down, and an optional title. The window includes an alarm to catch the user's attention.

— Lynne Noll
Senior programmer analyst
Battenfeld Gloucester, SMS Group
Gloucester, Massachusetts

PHP

Configure PHP Dynamically with the ini_set() Function

PHP's runtime configurations are stored in a file called PHP.ini. Dozens of settings, including error-handling directives, database options, and syntax conventions, are stored here. Although PHP.ini's location can vary, the default location in Zend Core for i5/OS is /usr/local/Zend/core/etc/php.ini.

For semipermanent, installation-wide settings, PHP.ini is effective. Many situations, however, call for a more dynamic approach. The ini_set() function (visit the manual page at php.net/ini-set) fills this need.

Applications can use ini_set() to spontaneously alter most runtime settings. The changes take effect only in the calling script and expire when the script ends. Basing its parameters on PHP.ini's key names and values, ini_set() is easy to use and useful in many situations, such as when

- a single PHP installation hosts multiple applications with different needs
- the developer lacks editing privileges or connectivity with PHP.ini
- the application must change settings on the fly in response to data
- developers want to experiment with settings without rebooting the web server
- you want to reduce the application's reliance on the development server's settings; the application can generate its own settings based on the environment in which it finds itself

Figure 7 shows examples of using ini_set().

— Alan Seiden
Senior developer/technical lead
Strategic Business Systems, Inc.
Ramsey, New Jersey

FIGURE 7
Examples of ini_set()

```
/* SMTP
   Name of server to be used by mail() function
*/
ini_set('SMTP', 'mail.myserver.com');

/* max_execution_time:
 number of seconds script is allowed before timing out
(default: 30)
 For a script expected to run longer, this value should be
increased.
*/
// allow 120 seconds of processing.
ini_set ('max_execution_time', 120);
// now run long process without risk of time-out


/* include_path:
Path where PHP will try to find files specified by include
and require_once commands.
*/
// Example of dynamically generating a value for ini_set():
// File includefile.inc is located in a folder (/includes)
within the document root. The document root's value, defined
by the web server, is found in the $_SERVER array.
// start with document root
$basedir = $_SERVER[DOCUMENT_ROOT];
// concatenate root with /includes
$includes = $basedir . '/includes';
// now equal to DOCUMENT_ROOT + /includes
ini_set('include_path', $includes);
// file will be found in include_path
require_once 'includefile.inc';
```

Process Check Boxes as an Array in PHP

PHP provides dozens of functions for processing arrays, and thus thrives on array structures. PHP's power to process web forms is attributable in part to its ability to read form data as arrays.

Check boxes allow users to select multiple items from a list on a web form. When a form containing check boxes is submitted, the browser sends only the values that have been checked. In PHP, an array called $_POST contains all values submitted by the form, including check boxes and other data. PHP can process the check boxes separately if you use a special naming convention.

If the check boxes in a list are named identically with a pair of brackets appended to the name (e.g., "item[]"), PHP can view the submitted check boxes as a distinct array.

Figure 8 shows a list of check boxes that follow this convention in the script choose.php. The form is submitted to a second script (process.php) that reads the check boxes. At that point, the values will be in an array that can be easily and conveniently processed.

— Alan Seiden
Senior developer/technical lead
Strategic Business Systems, Inc.
Ramsey, New Jersey

Define a Constant Array in PHP

Constants encourage reliable, self-documenting code. Once defined, a constant's value cannot be changed by code that executes later. The constant's scope is global, accessible (read-only) from anywhere in the script. Despite their usefulness, constants in PHP normally are limited to scalar (single value) data types — boolean, integer, float, and string. However, with a little effort, programmers can use constants to store arrays.

The trick is to store the array as a string. Its value can then be assigned to a constant. Later, when the value is needed, you can convert the string back to an array. PHP's serialize() and unserialize() functions serve this purpose perfectly. Serialize() converts a compound

FIGURE 8
choose.php

```
<html>

<!-- choose.php -->

<!-- submit the form to action="process.php" --><form name="items" method="POST" action="process.php">
<table>

<tr><td>PO#: <input type="text" name="PO" id="PO"></td></tr>
<tr><td>&nbsp;</td></tr>

<tr><td>Choose items in low supply. We will buy more of them.</td></tr>

<!-- Note "[]" following the name "item."
     Naming all the checkboxes the same and inserting the "[]" causes the receiving script to see all "item" checkboxes as an array.
     Note: For illustrative purposes, the list's values are hard-coded.
     In a production application, a database would generate the item values. -->
<tr><td><input type="checkbox" id="item[]" name="item[]" value="CAT11111">Glue</td></tr>
<tr><td><input type="checkbox" id="item[]" name="item[]" value="CAT12223">Pencils</td></tr>
<tr><td><input type="checkbox" id="item[]" name="item[]" value="CAT23342">Pens</td></tr>
<tr><td><input type="checkbox" id="item[]" name="item[]" value="CAT43556">Scissors</td></tr>
</table>
<br><br>
<input type="submit" value="Place order">
</form>
</html>


process.php

<?php
/* process.php:
   list the selected items.
*/

// verify and extract PO# from $_POST array
if (isset($_POST['PO'])) {
 $PONum = $_POST['PO'];
}

// verify existence of 'item[]' checkbox array within larger array of POSTed data
if (is_array($_POST['item'])) {

// extract checkboxes into $item_array
$item_array = $_POST['item'];

/* $item_array contains the selected items.
 We could easily loop through the array to process each item.
 In this example, we will use the "implode" function
 to transform the array into a comma-delimited string
 for display.
*/

// implode: create comma-delimited list (string) of item numbers
$itemlist = implode(",", $item_array);

// display string (comma-delimited list)
echo "We will order more of $itemlist using PO# $PONum";
}

?>
```

object, such as an array, into a string; unserialize() restores the string back to its original type and structure.

Figure 9 shows a constant array that is serialized at the beginning of the script, to be unserialized later when needed.

— Alan Seiden
Senior developer/technical lead
Strategic Business Systems, Inc.
Ramsey, New Jersey

RPG/CL

Use CPYF to Ignore the Key

Occasionally, you might want to copy a keyed file sequentially, ignoring the key. The CPYF command defaults to using the keyed sequence for the target file. To get around this, change the following parameter:

```
CPYF   FROMFILE(X)  TOFILE(Y)  FROMRCD(1)
```

Changing FROMRCD from the default of *START to a value of 1 tells the CPYF command to ignore the keyed sequence.

— Wally Day
Senior programmer/analyst
Miicor Consulting, Inc.
Boise, Idaho

The Q Command

I use Q more than any other System i command, so I created a version on nearly all of our client machines. It is useful not only for quick data views, but also for retrieving general information about a file. Creating the command is a snap and takes just a couple of minutes.

```
CRTDUPOBJ OBJ(RUNQRY) FROMLIB(*LIBL)
          OBJTYPE(*CMD) TOLIB(QGPL) NEWOBJ(Q)
CHGCMDDFT CMD(Q) NEWDFT('rcdslt(*yes)')
```

You can name the new command anything you like, and the target library can be something other than QGPL, but it should be a readily accessible library. Using the command is simple: On a command line, type q *n file-name.

The first panel that appears is the selection criteria prompt. You can use this view to determine field names, data types and lengths, and descriptive information. Pressing F18 provides record format information. To view actual data values, type the appropriate selection criteria (or leave blank to view all data) and press Enter.

— Wally Day
Senior programmer/analyst
Miicor Consulting, Inc.
Boise, Idaho

FIGURE 9
Serialized constant array

```
<?php

/* Compare the current day of the month with a list of
paydays stored in a constant array. An announcement is
displayed to show whether today's date matched any day in the
list.
*/

/* Define constant PAYDAYS: Array of days that are paydays
   The serialize() function converts the array of paydays
into a string, assigned to constant PAYDAYS.
*/
define('PAYDAYS', serialize(array(1, 15))); // payday is
always the 1st and 15th of the month

// Get today's day of the month (e.g. 25)
$today = date('j'); // 'j' extracts day of the month

// Restore array of paydays with unserialize().
$payrollDays = unserialize(PAYDAYS);

// Check if today's date is in the array of paydays
if (in_array($today, $payrollDays)) {
        echo 'Today is payday. Process payroll here.';
}
else {
   echo 'Today is an ordinary day. No payroll processing.';
}

?>
```

Open Browser or Other PC Program from i5/OS

Perhaps the most common question I receive from the forums and from readers of my newsletter is, "How do I open a browser from my System i program?" Most users today use PCs running 5250 emulators to access the System i, and that makes it easy. There's a simple CL command called STRPCCMD that tells the 5250 emulator to run a PC command on the Windows system.

Figure 10 shows a CL program that accepts a UPS tracking number as a parameter. When you call this program, it opens a browser on the PC that points to UPS's website for tracking packages.

A browser is just the beginning! You can run any PC command this way, as long as it's less than 123 characters long.

— Scott Klement
System iNEWS technical editor

FIGURE 10
Opening a browser from CL

```
PGM   PARM(&TRACKNUM)

      DCL VAR(&TRACKNUM) TYPE(*CHAR) LEN(18)
      DCL VAR(&CMD) TYPE(*CHAR) LEN(123)
      DCL VAR(&CARAMP) TYPE(*CHAR) LEN(2) VALUE(X'BØ5Ø')

      CHGVAR VAR(&CMD) VALUE('start http://wwwapps.ups.com+
                             /WebTracking/processInputRequest?+
                             TypeOfInquiryNumber=T"&"+
                             InquiryNumber1=' *CAT &TRACKNUM)

      STRPCO
      MONMSG MSGID(IWS4Ø1Ø)

      STRPCCMD PCCMD(&CMD) PAUSE(*NO)
ENDPGM
```

Execute Commands with the system() Function

It's common to use the QCMDEXC or QCAPCMD API when you want to execute a CL command from an RPG program. However, you might find it more convenient to use a C runtime library function called system() to accomplish the same purpose. The system() function passes a command string to the command processor without the need to pass the length of the command string — or any other parameters for that matter.

To call the system() function, you simply pass it a pointer to the command string. Figure 11 shows the suggested prototype (along with some necessary H-specs).

FIGURE 11
Prototype for system() function

```
 /If Defined(*Crtbndrpg)
H Dftactgrp(*No)
 /Endif
H Bnddir('QC2LE')

  // -------------------------------------------------------- Prototypes
D GoCmd           PR            1ØI Ø Extproc('system')
D   CmdString                     *   Value
D                                     Options(*String)
```

When it's time for your program to execute a command, you can refer to the prototype. The command string may be a variable, literal, named constant, or an expression. Figure 12 shows a typical use.

The return code lets you check for the success or failure of the system() function. The return code is zero if the command is successful or 1 if the command fails. If you pass a null pointer to a string, system() returns -1, and the command processor is not called.

If the system() function fails (i.e., return code is 1),