# RPG Meets the Web

# Part 2 - The CGIDEV2 Library

OCEAN Technical Conference
Catch the Wave

Jon Paris
Jon.Paris@ Partner400
www.Partner400.com

Partner400
Your Partner in AS/400 and iSeries Education

The CGIDEV2 library is featured in an article authored by Jon Paris and Susan Gantner which was published in the July edition of iSeries Magazine. This article was a follow up to the February and March articles which focussed on basic CGI functions.

The author, Jon Paris, is co-founder of Partner400, a firm specializing in customized education and mentoring services for AS/400 and iSeries developers. Jon's career in IT spans 30+ years including a 10 year period with IBM's Toronto Laboratory. Jon now devotes his time to educating developers on techniques and technologies to extend and modernize their applications and development environments.

Together with his partner, Susan Gantner, Jon authors regular technical articles for the IBM publication, eServer Magazine, iSeries edition, and the companion electronic newsletter, iSeries EXTRA. You may view articles in current and past issues and/or subscribe to the free newsletter at: eservercomputing.com/iseries.

Feel free to contact the author at: Jon.Paris @ Partner400.com

This presentation may contain small code examples that are furnished as simple examples to provide an illustration. These examples have not been thoroughly tested under all conditions. We therefore, cannot guarantee or imply reliability, serviceability, or function of these programs.

All code examples contained herein are provided to you "as is". THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE EXPRESSLY DISCLAIMED.

Unless otherwise noted, all features covered in this presentation apply to all releases of RPG IV. Where a function was added in a later release, I have attempted to identify it. For example V3R2+ indicates that the feature is available at V3R2, V3R6, V3R7, etc.

## *"Native" CGI problems*

Partner400

### The API's are not as simple as one would like

- And the documentation is less than wonderful
  - I'm being polite here!

### Generating HTML directly in the program reduces flexibility

- To change the text, one must change the program
  - And it is much more difficult to use HTML editors to design the pages

### It would be nice to have something like DDS

- That way only the variable content is handled by the program
  - But writing such a generalized routine to process HTML text and perform text substitution is not a trivial task

### IBM has an answer

- And it is FREE !!

### The CGIDEV2 Library

- Available for download at www.easy400.ibm.it

As with any type of programming, once you start working in CGI you quickly realize that building up a collection of frequently used routines will save you a lot of time. If you have been using RPG IV for a while, hopefully you know by now that the best way to build these routines is through the use of subprocedures.

Many RPG programmers have reinvented this particular wheel, but there's no need for you to add to their number. IBM's CGIDEV2 library was produced by IBM's Custom Technology Center in Rochester. At one time it was offered as a chargeable item, but is now available for download free-of-charge. Don't confuse this offering with the previous version of CGIDEV that IBM made available. This version is far more powerful and significantly faster in operation.

As we hope to convince you in the balance of this presentation, adoption of this library as the foundation for you your future web explorations can save you hours of effort and frustration. No only this, but by their nature and construction they also serve as an excellent teaching vehicle to show you just how sets of such routines can be designed and built. Even if you don't like everything about the way the code operates, it is supplied as source so you can modify the base routines to match your own needs.

If you download from IBM Italy's Easy400 web site (www.easy400.ibm.it) you will also find tutorials, sample programs (including a work-alike version of Amazon.com) and much more. We will show you some examples later in the presentation.

Notes

## Review of the "Native" CGI Approach

Partner400

Remote browser
http://GROMIT/cgi-bin/CreditChk?custname=Smith

Main API's

**QtmhGetEnv**

**QtmhRdStdin**

**QtmhCvtDb**

**QtmhWrStout**

Response HTML

**<HTML><BODY>The credit limit for customer Smith is $5,000</BODY></HTML>**

"Native" CGI Program

Determine input method used (GET or POST)

If GET retrieve query string
If POST determine input length and retrieve data from standard in

Map input to external data structure

Perform program logic

Prepare output HTML

Send output HTML

Return to caller

This page illustrates the basic CGI technique used in most published works on RPG CGI programming.

The function of the QtmhGetEnv and QtmhRdStdin APIs has effectively been replaced by the more recent QzhbCgiParse API. However, examples of the use of this (relatively) new API are few and far between and the IBM documentation is ... well ... less than useful would be a polite way to put it.

Some people have invented their own methods for externalizing the HTML and for "wrappering" the various CGI APIs. See Brad Stone's web page (www.bvstools.com) for examples of this. You will also find reference to his e-RPG books which more fully describe the techniques he uses. Similarly Bob Cozzi has produced some CGI wrappers which you can find at his web site www.RPGIV.com

Notes

## CGIDEV2 Process

Remote browser
http://GROMIT/cgi-bin/CreditChk?custname=Smith

Skeleton HTML

```
/$Response
<HTML><BODY>
The credit limit for
customer
/%custname%/  is
<b>/%crlimit%/</b>
</BODY></HTML>
```

Response HTML

```
<HTML><BODY>The credit limit for
customer Smith is
<b>$5,000</b></BODY></HTML>
```

CGIDEV Program cycle

- Load output HTML skeleton member
- Get input (GET or POST)
- Perform program logic
- Assign values to substitution variables
- Write HTML section(s)
- Send output HTML (section *fini)

This chart outlines the CGIDEV2 approach. While it may be obvious from this chart that this approach significantly simplifies the generation of HTML, many of the ancillary benefits are less obvious.

For example all of the functions in the library have significantly simpler (and often more logical) interfaces than their native API counterparts. Not only do these interfaces more naturally map to the tasks that the programmer is used to performing, but in many cases they supply default values for parameters and have built-in error checking.

In many cases these simple interfaces mask highly complex pieces of code which make extensive use of dynamic memory and pointers to provide the fastest possible response. This is particularly true in the case of the HTML related functions. Trust us - you don't want to write this kind of stuff yourself unless you have to! (and you don't - Mel Rothman has already done it for you).

Hopefully we will convince you of the benefits as we study the functions in more detail.

First though we will take a look at some of the sample applications on the IBM Easy400 web site.

Notes

## CGIDEV2 Comparison - HTML Skeleton

Partner400

### The final version of our program stored HTML as constants

- See the notes for the full text of this section of the program

### This was better than building it "on the fly" but ....

- Not easy to fit it to the limits of the RPG layout
- Can't be modified with a conventional text editor
  - Continuation markers mess it up if nothing else

```
 * Skeleton HTML for table row
D TableRow        C                   '<tr align=right>-
D                                     <td align=left>&Nam</td>-
D                                     </td><td align=center>&Bdg</td>-
D                                     <td>&Pay</td>-
D                                     <td>&Tax</td>-
D                                     <td>&Net</td></tr>'
```

This is the complete set of constants used as the HTML skeleton in the original program. Note that each part of the HTML was coded as a separate constant and was subsequently loaded individually into the buffer.

Since the formatting requirements of RPG differ significantly from those for HTML the resulting HTML is messy and very hard to edit.

```
D HTTPHeader      C                   'Content-type: text/html'
D NewLine         C                   X'15'

D TableHeader     C                   '<html><body><center>-
D                                     <B> Pay and Tax Calculation for: -
D                                     &Dat</B></center><BR><center>-
D                                     <table border=1 CellSpacing=1 -
D                                     CellPadding=3 Width="80%">-
D                                     <th>Name<th>Badge<th>Earned-
D                                     <th>Pay Tax<th>Net Pay'

D TableRow        C                   '<tr align=right>-
D                                     <td align=left>&Nam</td>-
D                                     </td><td align=center>&Bad</td>-
D                                     <td>&Pay</td>-
D                                     <td>&Tax</td>-
D                                     <td>&Net</td></tr>'

D NextPage        C                   '</center></table><BR>-
D                                     <A HREF="/cgi-bin/PAYCGI5?BADGE=-
D                                     &Bad"> Next Page </A>'

D NoMore          C                   '<tr><td>No more records</td></tr>-
D                                     </center></table><BR>-
D                                     </body></html>'
```

Notes

## CGIDEV2 Comparison - HTML Skeleton

Partner400

### CGIDEV2 stores skeletons as source members or in the IFS

- The full text is reproduced in the notes

### Substitution variables and Section names are highlighted

- Substitution variables begin with "/%" and end with "%/"
- Section names begin with "/$"
- These delimiters can be changed if required
  - More on this later

```
/$TableRow
<TR ALIGN=RIGHT>
<TD ALIGN=LEFT>/%Name%/</TD>
<TD ALIGN=CENTER>/%Badge%/</TD>
<TD>/%Pay%/</TD>
<TD>/%Tax%/</TD>
<TD>/%Net%/</TD>
</TR>
```

This is the full text of the HTML template used in the sample program. We have highlighted the text substitution fields in **bold**, section headers are **bold and underlined**. Notice that the substitution field /%Badge%/ appears in both the TableRow and NextPage sections. We only have to load this value once, the routines will extract the stored value and insert it as we write the section.

```
/$Init
CONTENT-TYPE: TEXT/HTML

<HTML><BODY><CENTER>
<B> Pay and Tax Calculation for: /%Date%/</B>
</CENTER>
<BR><CENTER>
<TABLE BORDER=1 CELLSPACING=1 CELLPADDING=5 WIDTH="600">
<TH>Name<TH>Badge<TH>Earned<TH>Tax Due<TH>Net Pay
/$TableRow
<TR ALIGN=RIGHT>
<TD ALIGN=LEFT>/%Name%/</TD>
<TD ALIGN=CENTER>/%Badge%/</TD>
<TD>/%Pay%/</TD>
<TD>/%Tax%/</TD>
<TD>/%Net%/</TD></TR>
/$NextPage
<TR><TD><A HREF="PAYCGI4?BADGE=/%Badge%/">
<B>Next Page</B></A></TD></TR>
/$NoMore
<tr><td>No more records</td></tr>
/$Trailer
</TABLE></BODY></HTML>
```

**Note:**
**Blank line needed after CONTENT-TYPE:**

When developing our own programs, we normally use a delimiter of "<!--$" for the section name in place of the default value of "/$" shown in the example above. This is recognized as a comment by the browser, which makes it easier to design and test the HTML using conventional tools. We will show you how to change this default later when we discuss the HTML functions. In the examples in the handout, we have (for the most part) used the CGIDEV2 default.

## CGIDEV2 Comparison - Reading Input Data

### The original used QtmhGetEnv to read the input

- This presumes that the GET method was used by the form
- If data was found it was assumed to be the Badge number
  - We should really have used APIs to parse the string

### CGIDEV2 uses ZhbGetInput to processes the data

- This works regardless of the method used (GET or POST)
- The Badge data is retrieved using ZhbGetVar

```
C                   CallP     GetEnvData(EnvData : %Len(EnvData) :
C                                        EnvDataLen : QueryString :
C                                        %Len(QueryString) : APIError )
 * If data found assume it was Badge number
C                   If        EnvDataLen <> 0
```

```
 * Retrieve Input (if any)
C                   If        ZhbGetInput(SavedQryStr:APIError) > 0
 * Variable found - Assume it was a Badge Number and retrieve it
C                   Eval      Badge = ZhbGetVar('Badge')
```

## CGIDEV2 Comparison - "Loading" HTML

### The original program stored the HTML as constants

- Then loaded the appropriate constant into the Output buffer
  - Later %Replace was used to merge field data into the template

### CGIDEV2 stores the HTML in source members or the IFS

- And uses the GetHTML procedure to load it into memory
  - Or GetHTMLIFS if you have stored the HTML in the IFS
- Not only is it simpler, but its intent is more obvious
  - It also loads all of the HTML templates in one operation
- Source stored in IFS is easier to modify with your favorite HTML editor

```
 * Load basic row template and then substitute variables for markers
C                   Eval      WebData = TableRow + NewLine
```

```
 * Load HTML source skeleton
C                   CallP     GetHtml('QRPGLESRC':'*LIBL':'PAYCGIHTML')
```

## CGIDEV2 Comparison - Setting variables

### The original program used %REPLACE

- A powerful function but its intent is not that obvious
  - And it has to be run multiple times if the field appears multiple times

### CGIDEV2 uses UpdHTMLVar

- Much more obvious what it is doing
  - Maps to a MOVE or EVAL from the source field to the Display/Print field
- It stores the replacement value internally
  - It will perform the actual substitution later
    - More on this in a moment

```
 * Substitute content of Badge field for &Bdg marker
C                   Eval      WebData = %Replace(Badge:WebData:
C                                                %Scan('&Bdg':WebData):4)
```

```
 * Load Badge field content into substitution variable
C                   CallP     UpdHTMLVar('Badge':Badge)
```

## CGIDEV2 Comparison - Writing the HTML

### The original program uses the QtmhWrStout API

- You either have to call it multiple times
  - Once per table row in our example
- Or perform your own buffering to improve efficiency

### CGIDEV2 uses the WrtSection API

- Very similar to writing a format to a Display or Printer file
  - But can write multiple formats with a single call
- It "plugs in" the values for all replacement variables in the HTML
- Automatically buffers output for efficiency

```
 * Note "WriteWebData" is the prototype name for QtmhWrStout
C                   CALLP     WriteWebData(WebData : %Len(WebData)
C                                          : APIError)
```

```
 * Add row to table
C                   CallP     WrtSection('TableRow')
```

## IBM's Easy400 Web Site

Partner400

### www.easy400.ibm.it

- The development resource for CGIDEV2 programming
  - There is also an associated mailing list at groups.yahoo.com

### The screen shots are from two of the demo applications

- They are here in case we can't connect to an AS/400
  - And also to serve as a reminder to you when you return home

### "Yachtworld"

- Allows you to search for and purchase (you wish!) a luxury yacht

### "Centaur"

- Allows you to purchase books
  - It is based on Amazon.com but does not attempt to implement all features
- Demonstrates "Shopping Cart" support

### There is also "Websiter/400"

- A tool for generating an event registration web site

The Easy400 web site is a wonderful resource that contains far more than we can ever hope to tell you in this brief introduction.

In addition to the ability to download the CGIDEV2 library itself and the various demonstration programs (including all sources) you will find the following:

Demonstrations:

- All of the demo programs can be run on the Easy400 web site if you want to demonstrate to folks what RPG can do. Not only can they be run, but in most cases each page also contains links that will show you the actual source code that executed and the HTML skeleton that was used.

Tutorials:

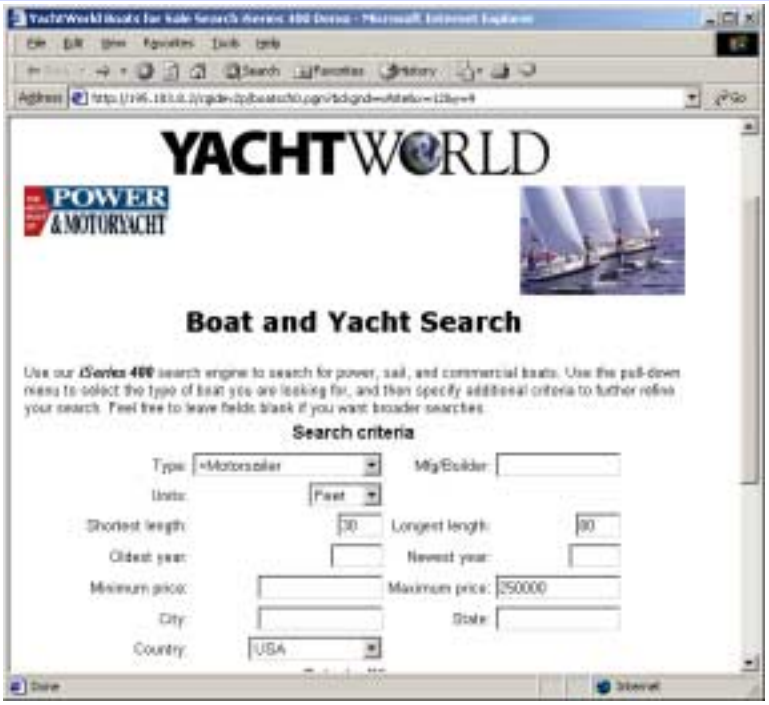
- Configuring OS/400 for TCP/IP communications
- Configuring the HTTP server (Traditional and Apache)
- Securing your web site
- Basic HTML
- Basic JavaScript
- and much, much more

Utilities for:

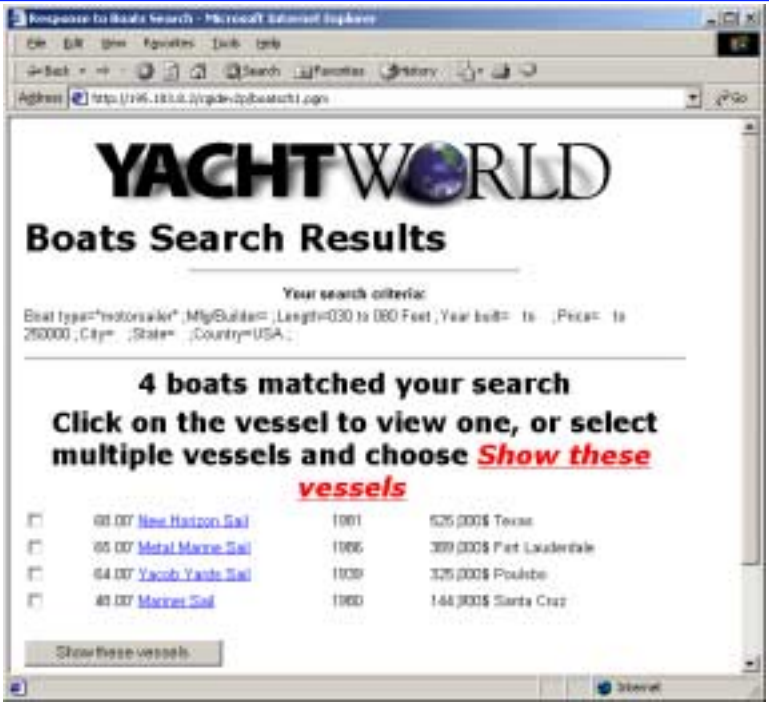
- Configuring your HTTP server
- Securing your system
- Packaging files for upload/download
- and more

Notes

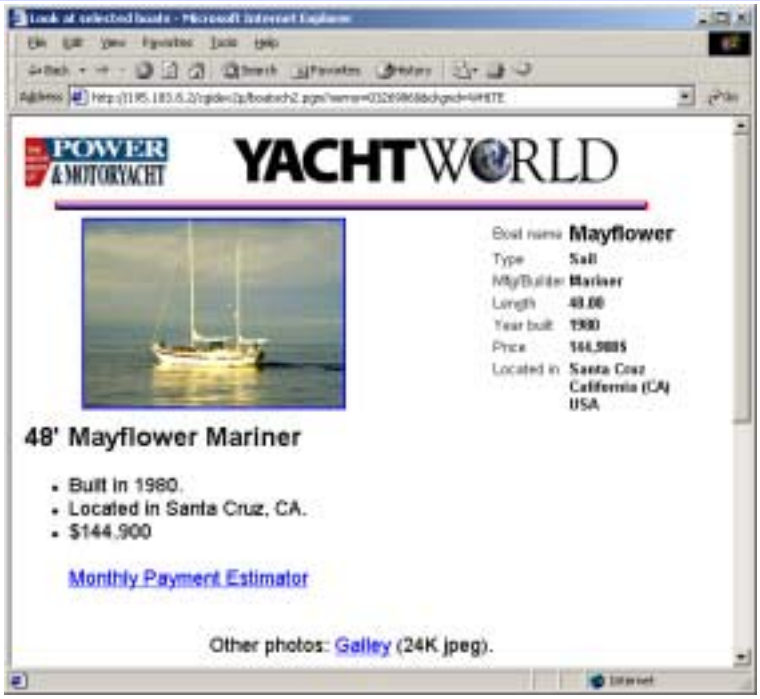
## Initial Screen for "YachtWorld" Demo



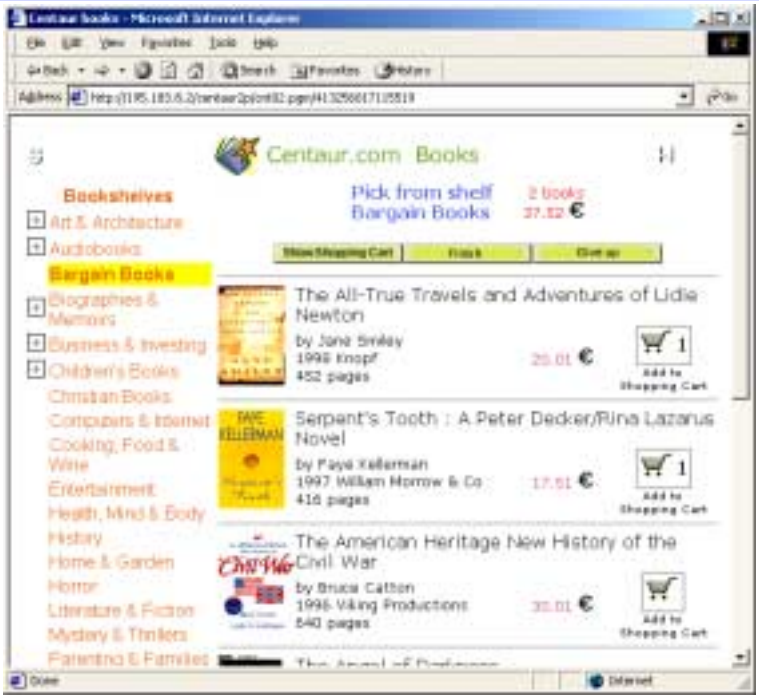

## "YachtWorld" Demo (contd.)



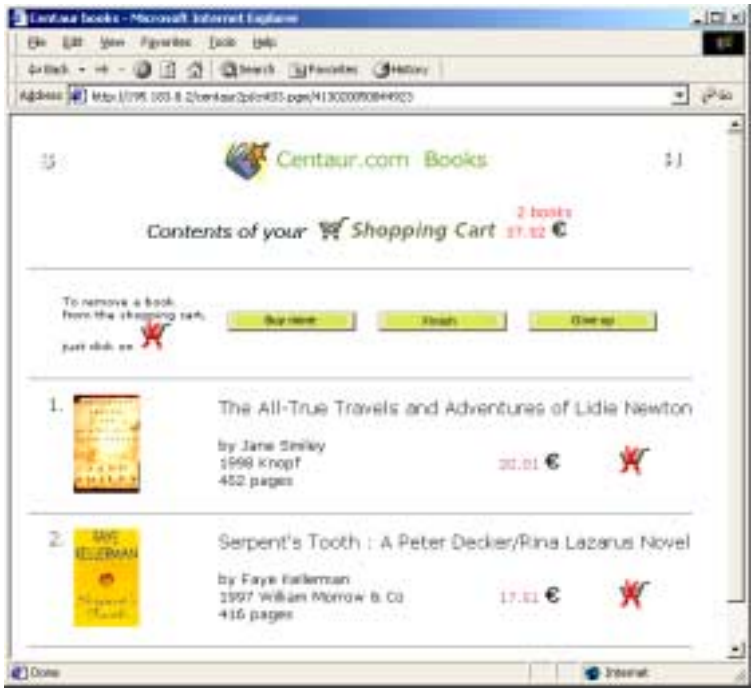
## "YachtWorld" Demo (contd.)




## "Centaur" Demo



# "Centaur" Demo (contd..)




If we haven't been able to connect to the web during the class and all you've seen are these screen shots, you might be excused for not believing just how good this stuff is.

Don't take our word for it - go home and try it for yourself - www.easy400.ibm.it

It will be worth your while.

Notes

## CGIDEV2 Templates

### Five templates are currently supplied

- Based on the Computer Discount Warehouse demo programs
  - You can try these on the Easy400 site if you don't (yet) want to download the whole CGIDEV2 library

### Templates 1 and 2 use "old" techniques

- They are mostly there for people who need compatibility with Version 1

### Templates 3, 4 and 5 use the latest methods

- The differences between the three are:
  - Template 3 uses a source physical file to store the HTML
  - Template 4 uses the IFS to store its HTML
  - Template 5 uses getHTMLIFSMult to allow the HTML to be split into multiple files

### We will do a brief walk through of Template 3 in a moment

One part of the CGIDE2 library that is worthy of mention, because it is not obvious when you download the software, is the collection of template programs.

In a moment we will take a quick look at TEMPLATE3, but there are a number of others that you can study. Each represents a different way of using the CGIDEV2 functions to perform the same basic web task.

Each template is "marked" in the sequence number area to indicate the role of that line in the template. For example a record containing blanks in the sequence area should be left in the program. One with xxxxx is an example e.g. file handling, optional HTML output etc. One containing ccccc should be changed to tailor the template to your personal needs.

The TEMPLATE and TEMPLATE2 programs use the older CGIDEV2 routines. These are delivered as part of the current package mostly for compatibility reasons. If you study the code provided for the underlying subprocedures, you will also see excellent examples of how to use the basic CGI APIs that we discussed in the earlier CGI session.

In terms of the input methods used, program TEMPLATE uses the GetInput, cvtdb, cgivarcnt and cgivarval functions. Program TEMPLATE2 uses GetInput, cgivarcnt, and cgivarval. TEMPLATE3 is our preferred approach and uses ZhbGetInput, ZhbGetVarCnt and ZhbGetVar.

PS. You may notice that in samples and in the CGIDEV2 documentation use of capitalization of the function names is somewhat inconsistent. You can use any convention that you wish - the routines are all exported from their associated Service Programs with a traditional all upper-case RPG name.

## TEMPLATE3 - H and F Specs

### Notice the use of /COPY for standard H-specs

- And of course for Prototypes and the standard API Error structure

```
      /copy cgidev2/qrpglesrc,hspecs
      /copy cgidev2/qrpglesrc,hspecsbnd

      * For files that are not in the CGI program's library,
      * use the docmd subprocedure to add libraries to the LIBL
      * or to issue overrides before opening the file(s). Although
      * not required in this program, the file HoursOp uses UsrOpn

XXXXXFHoursOp   IF   E           K Disk    Usropn
XXXXXF                                     InfSr(*pssr)
      * Copy Prototype defintions and standard API error structure
      /copy cgidev2/qrpglesrc,prototype2
      /copy cgidev2/qrpglesrc,usec
      * Number of variables
     D NbrVars         S             10I 0

      * Saved query string
     D SavedQueryString...
     D                 S          32767    Varying
```

## TEMPLATE3 - Main Line

- In your own code you may need to determine if there was any input
  - i.e. Check that nbrVars is not zero before proceeding
  - The template program does not need this check as it always receives input from a Form

```
      * Mainline
      ************************************************************
      * Initialization
     C                   exsr      Initialize
      * Write qualified job name to debug file.
     C                   callp     wrtjobdbg(*on)
      * Read externally defined output html
     C                   callp     gethtml('HTMLSRC':'*LIBL':'TALK2':
     C                             '<as400>')
      * Get input
     C                   eval      nbrVars =
     C                             zhbgetinput(savedquerystring:qusec)
      *
      * Parse variables returned from HTML Form
xxxxx * Customer name
xxxxxC                   eval      custname = zhbgetvar('custname')
```

## TEMPLATE3 - Input Validation

- GetMsgCnt procedure is used to determine if messages are present
- Indicator "wrotetop" signals to PSSR routine whether header has already been output.

```
xxxxx * Edit input
xxxxx * Name
xxxxxC                   if        custname = *blanks
xxxxxC                   eval      rc = addmsg('Name':1)
xxxxxC                   eval      rc = addmsg('Was blank.':2)
xxxxxC                   eval      rc = addmsg(TryAgain:3)
xxxxxC                   endif
 :                        :
      * Write sections of HTML.
     C                   callp     wrtsection('top')
     C                   eval      wrotetop = *on
      * If any errors, write error messages
     C                   if        GetMsgCnt > 0
     C                   callp     WrtMsgs
     C                   endif
xxxxxC                   callp     wrtsection('top2')
```

## TEMPLATE3 - Wrap up the page

- Note that "*fini" is always the last section written

```
xxxxxC                   callp     wrtsection('tablebot')
xxxxx * If we have e-mail address, say we will send package.  Else,
xxxxx * say we won't and give opportunity to re-enter.
xxxxxC                   if        emailadd = *blanks
xxxxxC                   callp     wrtsection('WeWontSend')
xxxxxC                   else
xxxxxC                   callp     wrtsection('WeWillSend')
xxxxxC                   endif
xxxxx * Send rest of information
xxxxxC                   callp     wrtsection('RestOfInfo')
xxxxx * Get program end time and write execution time to cgidebug file
xxxxxC                   exsr      dotime2
xxxxxC                   callp     updhtmlvar('runtime':
xxxxxC                             %trim(%editw(sec:'     0 .   ')))
xxxxxC                   callp     wrtsection('runtime')
      * Write the *fini section to ensure all buffered output is sent
      * to the browser.
     C                   callp     wrtsection('endhtml *fini')
     C                   return
```

## TEMPLATE3 - Initialize Subroutine

- Notice the call to ClrMsgs to clear out any error messages from the previous invocation

```
     C     Initialize    begsr
      * Initialization

      * Do every time
xxxxx *   Get program start time for calculating execution time
xxxxxC                   exsr      dotime1
      *   Clear messages
     C                   callp     ClrMsgs
      *
      * First time only
     C                   if        not InitComplete
     C                   eval      InitComplete = *on
      *  Set up message handling section names (if default names are
      *  used, there is no need to program this call.  Done for
      *  illustrative purposes).
     C                   callp     CfgMsgs('msgtext':'msgstart':
     C                               'msgend':'msgl1':'msgl2':'msgl3')
     C                   endif
     C                   endsr
```

## TEMPLATE3 - *PSSR subroutine

- Note that it forces completion of the HTML & writes debug data
  - If this is not done, nothing will show up in the browser until it times out

```
     C     *pssr         begsr
      * If have already been in pssr, get out to avoid looping
     C                   if        pssrswitch=*on
     C                   eval      *inlr = *on
     C                   return
     C                   endif
      * Set on switch to indicate we've been here
     C                   eval      pssrswitch=*on
      * Write HTML sections (top if not already done, pssr, and *fini)
     C                   if        wrotetop=*off
     C                   callp     wrtsection('top')
     C                   endif
     C                   callp     wrtsection('pssr endhtml *fini')
      * Send psds data to cgidebug physical file
     C                   callp     wrtpsds(psds)
     C                   eval      *inlr = *on
     C                   return
     C                   endsr
```

## Summary of the CGIDEV2 Library

### CGIDEV2 contains a wide variety of functions

- They provide assistance in all areas of CGI development
  - Supplied in source form so you can modify them to your own requirements
- Demos and program templates to speed up your CGI development
  - We will look at one of the templates in a moment

### Functions are provided to assist with:

- Handling Browser input and output
- Externally describing HTML
- Data handling and validation
- Message formatting and handling
- Error handling and reporting
- Debugging
- Other miscellaneous functions
  - Generation of random number and unique session ids for Persistent CGI
  - Issuing CL commands
  - Escape sequence handling

## Browser Input Functions

### ZhbGetInput

- Retrieves browser input
  - Regardless of input method (GET or POST)
- Loads it in memory for subsequent high speed access

### ZhbGetVar

- Returns the value of a named variable
  - Handles multiple occurrences of the same variable name

### ZhbGetVarUpper

- Works the same as ZhbGetVar but returns the value in upper case

### ZhbGetVarCnt

- Returns the number of occurrences of a given named variable
  - Very useful when dealing with Multiple selection lists and "Subfiles"

### GetInput

- An alternative to ZhbGetInput

## ZhbGetInput and ZhbGetVar

### ZhbGetInput

- Returns a count of the number of variables in the input
  - Multiple occurrences of a single variable count as one
- First parm is the saved query string, second is the API error structure

### ZhbGetVar

- Returns the value in the requested variable as a character string
  - Or null if variable does not exist
- Optional second parm specifies the occurrence number (defaults to 1)

```
 * Get and obtain count of input variables
C                   eval      nbrVars =
C                             zhbgetinput(savedquerystring:qusec)
C                   If        nbrVars > 0
 * Extract Customer name etc. from browser input
C                   Eval      custname = zhbgetvar('custname')
C                   Eval      emailadd = zhbgetvar('emailadd')
C                   EndIf
```

## Some of the HTML Functions

### getHTML, getHTMLIFS and getHTMLIFSMult

- Retrieve named HTML skeleton(s) and stores them in memory
  - Skips the loading if the file is unchanged since last loaded

### updHTMLvar

- Adds or updates a variable's substitution value in the arrays used for variable substitution when HTML sections are written by ...

### wrtSection

- Prepares the named HTML section(s) for output to the browser
  - Performs all required variable substitution before writing the HTML
- Special section name "*fini" is used to force the buffered output

### wrtHTMLToStmF

- Writes all HTML prepared by wrtSection to a stream file (IFS)
  - Useful if you want to use CGIDEV2 routines for other than browser output

### CrtTagOpt

- Creates formatted OPTION tags to dynamically build selection lists

## *getHTML, getHTMLIFS & getHtmlifsMult*

Partner400

### getHTML - Three compulsory parameters

- They identify the file, library and member names
- Four optional parameters allow you to specify alternate start and end delimiters for the HTML section names and substitution variables
  - In the example below the default delimiter of '/$' for the start of a section is being overridden to '<as400>'

### getHTMLIFS/getHTMLIFSMult - Single compulsory parameter

- Identifies the HTML file in the IFS
  - Remaining parameters are optional and match those for getHTML
- getHTMLIFSMult differs in that it returns an error indicator structure

```
 * Read externally defined output HTML
C                   CallP     gethtml('HTMLSRC':'*LIBL':'TALK2':
C                             '<as400>')

 * Same but retrieve source from IFS
C                   CallP     gethtmlifs('/Cgidev2/talk2ifs.html':
C                             '<as400>')
```

When you are using the getHTMLIFS or getHtmlifsMult subprocedures, you must ensure that the IFS files containing the external HTML code can be read by the HTTP server user profile QTMHHTP1. This is the one that is adopted when running CGI.

You can do this either by allowing *PUBLIC read access to the files or (if you prefer to exclude *PUBLIC) by explicitly giving read access to the QTMHHTP1 user profile.

Notes

## updHTMLvar

### Updates internal arrays of variable names and values

- First two parameters are compulsory
  - First parm is the name of the substitution variable
    - The case in which the name is written does not matter
  - Second parm is the new value to be given to that variable
    - Must be a character string
- Third parameter is optional
  - A value of '0' clears the arrays and writes this variable as the first entry
  - A value of '1' replaces the variable if it exists, otherwise it adds it.
- Note: UpdHTMLvar trims leading and trailing spaces from input variables before storing them

```
 * Store substitution values for new page, clearing previous values
C                   CallP     updHTMLvar('custname':custname:InitVars)
C                   CallP     updHTMLvar('emailadd':emailadd)
:                   :         :
 * When all values stored write HTML section
C                   CallP     wrtsection('CustInfo')
```

## Data Handling Functions - chknbr

### Such a useful utility function deserves its own chart!

### Checks if the string in parm one is a valid number

- Returns a data structure containing seven indicators
  - 1 = *On = One or more errors.  *Off = No errors
  - 2 = Non-numeric characters (includes minus sign in wrong place)
  - 3 = Multiple decimal points
  - 4 = Multiple signs (both leading and trailing)
  - 5 = Zero length input or no numeric characters
  - 6 = Too many digits to the left of the decimal point
  - 7 = No errors, but value is negative
- Optional second parm specifies maximum number of digits to the left of the decimal point
  - If not passed it means don't bother to check the number of decimals
- Additional parms can be used to control:
  - If error messages are to be automatically generated using AddMsg
  - If so, the field's description for AddMsg purposes
  - Whether to consider a negative value an error for AddMsg
    - Default is that negative values are not considered an error

## Other Data Handling Functions

### chknbr

- Checks if a character string represents a valid number
  - Checks for and identifies a number of different problems including
    - Non-numeric, Multiple decimal points, Multiple signs, No numeric characters, etc.

### c2n2

- Converts a character string to a packed 30,9 value
  - Use chknbr to validate string before calling c2n2
    - Use this function and not "c2n" which is supplied only for backward compatibility

### char2hex and hex2char

- Convert between character and hexadecimal representation

### uppify

- Convert character string to all upper case

### xlatWCCSIDs

- Uses CCSIDs to translate a string from one CCSID to another

## Message Formatting and Handling

### AddMsg

- Adds a message at a specified format level (1,2, or 3)

### ClrMsgs

- Clear all currently stored messages

### GetMsgCnt

- Returns the current number of stored messages
  - There's a limit to just how many error messages any one user needs!
    - Besides WrtMsgs does not check

### WrtMsgs

- Sends to the browser all currently stored messages
  - Messages are formatted using named sections of the HTML skeleton
    - The function CfgMsgs can be used to override the default names

## Sending Messages

### AddMsg

- Returns zero if message added sucessfully
  - Negative values indictate an error
- Parameter 1 identifies the message
  - Optional parameter 2 specifies the level (1, 2 or 3)

### WrtMsgs

- No parameters - simply writes current messages to browser
  - Condition by using GetMsgCnt to check if messges have been stored

```
C                   If        emailadd = *blanks
C                   Eval      rc = AddMsg('E-mail address':1)
C                   Eval      rc = AddMsg('Was blank.':2)
C                   Eval      rc = AddMsg(TryAgain:3)
C                   EndIf
:                   :         :
C                   If        GetMsgCnt > 0
C                   CallP     WrtMsgs
C                   EndIf
```

## CGIDEV2 Debugging Assistance

### Debugging of CGI programs can be difficult

- And laborious

### CGIDEV2 includes a number of useful debug features

- All main functions automatically write to the debug file
  - If debugging is turned on
- Plus you can write your own entries to the file at any time
  - We will look at the functions for this on the next chart

### The CGIDEBUG command is used to process the log file

- There are options to:
  - Clear Debug data
  - Display Debug data
  - Display Debug status (On or Off)
  - Turn Debug On/Off

## Debugging Functions

### isdebug

- Used to test whether debugging is currently on or off

### wrtdebug

- Information supplied in the first parm is written to the debug file
- Second (optional) param can be used to force output
  - Otherwise no data will be written unless debugging is turned on
    - Via the CGIDEBUG command - not STRDBG
  - Allows you to ensure logging takes place under error conditions

### wrtjobdbg

- Writes current job name, date and time to the debug file

### wrtpsds

- Formats and writes the current PSDS to the log file unconditionally

## Environment Variable Functions

### getenv

- Returns the current value of the requested variable
  - Many "interesting" pieces of information can be retrieved inlcuding:
    - DOCUMENT_NAME (e.g. /QSYS.LIB/NEWCGI.LIB/envvars.PGM)
    - DOCUMENT_URI (e.g. /newcgi/envvars )
    - REMOTE_ADDR (e.g. 192.168.0.12 )
    - REQUEST_METHOD (e.g. GET or POST)
    - SERVER_ADDR (e.g. 192.168.0.5 )
    - SERVER_NAME (e.g. gromit )

### putenv

- Creates or changes the specified environment variable

### contlen

- Returns the content of CONTENT_LENGTH as an integer

## *References*

### The CGIDEV2 web site

- www.easy400.ibm.it
  - The Readme for CGIDEV2 is at www.easy400.ibm.it/cgidev2h/readme.htm
  - The References are at www.easy400.ibm.it/easy400p/ref00.cgi

### Learning HTML

- "HTML for the World Wide Web" by Elizabeth Castro
  - A very useful tutorial/reference backed up by a great web site
    - A straightforward approach to web page design

### Redbooks (www.redbooks.ibm.com)

- Who Knew You Could Do That With RPGIV? (SG24-5402)

### Other Useful Web Sites

- www.ignite400.org
- www.bvstools.com