Part II of The Pattern to Good ILE

with RPG IV

Presented by

Scott Klement

http://www.scottklement.com

"There are 10 types of people in the world.
Those who understand binary, and those who don't."

Objectives Of This Session

- Demonstrate a CGIDEV2 Web front-end to the RPG business logic described in part I.
- Describe how ILE business logic can be extended to non-ILE languages through SQL
- Demonstrate a web-front end written in PHP

This is part II. Familiarity with the code presented in part I of this series is useful in understanding this talk. (But a summary will be provided.)

Summary Of Part I

In part I of this series, I talked about:

- We all develop "patterns" in our minds, and use them when we write code.
- ILE concepts don't fit well with our old pattern. To use them effectively, a new pattern must be learned.
- A new pattern should not only fit well with ILE concepts, but should produce code that's easier to test, maintain, and re-use.
- Scott has noticed a 40% productivity improvement using his new pattern.
  - ✓Coding takes a little longer.
  - ✓Testing and debugging takes far less time, and is easier.
  - ✓Code is easy to re-use, making future projects faster.
- New pattern involves:
  - ✓Separate program into Business Logic (Model), User Interface (View), and Program Flow (Controller) components, and keep them distinct.
  - ✓Encapsulation, Stateless Business Logic, Name spaces, SOA.

Re-Using the Pattern

Following the pattern means it's possible to replace one part of the application without changing the others.

Model:

- When your business rules change, changing the model gives you new business rules without needing to change the other parts. "Agility"
- You may also decide to store your business logic and databases on another server. By replacing the Model, you can do that without any upheaval to the user.

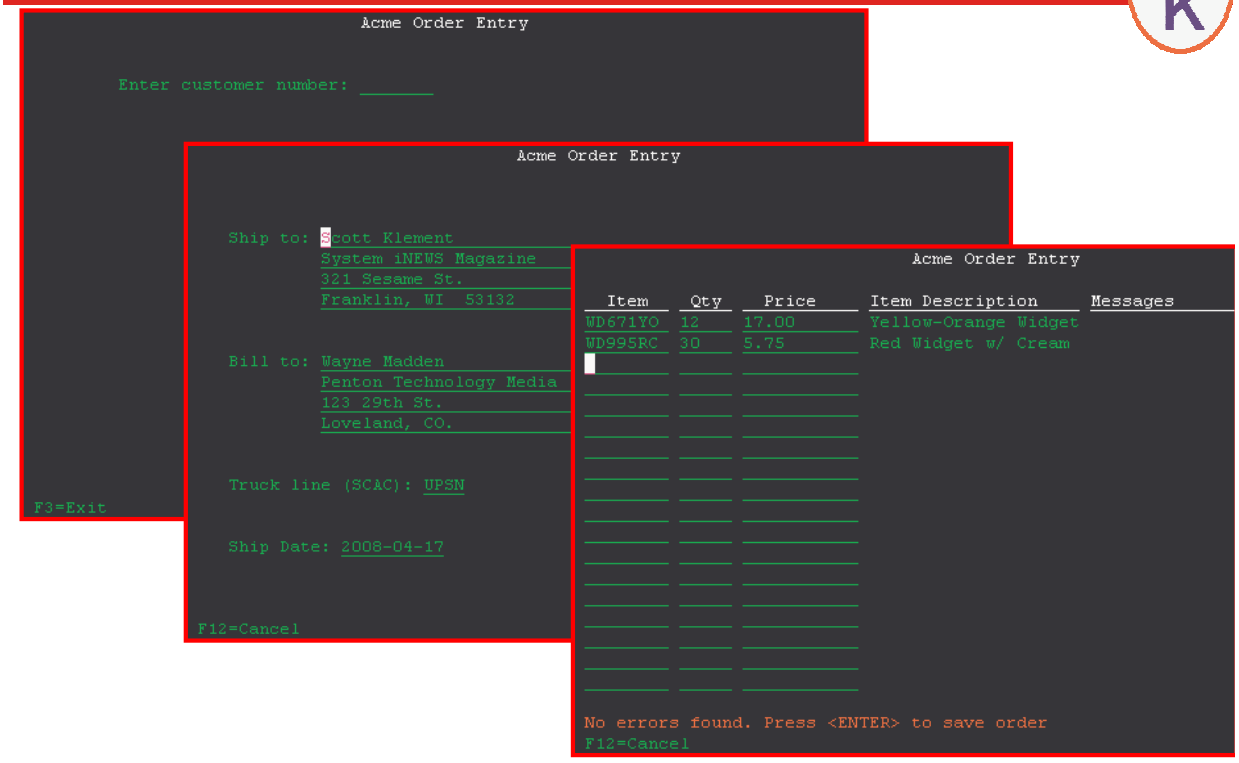
Controller:

- If you change the view, you'll almost certainly also have to change the controller, since different UI paradigms require a different program flow.
- Also, you might use a different controller while testing your routines.

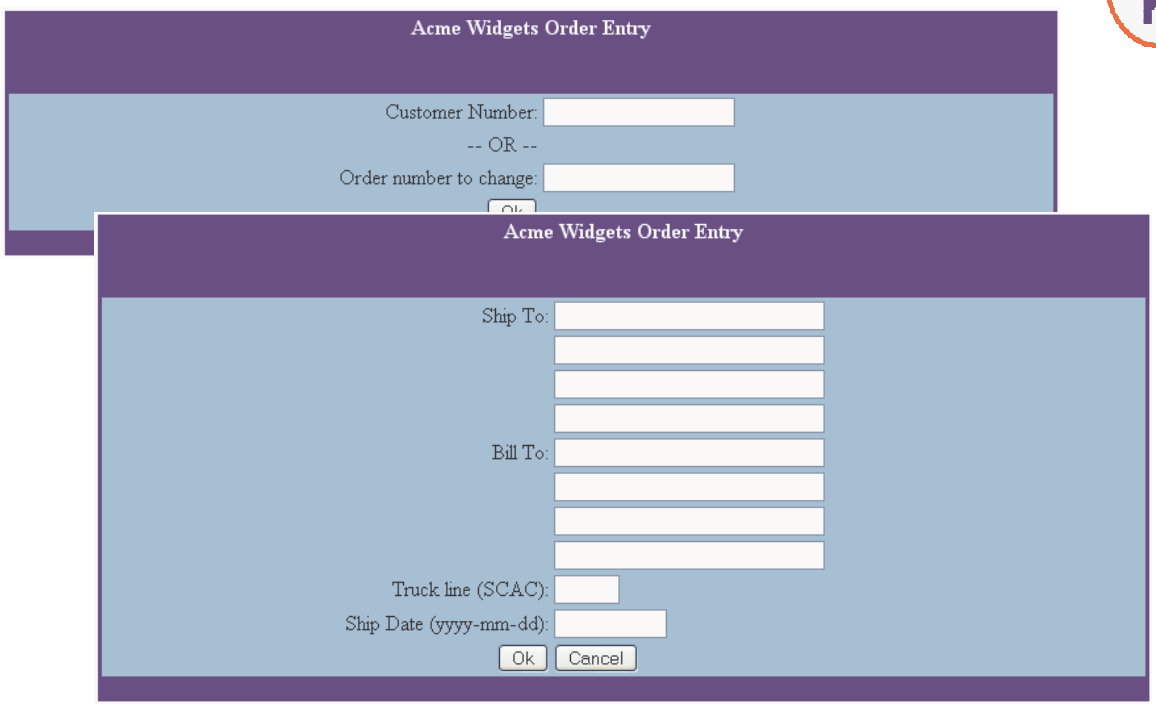
View:

- Perhaps the hottest topic today: Put a different UI on an existing app, or have multiple UIs for a single app.
- *This is the focus of this presentation.*

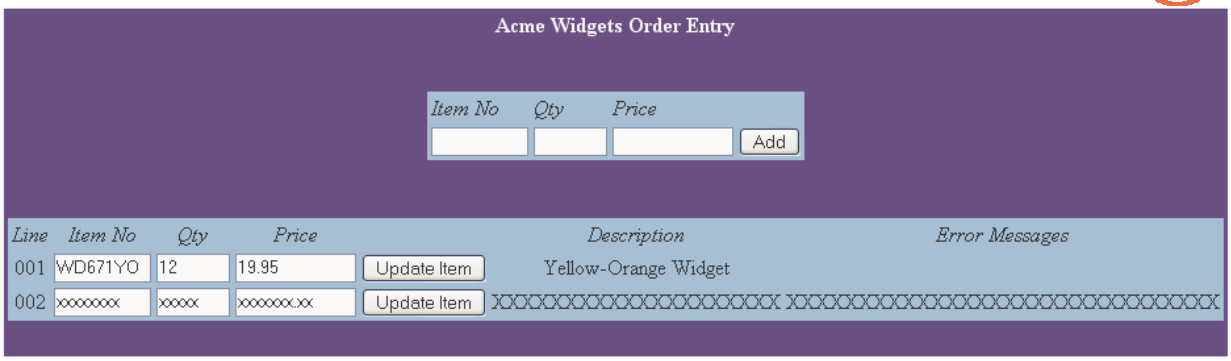
A Traditional View



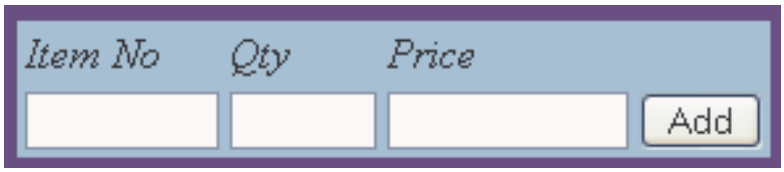
The Web View (1 of 2)



The Web View (2 of 2)



Statelessness of Web Apps



Each time you click a button, or follow a link, in an web application, the browser creates a new request to the HTTP server.

- In the case of a web app – that tells the HTTP server to run a program.
- It's basically a CALL PGM(MYWEBPGM) command.
- The program is passed all of the input (form) information.
- The program processes this input, and writes out a new web page.
- When the program ends, the page is displayed in the browser.

The only time your program runs is in-between the time a link is clicked, and the time it takes to display the next screen.

This means that your program cannot control the flow of screens like we are used to in traditional RPG. Instead, the browser controls them, and a new call is made with every button press, or click of a link.

Statelessness Challenges

Due to the "statelessness" of having a separate, individual call for each click, the following challenges have to be overcome:

- Your program doesn't know if the user will click the next click, or hit the "back button", or close the browser.
- Since you may have many users clicking links at the same time, you never know if the next call of your program will be from the same user running the same "job" as the last call.
- This means you can't remember variable values from call to call.
- In fact, you can't (reliably) set up anything in a previous call that will be used in subsequent calls.

Stateless Pgm Flow (Controller)

First call to program:

- Shows the screen for order/customer number.
- Sets a hidden parm on that page (action=LoadHeader)
- Program ends, and user sees page.

Second call to program:

- Sees that action is LoadHeader
- If order number given, calls ORDER_loadHeader() to load order into memory.
- If no order number, uses customer number to create new order by calling ORDER_new()
- Creates HTML page with order details, and action=EditItems
- Program ends, user sees page.

… etc …

CGIDEV2

CGIDEV2 is not a programming language. It's a toolkit from IBM to simplify the job of writing a web application in native ILE RPG.

- Runs in the native, free HTTP server provided with i5/OS (Apache or Original)

It provides routines:

- To make it easier to output HTML to a browser.
- To make it easier to read input from fields filled in on a web page.

Writing HTML Output:

- Divide your HTML into sections (like "record formats" in DDS)
- Insert replacement values (like "fields" in DDS)
- RPG code can insert replacement values, then write the sections
- Sections can be written repeatedly, to repeat elements that go out to the browser.

These examples use `<!-- $SECTIONNAME$ -->` and `<!-- %REPVAL% -->`

CGIDEV2 Controller

```
GetHTMLIFS('/acmecgi/acmecgir4.html':
           '<!-- $':'$ -- >':'<!-- %':'% -->');

zbhGetInput(SavedQryStr: QUSEC);

Action = zhbGetVar('action');
if (action='' or action=*blanks);
   action='AskOrdCust';
endif;

select;
when action = 'AskOrdCust';
   AskOrdCust();
when action = 'EditHeader';
   LoadHeader();
   EditHeader();
when action = 'EditItems';
   LoadItems();
   EditItems();
when action = 'PlaceOrder';
   SaveOrder();
   ShowResult();
endsl;

return;
```

Load HTML template into CGIDEV2's memory.

Specify what to search for for sections/variables.

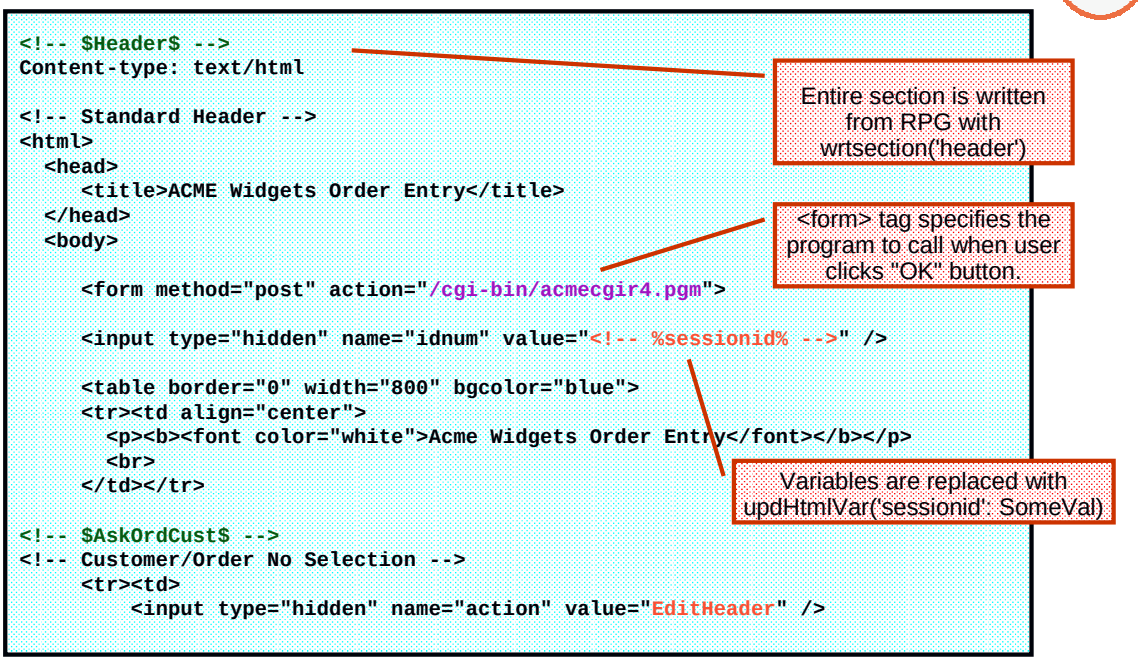
ZhbGetInput() loads all variables ("parameters") from browser into CGIDEV2's memory.

ZhbGetVar() returns one variable from CGIDEV2 to your program

Each time the program is called, action is set to a different value to specify which screen we're handling…

- …all values are character strings…

CGIDEV2 View (HTML) 1 of 2



```
<!-- $Header$ -->
Content-type: text/html

<!-- Standard Header -->
<html>
  <head>
     <title>ACME Widgets Order Entry</title>
  </head>
  <body>

     <form method="post" action="/cgi-bin/acmecgir4.pgm">

     <input type="hidden" name="idnum" value="<!-- %sessionid% -->" />

     <table border="0" width="800" bgcolor="blue">
     <tr><td align="center">
       <p><b><font color="white">Acme Widgets Order Entry</font></b></p>
       <br>
     </td></tr>

<!-- $AskOrdCust$ -->
<!-- Customer/Order No Selection -->
     <tr><td>
         <input type="hidden" name="action" value="EditHeader" />
```


CGIDEV2 View (HTML) 2 of 2

```
         <table border="0" width="100%" bgcolor="lightblue">
         <tr>
           <td align="right">Customer Number:</td>
           <td align="left"><input type="text" name="cust"
                                   maxlength="6" width="6" /></td>
         </tr>
         <tr>
           <td align="right">-- OR --</td>
           <td align="left"> </td>
         </tr>
         <tr>
           <td align="right">Order number to change:</td>
           <td align="left"><input type="text" name="order"
                                  maxlength="10" width="10" /></td>
         </tr>
         <tr>
           <td align="right"><input type="submit" value=" Ok "></td>
           <td> </td>
         </tr>
         </table>

         </form>

     </td></tr>
     </table>

  </body>
</html>
```

The first two <input> tags produce the blanks for the order and customer.

The last <input> is the OK button.

CGIDEV2 View (RPG code)

```
P AskOrdCust      B
D AskOrdCust      PI
 /free
    wrtsection('header');
    wrtsection('AskOrdCust');
    wrtsection('bottom');
    wrtsection('*fini');
 /end-free
P                 E
```

On the next call, the program starts by loading the user's order/cust…

```
P LoadHeader      B
D LoadHeader      PI

 /free
    order  = zhbGetVar('order');
    cust   = zhbGetVar('cust');
    errmsg = *Blanks;
```

These variables come from the <input> tags in the HTML.

… continued on next slide…

CGIDEV2 Combined View/Controller

… code continued from previous slide …

```
    if ( order<>'' and order<>*blanks );
       if ORDER_loadHeader(Order: Hdr) = *off;
          errmsg = ORDER_error();
       endif;
    else;
       if ORDER_new(Cust: Hdr) = *off;
          errmsg = ORDER_error();
       endif;
    endif;

    wrtsection('header');
    if errmsg <> *blanks;
       updHtmlVar('errormsg': ORDER_error() );
       wrtsection('error');
    else;
       updHtmlVar('shipaddr1': Hdr.ShipTo.Name);
       updHtmlVar('shipaddr2': Hdr.ShipTo.Addr1);
       . . .  copy all other needed fields . . .
       wrtsection('EditHeader');
    endif;
    wrtsection('bottom *fini');
 /end-free
P                 E
```

Since this is an ILE RPG program, it can call the procedures in the MODEL directly as ILE procedures!

Non-ILE languages will have to jump through an extra hoop or two (as you'll see later.)

A Note About CGI

Web programming with CGI has gained a bad reputation in today's world.

Here's why:

- Interpreting the input from the browser is complex.
- The output HTML has to be coded into your program
- You must manually escape your output data to be valid in HTML.
- A new "process" (or "job" in OS/400 terminology) must be launched for each call to your program.

Unfortunately, this reputation has somehow spread to CGIDEV2 on i5/OS. CGIDEV2 is not CGI programming! Rather, CGIDEV2 prevents you from having to do the low-level work that would be required in traditional CGI.

- It takes care of interpreting the data from the browser, so you don't have to.
- It lets you keep your HTML outside of your program code
- It takes care of escaping output for you.
- And, on i5/OS, the HTTP server has NEVER required starting a new process for each call to the program! You can load your program into an ILE activation group so that it remains in memory. In that case, subsequent calls are INCREDIBLY fast, because all you're doing is calling a routine in memory.

Therefore, CGIDEV2 should not be thought of as "CGI programming."

Does CGIDEV2 Have a Future?

CGIDEV2 is free software from IBM, it can be downloaded from the IBM Systems and Technology Group (STG) at the following link:
http://www-03.ibm.com/systems/services/labservices/library/

There has been much controversy about whether IBM will continue to support CGIDEV2? Some say that IBM intentionally ignores CGIDEV2 in hopes of selling more of their WebSphere products? (But I don't know if this is true?)

There are more than 20,000 shops using CGIDEV2, and it's my opinion that if IBM ever stops providing it, someone else (possibly me!) will come out with an open source toolkit that's compatible.

The other big problem with CGIDEV2 is the difficulty finding developers. When you need to hire a new developer, you usually have to hire someone with RPG expertise, and train them in CGIDEV2. It's hard to find experienced individuals.

For these reasons, some folks think it's a good idea to use a different tool for web development.

Why PHP?

In my shop, we primarily use RPG/CGIDEV2 for web development, and are very happy with it. However, for those who would prefer an alternative, I suggest PHP:

- PHP is the #1 language for web development, worldwide.
- PHP is the #5 programming language (for all disciplines) according to the Sept 2008 TIOBE index (behind Java, C, C++ and Visual Basic).
  source: www.tiobe.com
- PHP is quickly growing in the IBM i community. Currently in 4th place behind RPG, Cobol and Java. (It has already surpassed C and Net.Data on the i)
- It's very easy to find PHP programmers to hire.
- It's very easy to find PHP code samples on the web.
- PHP is extremely easy to learn.
- PHP is free/open-source.

However, PHP is not an ILE language. How do you call the RPG Model from PHP?

Reusing the Model from Non-ILE Apps

As ILE objects, service programs can only be called from other ILE code, right? Wrong. Here are a few ways that you can call a service program from a non-ILE language:

- PHP and Java have toolboxes for calling IBM i software directly.
- Web Services
- External Stored Procedures (SQL) – *my preference!*

Stored procedures are callable from just about anywhere:

- .NET, ASP, Java, PHP, NET.DATA, Visual Basic, C/C++, even Microsoft Office!
- Can be on the same machine, or different machine (via ODBC or JDBC)

I always write a separate ILE sub procedure to be called from the stored procedure – never call the "regular" ILE procedure directly. This stored procedure interface will call the "regular" routine, but will do some massaging of the data. (I call this a "wrapper")

Why use a wrapper?

- Result sets for output parameters (Meta data for returned variables)
- Enables ILE to call directly
- Enables a façade over the error handling

Configuring Stored Procedure in SQL

To define the stored procedure to SQL so that SQL statements can use it, and it knows where to find the service program, etc, run the create procedure statement like this (one-time):

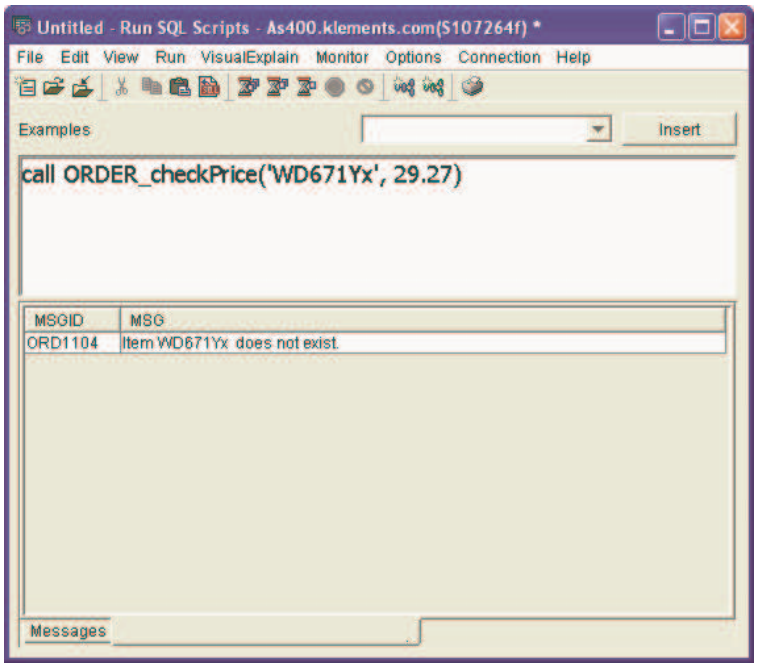
```
CREATE PROCEDURE ORDER_CHECKPRICE(
          IN CustNo    CHAR(10),
          IN ItemNo    CHAR(8),
          IN Price     DECIMAL(9,2)
        )
        LANGUAGE RPGLE
        NOT DETERMINISTIC
        CONTAINS SQL
        EXTERNAL NAME 'SCOTTLIB/ORDERR4(ORDER_CHECKPRICE_SP)'
        PARAMETER STYLE GENERAL;
```

The code, above, says:

- When CALL ORDER_CHECKPRICE is run from SQL
- call the ORDER_CHECKPRICE_SP routine in the ORDERR4 service program in SCOTTLIB
- Pass the following parameters:
  - CustNo, as 10A
  - ItemNo, as 8A
  - Price as 9P 2

Calling SQL Procedure (1 of 2)

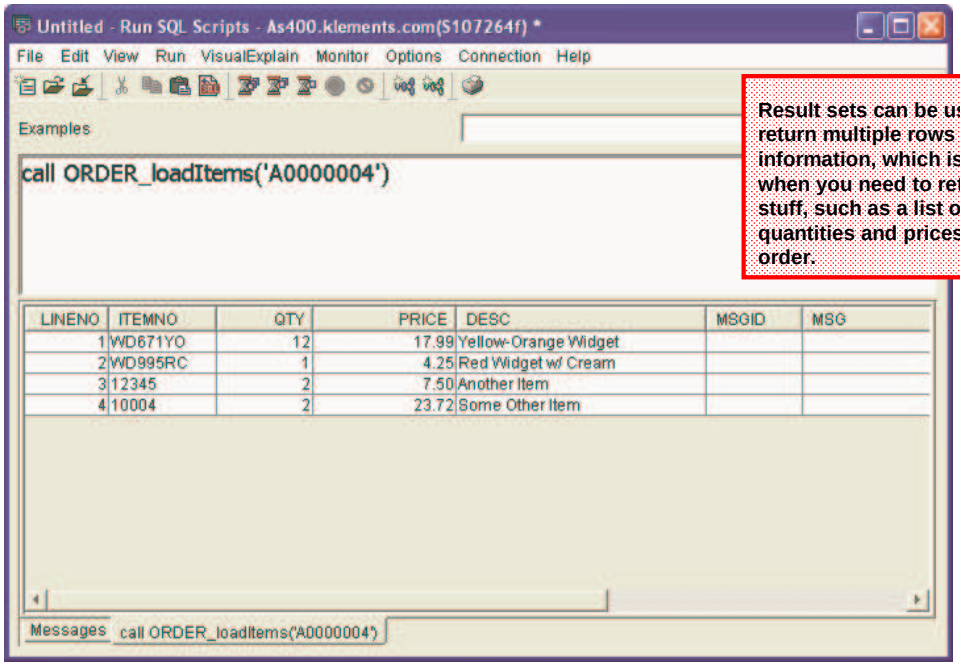
Now the procedure can be called from an SQL statement like this one:



Again, this statement can be run from anywhere you can run SQL statements. In this example, I'm running it from iSeries Navigator's Run SQL Scripts – which is a great place to test your Model code to make sure it's solid before you begin writing your controller/view.

Calling SQL Procedure (2 of 2)

Now the procedure can be called from an SQL statement like this one:



Result sets can be used to return multiple rows of information, which is great when you need to return lots of stuff, such as a list of items, quantities and prices on an order.

Sample Stored Procedure Wrapper

```
P ORDER_checkPrice_sp...
P                 B                   export
D ORDER_checkPrice_sp...
D                 PI
D  CustNo                        8a   const
D  ItemNo                       10a   const
D  Price                         9p 2 const

D Result7         ds                  qualified occurs(1)
D  MsgId                        10i 0 inz
D  Msg                          80a   varying inz
 /free
      %occur(Result7) = 1;
      if ( Order_CheckPrice(CustNo: ItemNo: Price) = *OFF);
         Result7.Msg = ORDER_error(Result7.MsgID);
      endif;

 /end-free
C/exec SQL set Result sets Array :Result7 for 1 Rows
C/end-exec
P                 E
```

Input data comes from SQL as parameters.

The information in this DS (field names, sizes, data types) will be communicated as "meta data" in the result set. Callers can use that information

Calls the "regular" routine and the error message routine – so there's no duplication of code – still just one place to change rules.

Returns the output data to SQL as a result set.

Extending the Binder Language

```
STRPGMEXP SIGNATURE('ORDERR4 ver 1.00')
   EXPORT SYMBOL(ORDER_new)                    #1
   EXPORT SYMBOL(ORDER_loadHeader)             #2
   EXPORT SYMBOL(ORDER_loadItems)              #3
   EXPORT SYMBOL(ORDER_saveHeader)             #4
   EXPORT SYMBOL(ORDER_saveItem)               #5
   EXPORT SYMBOL(ORDER_checkItem)              #6
   EXPORT SYMBOL(ORDER_checkPrice)             #7
   EXPORT SYMBOL(ORDER_checkQuantity)          #8
   EXPORT SYMBOL(ORDER_error)                  #9

   EXPORT SYMBOL(ORDER_new_sp)                 #10
   EXPORT SYMBOL(ORDER_loadHeader_sp)          #11
   EXPORT SYMBOL(ORDER_loadItems_sp)           #12
   EXPORT SYMBOL(ORDER_saveHeader_sp)          #13
   EXPORT SYMBOL(ORDER_saveItem_sp)            #14
   EXPORT SYMBOL(ORDER_checkItem_sp)           #15
   EXPORT SYMBOL(ORDER_checkPrice_sp)          #16
   EXPORT SYMBOL(ORDER_checkQuantity_sp)       #17
ENDPGMEXP
```

Remember that bound procedures from a srvpgm are called "by number".

I did not change the signature – so there will be no need to re-compile or re-bind existing programs.

The new procedures are added at the end, so none of the previously used numbers has changed – everything remains backward compatible!

PHP

PHP is a script language – very much like the others used for writing Unix shell scripts (Bourne Shell, Korn Shell, Perl, QShell, etc). However, it was designed with a strong focus on being particularly suited for web programming.

- An interpreted (non-compiled) script language.
- Runs under Apache, but in a separate (PASE) instance.

It provides:

- Arrays (automatically populated) containing input from the browser.
- A very easy means of writing out HTML code.
- Hundreds of built-in functions.
- Thousands of additional functions (added as an extension)

Syntax:

- Any data typed into a PHP document is assumed to be HTML output that will be sent, as-is, to the browser.
- Anything insider <?php and ?> delimiters is PHP program code, and will be run, and it's output will be sent to the browser.
- Anything that starts with $ (dollar sign) is considered a variable.

PHP "Hello World"

To write PHP, you can embed it into the middle of an HTML document. Anything outside of the <?php and ?> tags is written to the browser as-is. The stuff between those tags is your PHP program code.

```
<html>
<body>

<?php

// You can put any PHP code here.

echo "Hello World";
$info='Yes!';
?>

<p>Nice day, isn't it? <?php echo $info;?></p>

</body>
</html>
```


PHP's Fancy Arrays

PHP has very powerful support for arrays. It's one of the nicer features of the language. Arrays in PHP are numbered from 0 (whereas RPG is numbered from 1)

PHP Code

```
$var = $myarray[0];
$foo = $myarray[1];
```

RPG equivalent

```
var = myarray(1);
foo = myarray(2);
```

PHP has something called "associated arrays" that let you associate array values with words instead of numbers.

PHP code

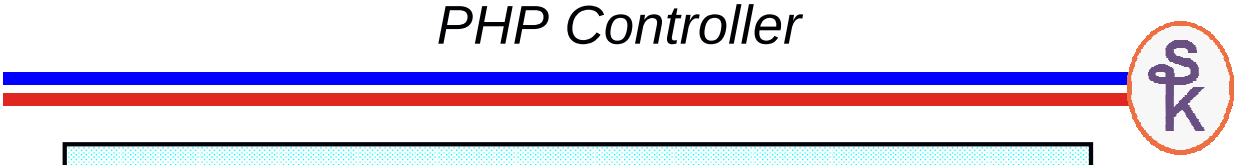
```
$var = $mysales['January'];
```

RPG equivalent *(but, RPG requires two related arrays… or a table)*

```
x = %lookup('January': mymonths);
var = mysales(x);
```

Unlike RPG, PHP uses a sort of a "keyed index" on it's array, much the way that database files work, except it's entirely in memory. That makes array lookups very fast and efficient. PHP programmers use arrays a lot. They work well both as an array, and also for situations where an RPGer might typically use a data structure.

PHP Controller



```
<?php
   isset($_POST['action']) ? $action=$_POST['action'] : $action="AskCustOrd";

  switch ($action) {
  case "AskCustOrd":       // first call
    AskCustOrd();
    break;

  case "EditHeader":       // second call
    LoadHeader();
    EditHeader();

  case "EditItems":        // third call
    LoadItems();
    EditItems();
    break;

  case "PlaceOrder":       // fourth call
    SaveOrder();
    ShowResult();
    break;
  .
  .
```

PHP automatically populates the $_GET and $_POST arrays with the data submitted from the browser.

PHP HTML Output

```
<html>
  <head>
     <title>ACME Widgets Order Entry</title>
  </head>
  <body>

     <form method="post" action="<?php echo $_SERVER['PHP_SELF']; ?>">

      <input type="hidden" name="idnum" value="<?php echo $sessionid ?>" />

     <table border="0" width="800" bgcolor="blue">
     <tr><td align="center">
       <p><b><font color="white">Acme Widgets Order Entry</font></b></p>
       <br>
     </td></tr>

     <!-- Customer/Order No Selection -->
     <tr><td>
         <input type="hidden" name="action" value="EditHeader" />

        <table border="0" width="100%" bgcolor="lightblue">
         <tr>
           <td align="right">Customer Number:</td>
           <td align="left"><input type="text" name="cust"
                                   maxlength="6" width="6" /></td>
         </tr>
```

PHP View

```
function AskOrdCust() {
    require_once("acmeords1.php");
}
```

require_once() is like /COPY in RPG – inserts the contents of another file into the PHP program.

Next call, it'll load the order data….

```
function LoadHeader() {

   isset($_POST['order']) ? $order=$_POST['order'] : $order="";
   isset($_POST['cust']) ? $cust=$_POST['cust'] : $cust="";
   $errmsg = "";

   if ( $order!="" ) {
       if ( !ORDER_loadHeader($order, $hdr) ) {
          $errmsg = ORDER_error();
       }
   } else {
       if ( !ORDER_new($cust: $hdr) ) {
          $errmsg = ORDER_error();
       }
   }

   if ( $errmsg != "" ) {
       require_once("acmeords2.php");
   } else {
       require_once("acmeords2.php");
   }
```

Unlike the RPG version, these are not direct calls. Instead, these are PHP functions that use SQL to call the underlying RPG functions…

PHP Calling RPG (1 of 2)

Since the RPG code is called via SQL statements, PHP needs to connect to the database first. This is required in order to run the SQL statements.

```
function ORDER_connect() {
    $conn = db2_pconnect("","","",
                         array("i5_naming"=>DB2_I5_NAMING_ON,
                               "i5_commit"=>DB2_I5_TXN_NO_COMMIT) );
    if (!$conn) {
        echo "<b>Connect Failed: " . db2_conn_errormsg() . "</b>";
        return FALSE;
    }
    return $conn;
}
```

The first three parameters to db2_pconnect() are set to "", "", "" – and they specify the host name, userid and password, respectively. You can set them to nothing when connecting to the database on the same computer.

Note: db2_pconnect() is for connections established once and re-used throughout all PHP code on the system. It'll keep the connection between calls to the PHP code to improve performance. But, this might have security drawbacks.

There's also a db2_connect() that has to re-connect with every call to the script.

PHP Calling RPG (2 of 2)

Each routine in the model is coded something like this in PHP. It simply takes it's parameters and constructs an SQL statement. Then returns the results in another parameter…

```
function ORDER_loadHeader($order, &$hdr) {

    $conn = ORDER_connect();
    if (!$conn) return FALSE;

    $sql  = "CALL ORDER_loadHeader('" . $order . "')";
    $stmt = db2_exec($conn, $sql);
    if (!$stmt) return FALSE;

    $row  = db2_fetch_assoc($stmt);
    if (!$row) return FALSE;

    $hdr['orderno']  = $row['ORDERNO'];
    $hdr['custno']   = $row['CUSTNO'];
    $hdr['shipname'] = $row['SHIPNAME'];
    $hdr['billname'] = $row['BILLNAME'];

    . . . and so forth, for all of the fields in the header...
}
```


Links to April 2008 Articles

Articles from the April 2008 issue of System iNEWS magazine:

RPG and the Web: Technologies to Get There (Scott Klement)
http://systeminetwork.com/article/rpg-and-web-technologies-get-there

RPG and the Web: The CGIDEV2 Way (Paul Tuohy)
http://systeminetwork.com/article/rpg-web-cgidev2-way

RPG and the Web: The PHP Way (Tony Cairns)
http://systeminetwork.com/article/rpg-web-php-way

RPG and the Web: The Java or Groovy Way (Don Denoncourt)
http://systeminetwork.com/article/rpg-web-java-or-groovy-way

From System iNetwork Programming Tips e-newsletter:

Writing Reusable Service Programs (Scott Klement)
http://systeminetwork.com/article/writing-reusable-service-programs

This Presentation

You can download a PDF copy of this presentation (as well as Part I) from:

http://www.scottklement.com/presentations/

Thank you!