at the end of the program. The RPG executable code must be between the opening and closing delimiters. There are times in the RPG program where content needs to be included and not executed. For example, consider the include statement %> <!--#include file="TUG1.htm" tag="page_header"--> <% in the subroutine in **Figure 3**. This include directive is not an executable statement, but a directive that inserts content into the Web page. Therefore, the first delimiter %> is specified to close the RPG executable code. At the end of the include directive, the delimiter <% continues with RPG executable code. When a user requests an application, the HTTP/application server locates the program object in the library and executes all the RPG code between <%...%>. The RPG code between <%...%> is never rendered to the browser.

Compile the RPG Program

Once the RPG and HTML source files are created and saved in the IFS folder, the application can be compiled into a native program object (*PGM). To compile the application, it is requested in a browser by entering http://server_name/TUG1.rpgle in the Address box, where server_name is the name of the server and TUG1.rpgle is the name of the RPG application. If the program is new or has been changed, the HTTP/application server will compile the application into a new object. If the program compiles without errors, a *PGM object is created and stored in the library that is assigned to the HTTP/application server. In addition, if the program compiles, the HTTP/application server executes the program from the library and renders the results to the browser as shown in **Figure 1**.

FIGURE 4

```
<html>
<head>
  <title> Employee Hourly Rate Listing </title>
  <link rel="stylesheet" type="text/css"
   href="../Theme/Master.css" />
</head>
<body>
  <img src="./images/premierebanner.gif" />
  <h1> Employee Hourly Rate Listing </h1>

  <table>
    <tr>
      <th> Employee                 </th>
      <th> Employee <br /> Number   </th>
      <th> Employee Name            </th>
      <th> Store                    </th>
      <th> Dept                     </th>
      <th> Hire      <br /> Date    </th>
      <th> Hourly    <br /> Rate    </th>
    </tr>

    <tr>
      <td> <img src="./images/111241333.jpg">
           </img> </td>
      <td>  111 24 1333                  </td>
      <td> Yvonne Henson                 </td>
      <td> 2257                          </td>
      <td> 555                           </td>
      <td> 1999-09-23                    </td>
      <td>  $19.50                       </td>
    </tr>

    <tr>
      <td> <img src="./images/111253256.jpg">
           </img> </td>
      <td>  111 25 3256                  </td>
      <td> John Thorton                 </td>
      <td> 1133                          </td>
      <td> 111                           </td>
      <td> 2005-02-02                    </td>
      <td>  $15.25                       </td>
    </tr>
     .
     .
  </table>
</body>
</html>
```

Run the RPG Application in a Browser

The HTTP/application server does not execute the source files in the IFS folder. Only the program object (`*PGM`) stored in the library is executed. Therefore, once an application is completed, only the program object needs to be available to run the application. To run the application, enter `http://server_name/TUG1.htm`, where server_name is the name of the server and TUG.rpgle is the name of the RPG application. The browser makes the request to the application server which retrieves and executes the program object. The result is a Web page displayed in the browser as shown in **Figure 1**.

View Source

As mentioned, a Web page rendered to the browser by an RPG application only contains HTML. **Figure 4** illustrates the result of clicking Page/View Source in the browser. There is no RPG code or business logic sent to the browser. The user can see only the HTML generated from the RPG application. Notice in **Figure 4** that the RPG application generated several table rows which are a result of looping (`doW/endDo`) through the employee file.

Compile Errors

If errors are encountered during the compile, a compile listing with errors is returned to the browser instead of the Web page. The errors in the compile listing are bookmarked so the errors can be located by clicking on the syntax error in the source. If there are syntax errors, correct the errors in the source code, save the program, and refresh the browser. This process continues until all compile errors are corrected which results in a program object being created and a Web page being displayed. **Figure 5** illustrates a program compilation with errors.

Runtime Errors

If a runtime error occurs, the HTTP/application server provides the current job log as an HTML page in the browser. The runtime error shown in **Figure 6** is returned to the browser. In the example in **Figure 6**, an attempt to divide by zero run-time error occurred.

IceBreak Application Server

Modern Web languages have a scripting language, an easy to use user interface, and an HTTP/application server that runs the applications. The application in this article demonstrates how easy it is to deploy RPG as a scripting language with an easy to use HTML user interface. So what makes this magic happen? The answer is IceBreak Application Server, one of the most exciting Web development technologies to be developed in recent years.

IceBreak is a powerful HTTP/application server that runs natively on System i5, iSeries, and AS/400. IceBreak installs in minutes with little configuration, and has an integrated development feature that allows developers to build and deploy Web applications using the

FIGURE 5

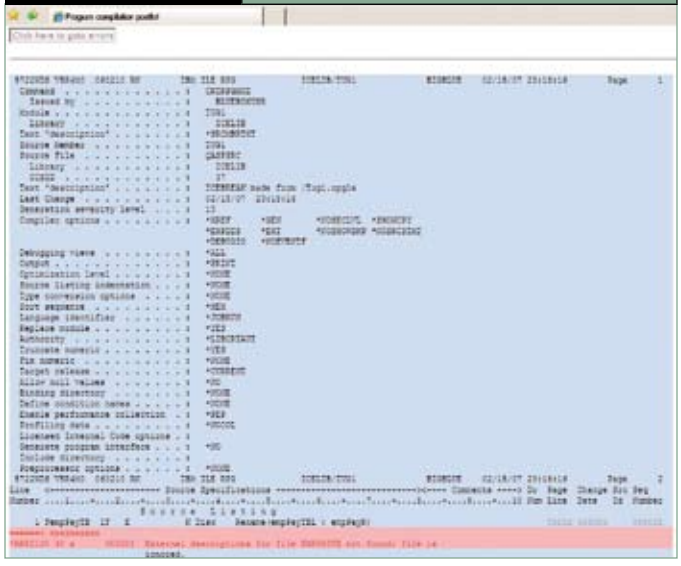


FIGURE 6

http://icebreak.lambton.on.ca/Tug1.rpgle

Message ID	Description	Message type	Severity	From Program	To Program
CPF1124	Job 055544/BLUEBOXUSR/SVC019 started on 02/15/07 at 23:23:57 in subsystem ICEBREAK in ICEBREAK. Job entered system on 02/15/07 at 23:23:57.	*INFO	0	QWTPIIPP	*EXT
CPF9811	Program TUG1 in library ICELIB not found.	*ESCAPE	40	QLIRONDL	SVC200
CPF1018	Data area $SERVERID in QTEMP not found.	*ESCAPE	40	QWCCCHVC	SVC018
CPC2206	Ownership of object SERVERID in QTEMP type *DTAARA changed.	*COMP	0	QSYCHONR	QLIINSRT
CPC0904	Data area SERVERID created in library QTEMP.	*COMP	0	QWCCCRVC	SVC018
CPC7301	File POSTLIST created in library QTEMP.	*COMP	0	QDDCPF	SVC200
CPC7305	Member POSTLIST added to file POSTLIST in QTEMP.	*COMP	0	QDDCPFM	SVC200
CPF2105	Object TUG1 in ICELIB type *PGM not found.	*ESCAPE	40	QLIDLOBJ	SVC200
CPC3301	Member TUG1 file QASPSRC in ICELIB changed.	*COMP	0	QDBCHGME	SVC200
CPD4090	Printer device PRT01 not found. Output queue changed to QPRINT in library QGPL.	*DIAG	10	QDHCOPEN	QC2IO
CPC2206	Ownership of object QRNFER in QTEMP type *USRSPC changed.	*COMP	0	QSYCHONR	QLIINSRT
CPC2206	Ownership of object QRNFEF in QTEMP type *USRSPC changed.	*COMP	0	QSYCHONR	QLIINSRT
CPC2206	Ownership of object TUG1 in ICELIB type MODULE changed.	*COMP	0	QLIINSRT	QLIINSRT
CPI2126	AUT parameter ignored.	*INFO	0	QLIINSRT	QBNINSCB
CPI2121	Replaced object TUG1 type *MODULE was moved to QRPLOBJ.	*INFO	0	QLIREINS	QLIINSRT
RNS9305	TUG1 ICELIB 0002/15/0723:23:58	*COMP	0	QRN4CPP	SVC200
CPC2206	Ownership of object RETURNCODE in QTEMP type *DTAARA changed.	*COMP	0	QSYCHONR	QLIINSRT
CPC0904	Data area RETURNCODE created in library QTEMP.	*COMP	0	QWCCCRVC	QRN4CPP
CPD4090	Printer device PRT01 not found. Output queue changed to QPRINT in library QGPL.	*DIAG	10	QDHCOPEN	QUIOPEN
CPC2206	Ownership of object TUG1 in ICELIB type *PGM changed.	*COMP	0	QSYCHONR	QLIINSRT
CPC5D07	Program TUG1 created in library ICELIB.	*COMP	0	QBNBIND	SVC200
MCH1211	Attempt made to divide by zero for fixed point operation.	*ESCAPE	40	TUG1	TUG1
CPF9999	Function check. MCH1211 unmonitored by TUG1 at statement 0000000261, instruction X'0000'.	*ESCAPE	40	QMHUNMSG	TUG1
RNQ0102	Attempt to divide by zero (C G D F).	*COPY	99	QRNXIE	QRNXIE

native ILE languages RPG and COBOL and other technologies such as SQL, XHTML, XML, Web Services, and AJAX. IceBreak is not a tool but a powerful advanced HTTP/application server that *does not* require CGI, Java, Apache, WebSphere (WAS), PASE, WebFacing, HATS, or third-party generator tools. With IceBreak, developers benefit from a single, integrated application-hosting environment. The ILE environment is the native environment of the System i and IceBreak provides the best Web infrastructure to take advantage of ILE.

Jim Cooper *is Coordinator of the Internet Application Developer program at Lambton College in Sarnia, ON, where he has taught System i (AS/400) technology since 1991. The discovery of IceBreak Application Server technology was so revitalizing that it saved his RPG courses from elimination, and as a result, he has joined System & Method, the distributors of IceBreak in North America. He can be reached at jim.cooper@LambtonCollege.ca, jac@system-method.com or (519) 542-1268 ext. 3219.*