Is CGI an Obsolete Technique for RPG Programmers?

Scott Klement

As an alternative to green screens, I use the CGIDEV2 toolkit from IBM to write my RPG code to have a web interface. Lately I've received a lot of comments like "CGI is old school", "kids coming out of school today won't touch CGI", "CGI is passé", and so forth.

The big problem with all of these claims is that they lump CGIDEV2 together with plain CGI, and they lump the way CGI programs were written on other platforms years ago with the way they're written today on the System i. They're not the same thing. In this article, I try to explain what the problems with CGI are, but I also explain how CGIDEV2 and similar toolkits overcome these problems. I also give a bit of a comparison between CGIDEV2, Java, and PHP.

Before I can really address why I think CGIDEV2 is still a viable alternative, I need to explain what CGI is, and what the the problems are with it.

What's CGI?

CGI is an *interface* between a web server and a program. It is not a programming language, nor the name of a toolkit or API. It's a set of guidelines that describes how a web server can interface with a program. You see, the original implementation of the HTTP protocol was for downloading HTML documents. People running these early web sites quickly discovered that it would be useful if a web site could have programming logic behind it to generate web pages on-the-fly. Without this, we wouldn't be able to have real-time information, shopping carts, or any other dynamic data on the web.

When a program is configured as a CGI script, it means that instead of downloading the target of a URL, as HTTP was originally designed to do, it actually runs a program at that location, then whatever that program writes out is send back and displayed in the browser.

For example, consider the following two URLs:

```
http://www.example.com/books/mockingbird.html
http://www.example.com/scripts/getbook?book=mockingbird
```

The first URL is a standard HTTP URL, as described in the original standard. It connects to a server named "www.example.com" and it requests a document named /books/mockingbird.html. This HTML file is then downloaded into the browser, and the browser will read the code inside the HTML to display a screen.

The second URL is a CGI URL. The browser still connects to www.example.com, and it requests a document named /scripts/getbook?book=mockingbird. But, this time, the server has "scripts" configured as a CGI script directory. It goes to that directory, and instead of downloading the object named "getbook", it actually *runs it as a program*. The getbook program gets a parameter of book=mockingbird that it can use as input. Whatever data that program writes out gets sent to the browser. That's how CGI works.

What's Wrong With CGI?

Here are the primary complaints with CGI:

- **Performance:** A new process has to be created for each invocation, and that process requires a program to be loaded from disk, which in turn will open and read files from disk.

- **Complex:** Data is passed to a CGI script in an encoded format, and CGI programs have to contain code that decodes it.

- **Hard-Coded HTML:** CGI programs require HTML to be coded inside the programming language statements, which leads to awkward code that's difficult to maintain.

I will address each of those complaints below.

Performance

On this score, it's important to understand that the complaints aren't really coming from the System i world. They're coming from the larger IT world, primarily Windows and Unix.

On these platforms, there's really no way to invoke a program that's already in memory. When you call a program, it goes out and finds it on disk, loads it into memory, and runs it. Since that program has to run concurrently with the HTTP server, a new process must always be created.

That's not the case on i5/OS, however! The HTTP server included with i5/OS (and it doesn't matter whether you're running Original or Apache) creates a pool of CGI server jobs. It does not have to start a new process, it just uses existing ones.

RPG programs have always had the ability to end with *INLR=*OFF by using the RETRN op-code in RPG/400, or the RETURN op-code in ILE RPG/400. When this happens, the program remains loaded into memory. The variables aren't re-initialized. The database files don't have to be reopened. Subsequent calls call the same copy of the program that's already in memory, and don't reload it from disk. All of this saves CPU time. *There's absolutely nothing like this on other platforms!*

With ILE, we also have activation groups, which let us load additional things such as service programs and programs written in other languages like C, Cobol, and CL into memory. We can keep all of those things loaded into memory so that the system doesn't have to go back and reload it from disk. Activation groups give all of the ILE programs the same capabilities that RPG has.

So, again, no new process is created, instead the work is passed off to a pool of CGI worker jobs, and the program only needs to be loaded once -- the first time it's called in a given worker job.

Starting in V5R3, there's a new Apache directive called CgiInitialUrl that lets you specify an HTTP URL that's run automatically when each CGI worker job begins, thus letting you write code that pre-loads programs, pre-opens files, etc, as needed. If you use that directive, you don't even have to wait for the CGI programs to load on the first call!

Since other platforms don't have this ability to load programs into memory and keep calling them from memory, they have to use another technique. They'll start up never-ending programs (background jobs, or "daemons") to serve out requests. These programs remain running all the time, and therefore don't have to be loaded. However, they can't be called via CGI, because CGI specifies loading a program object (by name) from disk. Therefore, CGI performs poorly by comparison.

But not on i5/OS! In fact, in many cases, a well-written CGI program will outperform a well-written JSP or PHP tool, especially on a smaller server.

Complex

The typical data transmitted from a form on a web page would look something like this:

```
language=en&order=54321&shipto=Scott+%26+Tracy+Klement
```

Of course, in a real application, there'd be far more fields than the three that I included in this sample, but hopefully you get the idea. Each parameter name is followed by an equal sign, followed by its value. An ampersand (the & character) is used to separate each parameter from the next. Any special characters (such as equal sign, ampersands, spaces, quotes, etc.) have to be encoded in %xx format, where xx is the hex ASCII code for the character. This can get extremely complicated, especially if you have to handle more than one character set for more than one language.

These strings aren't any different in CGI than they are in non-CGI environments. Tools like PHP and Java still receive the strings this way, but, those tools decode this information for you automatically so you don't have to worry about it.

On i5/OS, IBM includes APIs that parse this information for you. Other platforms don't have these! On other platforms you'd have to write your own, or download someone's "free code" to do it, so people coming from different platforms may not realize that you don't have to do this manually on i5/OS.

But, even with the APIs, it's not that easy. That's one reason that people use tools like CGIDEV2 (or similar alternatives such as eRPG SDK, CGILIB, etc) provide routines you can call to decode these things. They make it extremely easy, you just call a procedure asking for the parameter name, and the value is returned to you.

```
   shipto = zhbGetVar('shipto');
```

It's every bit as easy as it would be in languages like Java or PHP.

Hard-Coded HTML

The traditional CGI software that people complain about typically has the HTML hard-coded into the source. For example, in a CGI program written in C, you might have this:

```
    printf("<p>Click here for <a href=\"http://www.example.com\">Dudley's
Page</a>");
```

The entire page would be made up of these print statements, each one adding a little more HTML code. Every time there's a special character (such as the quotes in the above example) it'd have to be escaped with a backslash, or by inserting a hex value, etc.

I don't mean to pick on C. The same is true for other languages. In RPG you'd have lots of calls to the QtmhWrStout() API, each one writing one little chunk of data, the same way. In fact, at one point in time, the Perl programming language was synonymous with CGI programming because it's syntax made it easier to insert HTML code into the middle of a document without spending as much time with escaping quotes and other special characters.

In addition to the awkwardness of inserting the HTML into your code, it made for a maintenance nightmare. You see, the artistic people who are good at designing pretty pages and so forth are rarely the same people who are good at writing programs. Similarly, programmers (like me!) aren't always very creative or good artists.

Having the HTML code separated better from the program code makes things much easier to maintain, because the right people can work on the right part of the problem. In tools like PHP or JSP, they solve this problem by making everything in the source code be HTML by default. Then there are special "tags" that you insert to get program code. For example, here's a simple PHP document:

```
<html><head>
   <title>Scott's Cool Page</title>
</head><body>
<?php
     echo 'Hello World';
?>
</body>
</html>
```

As you can see, for the most part it's a normal HTML document. Anywhere I want PHP code, I just insert a <?php?> tag with the PHP code in it. That way, the HTML has a large degree of separation from the code. You don't have to worry about escaping any special characters in the HTML, because everything is HTML by default! Java's JSP stuff is very similar to the PHP example.

However, if you use CGIDEV2 (or a similar tool) you can actually take it one step further, and remove all of the program code from the HTML! You just insert special placeholders in the code where you want to divide it up, and where you want variables to be placed. In CGIDEV2, you'd have this, instead:

```
<html><head>
     <title>Scott's Cool Page</title>
  </head>
  <body>
     /%Message%/
  </body>
  </html>
```

Then you'd have an RPG source member that's completely separate that tells it to replace the /%Message%/ string with data from the RPG program, and write out the HtmlPage section of code.

There's no problem with escaping, and it's actually better separated (and easier to maintain) than the PHP or JSP examples would be.

Again, most of the world doesn't have CGIDEV2. I've spent a long time searching for a similar toolkit for other platforms, and there are some partially implemented ones, but nothing nearly as nice as CGIDEV2. When people complain about CGI programs, they're not expecting this much of the work to be done for you!

Other Feature Comparisons

Don't get me wrong, PHP and Java are both very good choices for web development. But if you work in a shop where there's a lot of RPG talent, it just makes sense to get started by doing your web programming in RPG. The learning curve is much shorter! You can reuse your existing RPG routines.

I've heard from many System i shops where they end up spending most of their web development time trying to make web services or stored procedures that let their existing RPG code work from a Java or PHP web page. If it's critical to keep using RPG as the back end, why is it so important to change the language you use for the front end? Surely reusing your RPG code is easier if you use CGIDEV2.

Having said that, there are lots of advantages to the alternatives as well:

- Java (via JSP or Servlets) code is compiled (even the HTML), which gives it a bit of a performance boost. CGIDEV2 has to read an HTML file and search out the sections. However, this isn't as big of a deal as it might sound. CGIDEV2 remembers the HTML templates it loads, and keeps track of the timestamp when they were last changed, so it doesn't have to reload it if it hasn't changed (assuming, of course, that CGIDEV2 has remained in memory). Plus, because RPG is generally a faster language that Java, it can afford to spend a few extra CPU cycles interpreting the HTML template.

- PHP and Java do not use the CGI interface (actually, they *can* use it, but most people avoid it). Instead, PHP uses a special plugin so that the web server itself understands the PHP language, and for Java the web server itself (Tomcat or WAS) is written specifically for running Java code, it pre-loads the JVM and provides a class loader that loads the classes into memory and re-runs them from memory. This gives it a performance boost, especially on other platforms. However, this is mitigated quite a bit by ILE's ability to leave things in memory.

- Java is capable of running in multiple threads (not sure about PHP, but I think it is as well) which saves the overhead of running multiple jobs. Granted, in a small scenario, it's not a big deal to have 5-10 jobs already loaded to handle CGI requests, but in a large shop where 10,000 might be needed, the difference between threads and jobs becomes very important. This is an important consideration if you run a very high-volume site, but not so important for smaller shops.

- JSP/Servlet and PHP have a "session" built in, where the language itself keeps track of each user from request to request. You can store variables in that session so that you know where yoru programs left off, as well as other stuff (what items were added to a shopping cart, for example). In CGIDEV2, you'd have to write your own code to track sessions.

- Another really big problem with RPG/CGI is the fact that so few programmers are available. If you want to hire someone, you pretty much have to train them yourself. It's not as mainstream ad the other techniques. You'll have no problem finding a PHP programmer or a Java programmer, you can't turn a corner without tripping over one, but CGIDEV2 programmers are relatively hard to find.

- Along the same lines, since CGIDEV2 and the other RPG CGI techniques aren't as maintstream, there's less tooling available. For example, WDSC knows about writing Java for the web, but knows nothing about RPG for the web, so you don't get its syntax checking capabilities (for the web part, anyway -- you still get it for the basic RPG operations). It's also harder to find articles, add-ons, and other free help on the internet, but of course, not having to learn a whole new programming language to use RPG might mitigate that somewhat -- assuming that you're already an RPG programmer.

Personally, I think the different alternatives all have their pros and cons. If you work in an RPG enviroment, and plan to stay in an RPG environment, CGI programming with CGIDEV2 (or one of the other toolkits) can make a lot of sense for you.

Java has a ton of features, and has unparalleled scalability. It has proven over time to be a robust language for business. However, it requires more hardware to run efficiently, and there's a big learning curve to get started (unless you already know it.)

PHP is simple to use, and runs relatively well even on small machines. It's not as fast as RPG, and it's not as scalable as Java. The weakly typed variables that it uses might not be ideal for business applications, so you might want to stick with an RPG back-end to a PHP front-end. However, it's the most widely used web development language in the world, and it's easy to find qualified PHP programmers. It's very well supported by Zend, who has put a lot of time and money into the System i market.

In the end, the choice is up to you, your company, and your clients. I hope that this article has provided you with some food for thought.