The PL/I Connection

Issue 5 IBM Santa Teresa Laboratory December, 1995

Editor: Shauntel Christensen

Season's Greetings

As this year comes to a close, IBM and especially our PL/I Development team wants to wish you an enjoyable holiday season and a prosperous new year. We value our association with you and look to the future with anticipation as we work together to make PL/I the language of choice.

PL/I for OS/2 CSD's...Hot Off the Press

The PL/I for OS/2 Corrective Service Diskettes (CSD's) are created to provide two things:

- Fixes to customer problems
- Product enhancements between releases.

Currently, the Personal and Professional Editions of PL/I for OS/2 are at CSD level 6 and the PL/I for OS/2 Toolkit is at CSD level 2.

Since CSD's for PL/I products are cumulative, you can get all of the fixes and updates provided in early CSD's by applying the most recent one. An overview on how to locate CSD's online is at the end of this article.

A README file provides a detailed description of what is included on each CSD. If you have missed CSD announcements in the past or have wondered about the value of applying CSD's, read on. Although this is just a summary of CSD enhancements, you should have a good idea what a difference applying these CSD's can make.

To start, a number of enhancements have been made to improve host compatibility. Just a few examples are:

- Array expressions can now be used as arguments to user functions or as the source in a multiple assignment.
- New suboptions of the DEFAULT compiler option are now supported.
- The restrictions on the STRING built-in function and on multiple REFERs now match the host compiler.

New compiler options are now supported:

- The INCAFTER compiler option lets you can specify the file to be included after all PROCESS statements have been read.
- The CURRENCY compiler option has been added so you can specify what character should be recognized as the 'dollar sign' in picture strings.

New built-in functions are available for your use:

- FILEID returns a file's handle
- RANK gives the decimal value for a character
- SYSTEM passes a command string to the operating system
- VALIDDATE indicates if a string holds a valid date
- FILEREAD reads a specified number of storage units from a file to a buffer
- FILEWRITE writes a specified number of storage units from a buffer to a file
- PLIASCII converts storage units from EBCDIC to ASCII
- PLIEBCDIC converts storage units from ASCII to EBCDIC.

We have made a number of improvements to I/O, including support for BTRIEVE files.

We have added function to PLITEST and included a new PL/I actions profile for those of you who use Workframe Version 2.1.

The SQL preprocessor and DCLGEN utility now work with DB2/2 Version 2.1.

If you purchased the PL/I for OS/2 Toolkit, CSD#2 contains support for the following:
- LPEX editor (including an online user's guide)
- ILINK linker (including online help)
- ILIB
- New C2PLI options and other updates from CSD#1

(As you would expect, the extra function requires some additional space on your hard drive, about 11.5 MB.)

These are just a few examples of what you can add to your compiler or toolkit by applying CSD's. And...to find out what's coming up...stayed tuned to The PL/I Connection!

Where can I get CSD's?

There are separate CSD's for the Professional Edition and Personal Edition of PL/I for OS/2. You should look for CSD#6 for these two products. The latest CSD for the PL/I for OS/2 Toolkit, however, is CSD#2 (the one that contains LPEX and ILINK).

Anonymous FTP

The CSD's are available on *ftp.software.ibm.com* in the */ps/products/pli/fixes* directory. Use the loaddskf utility to make diskettes from the diskette images. The loaddskf utility should be in */pub/os2/os2fixes*. The files should be named as follows:

- PLPCS6 - Professional Edition
- PLWCS6 - Personal Edition
- PLTCS2 - Toolkit

IBM Talklink

The CSD's are on TalkLink on the OS/2 Bulletin Board System (OS2BBS) in the Software Library. You should be able to find them in the OS2 CSD section.

CompuServe

You can access the CSD's on CompuServe by typing GO OS2DF1 to get into the OS/2 Developers Forum. Then access library 6 and browse the entries to find the CSD you need.

OS2CSD (for IBM internal customers)

Use the TOOLCAT OS2CSD command and search on PLI.

Table of Contents

To 'C' or Not to 'C' Is Recoding a Legacy PL/I Application to C The Right Thing to Do?

by Richard Perkinson, Liant Software Corporation
dickp@lpi.liant.com

This is the last of four articles. The series began in the March, 1995 issue of "The PL/I Connection".

Part 4 - Recompiling vs. Recoding: The Bottom Line

It is true that C or C++, not PL/I, is the language of much new development on open system platforms. More new programmers are being trained in C and C++ rather than PL/I. However, the reality is also that there is a lot of existing PL/I that can be rehosted simply and safely with recompilation rather than translation.

The following example shows the difference in costs to recompile versus recode. There may be some alteration of legacy code required due to language implementation differences when recompiling. These types of modifications top out at less than 2% of the total code and are usually done internally. Therefore, if you had an application of some 250,000 lines of legacy PL/I code (typical PL/I applications range from 250,000 to 6,000,000 line of code) it would cost you at most \$2,500 to recompile versus as much as \$250,000 to recode.

	Lines of Code	Lines Changed	Cost/ Line	Total Cost
Recompile	250,000	2%	$.50	$2,500
Recode	250,000	100%	$1	$250,000
Translator	250,000	30%	$1	$75,000*
Translation Service	250,000	100%	$.50	$125,000

**Does not include price of translator*

If you can use some type of PL/I to C translator (remember the shortcomings addressed earlier), assume they can convert anywhere from 50-70% of the code to C successfully. If we take the most optimistic prediction of 70%, that would still leave 30% or 75,000 lines to do by hand, that's $75,000!

Keep in mind that some of that code will cost you $1.50 per line because both internal people and external consultants must work together. Also, remember that there will most likely be a major data conversion effort if you use a translator and don't forget the cost of the translator itself. The Translation Service row (in the example) illustrates an offer by one of the code converter companies to give you a cleanly compiled C program from PL/I source for $.50 a line. Keep in mind that "clean compile" doesn't mean the program actually works with your data.

The economic advantage of recompilation speaks for itself.

Summary

PL/I is a language that has many productive and useful years ahead of it. It is a language admirably suited to many problems found in the new open systems environments. Rehosting provides the opportunity to migrate legacy PL/I applications to this new world with controlled risk, controlled cost, protected investment, portability across platforms, and the ability to modernize gradually.

Don't be in a hurry to abandon the old just because it is old. Don't rush to new technology simply because it is new. Don't forget the risk involved in meeting a deadline with new untried technology. Engineering a new solution to an old problem with new technology and with high pressure deadlines can be a recipe for disaster. Get the old system working in the new environment first, then make the switch to new technology where appropriate.

To receive this paper in its entirety, please call 1-800-818-4PLI ext. 221 or (508) 872-8700 ext. 221, or send email to openpl1@lpi.liant.com.

Let's Recode Our C Programs in PL/I!

by Conrad Weisert, Information Disciplines, Inc.

I'm getting tired of all the discussions over whether or not we should rewrite our PL/I programs in C/C++, and I'm puzzled by the continuing impression that the whole purpose of having PL/I for OS/2 or AIX is so that we can salvage old legacy systems from the mainframe. Why the defensive tone? Why aren't we vigorously promoting PL/I as a modern development tool for developing *new* applications?

C (without C++) was a major step back from PL/I. C's virtues were its widespread availability and its small size. It was easy to learn and easy to implement on small computers of the 1970's. Since the desktop platforms of the 1990's are far from small, those virtues no longer carry any weight. With today's wealth of development tools and languages, I see no justification for choosing C for *any* nontrivial new application.

With C++, the picture is quite different. From C's origin as a small language, C++ (with its essential standard libraries) has evolved into the *largest* and most complicated language in the history of programming. By comparison, PL/I, once condemned as the ultimate "large language," is a paragon of elegance and simplicity. C++ does, however, have one strong virtue: support of the full object-oriented programming paradigm (OOP).

There is, therefore, justification for choosing C++ over PL/I for an application in which we plan to exploit OOP. In doing so, however, we give up a lot, and not just the lists of detailed PL/I features cited in recent articles as "requiring special attention." What we lose is the whole approach to program organization and structure that makes well-designed PL/I programs so easy to maintain.

Three areas in particular give PL/I a strong edge over C++ in the fundamental approach to program organization: nested procedures, exception handling, and the powerful macro preprocessor. Any program that exploits those areas will be *impossible* to convert to another language in any direct way, and any attempt to restructure such a program to fit the constraints of another language will yield a maintenance nightmare.

Let's stop trying to stem the tide away from PL/I and take a positive attitude. If IBM and other vendors continue to support it, PL/I is well suited to the new generation of application systems, even for organizations that have no mainframe investment.

How about an article on how to convert your legacy C programs to PL/I?

If any of you have suggestions for promoting the use of PL/I, please submit them to TEAMPLI.

PCR - An Interface between PL/I, CMS, and REXX

by Dave Jones
djones@starbase.neosoft.com

Due to the size of this paper, it will be presented in three parts. This first part covers introductory material and the syntax of the PCR command. Look for Part 2 in the next issue (March 1996) of "The PL/I Connection".

Part 1 - Introduction to PL/I-CMS-REXX (PCR)

This document describes the PCR utility which allows routines written in PL/I to be nucxloaded as:

- CMS commands
- REXX functions or subroutines
- Immediate commands and subcommands

Multiple PL/I routines can be loaded and managed at once, all sharing common data structures.

PCR is similar to NUCXLOAD/NUCXDROP, but is designed to load PL/I modules, and allows them to share data structures and retain allocated data structures between invocations. Modules are given a logical name, with which they are invoked, in addition to a nucxname with which they are loaded. The logical name can be either a REXX function or subroutine, a CMS command, immediate command, or subcommand.

REXX Function Library Support

The PCRRXFN assembler program is a REXX function library which can manage PL/I and Assembler functions. The PCR command is issued to load PL/I programs, and the CMS NUCXLOAD command is used to load Assembler programs. This module (PCRRXFN) must be either renamed to one of the three supported names (RXUSERFN, RXLOCFN, or RXSYSFN), or specified as an extension to one of those names. The PCRRXFN module distributed with PCR supports extensions.

There are four tables within PCRRXFN that can be modified to indicate the supported modules:

PACKTAB -- Each entry in PACKTAB is another function package module that will be loaded and called to see if it has the function that is being requested by REXX. Any number of function packages can thus be listed and supported, much more than just RXUSERFN, RXLOCFN and RXSYSFN that REXX supports. Each entry in PACKTAB is a character 8 and the last entry must be an 8X'FF'.

XFUNTAB -- Each entry in XFUNCTAB represents a REXX function and its corresponding MODULE which can be NUCXLOADed (generally written in Assembler). The entries are character 16, the first eight bytes is the name of the REXX function (must begin with RX), and the last 8 bytes is the name of the Module to be NUCXLOADed. The last entry must be an 8X'FF'.

XPLITAB -- Each entry in XPLITAB represents a REXX function and its corresponding MODULE which is loaded using PCR, thus the module should be from a PL/I program. The entries are character 24, the first eight bytes is the name of the REXX function (must begin with RX), and the second eight bytes is the nucxname of which the module (last 8 bytes) is loaded. The last entry must be an 8X'FF'.

LFUNTAB -- Each entry in LFUNTAB represents a REXX function and its corresponding address. The code for the function is internally linked into PCRRXFN. The entries are character 16, the first eight bytes is the name of the REXX function (must begin with RX), and the next four bytes is the relocatable address of the internally linked routine, and the last four bytes is a flag which should be set to CL4'N'. The last entry must be an 8X'FF'.

Immediate Commands

When the PCR program is used, three immediate commands are defined. They each modify a bit within a byte which each PL/I program has access to. They can be used, or ignored, depending on the PL/I program logic.

- PTS -- Trace start. PL/I programs can check this flag to see if tracing messages are produced.
- PTE -- Trace end. This will turn off the PTS bit.
- PHX -- Halt execution. PL/I programs can test this bit at critical locations within the program's logic, and can thus support a controlled shut down at the user's control. This bit is reset to false when a PL/I program is first entered.

CMS may not invoke immediate commands during tight CPU loops. Some I/O or SVC must be issued before CMS will look for immediate command. This author does not know all the details as to when CMS will actually notice and respond to an immediate command.

Loading PL/I Runtime Modules

Upon first invocation of PCR, zero of more modules from PLILIB/IBMLIB Loadlib are nucxloaded. The PCR code can be modified to list each module to load from each of the two system loadlibs. Modify the tables LPLITAB and LIBMTAB within the PCR Assemble file. An 'FF'x entry must mark the end of the tables.

PCR Syntax

Here is the syntax for the PCR command:

```
PCR LOAD definition <( <options...> >

definition =
 logname|REXX|nucxname<modname<loadlib>>
        |CMSC|
        |SUBC|
        |IMMC|

options = |NOLOAD|
          |ENDCMD|

PCR DROP |logname|
         |*      |

PCR RESET

PCR TERM
```

LOAD

Specifies to load/define a logical name to a PL/I module.

Logname

The logical name of the subcommand, CMS command, immediate command, or REXX function/subroutine being defined, indicated by the type (SUBC, CMSC, IMMC, or REXX). The logname cannot be the same as the nucxname. If a nucxleus extension of this name already exists, an error occurs.

Nucxname

The nucxleus extension of which the PL/I module is loaded. This cannot be the same as the logical name. If a nucxleus extension of this name already exists, a new copy will not be loaded, although the definition of the logical name continues.

Modname

The name of the PL/I module. If omitted, an equal sign is assumed which will indicate to use the nucxname as the modname.

Loadlib

The optional load library from which the PL/I module is loaded.

DROP

Specifies to delete the logical name's definition. The PL/I program associated with the logical name is nucxdropped, only if it was actually loaded when the logical name was defined. If an asterisk is specified as the logical name, all logical names will be dropped.

RESET

Deletes all logical names, and any PL/I program loaded. Since this type of call is made by CMS when a NUCXDROP PCR is issued, the three immediate commands (PTS, PTE, PHX) are also removed.

TERM

Calls the PLISTART that is linked into the PCR module to terminate the PL/I environment. This is also done when CMS returns to "ready", via ENDCMD processing. However, currently, if TERM is done via ENDCMD, an abend occurs. Thus all execs invoking PCR PL/I modules that are compiled with the SYSTEM(MVS) option should have a PCR TERM as the last command so the environment does not get TERMed during

ENDCMD processing. No special provisions need be taken with PL/I modules compiled with the default SYSTEM(CMS) option.

NOLOAD
Specifies that no PL/I module is to be nucxloaded.

ENDCMD
Specifies that the PL/I program is to be called when CMS returns to the "ready" state. Not valid for types IMMC and SUBC.

To be continued in the next issue...

Peter's Performance Tips

by Peter Elderon, IBM PL/I Development
elderon@vnet.ibm.com

Unlike previous issues which discussed new PL/I features, this column describes how some existing PL/I features can help improve your performance.

The following program returns the largest element in a one-dimensional array of fixed bin(15) values.

```
max1_xb15:
  proc( a )
  returns( fixed bin(15) )
  ;
   dcl a(*) fixed bin(15);
   dcl m    fixed bin(15);
   dcl jx   fixed bin(31);
   m = a(lbound(a,1));
   do jx = lbound(a,1) upthru
           hbound(a,1);
     if a(jx) <= m then;
     else
       m = a(jx);
   end;
   return( m );
end;
```

This program works for a one-dimensional array with any lower bound. However, if you know that this routine is always invoked with an array that has a lower bound of 1, you can improve the performance of the routine by changing the declare for a as follows:

```
dcl a(1:*) fixed bin(15);
```

Additionally, if you know that this routine is always invoked with an array that is connected, you can improve the performance of the routine by changing the declare for a as follows:

```
dcl a(1:*) connected fixed bin(15);
```

If you know that all the arrays passed as arguments in your application are connected, you do not have to edit your program to make the change in the second declare statement. You would just compile the program with the option DEFAULT(CONNECTED).

This routine works only for arrays of FIXED BIN(15) values. You could write similar routines that performed the same task for arrays of FIXED BIN(31) and FLOAT BIN(53) values. If these routines were called `max1_xb31` and `max1_fltd`, for example, then you could also declare the following generic function:

```
dcl max1 generic
  ( max1_xb15 when( (*) fixed bin(15) )
   ,max1_xb31 when( (*) fixed bin(31) )
   ,max1_fltd when( (*) float bin(53) )
  );
```

The result is a function, `max1`, that returns the maximum value of any one-dimensional array having one of the three types specified. In addition, a compiler error message would be produced if the routine were invoked with a scalar or any other kind of array. You have effectively created your own built-in function.

While this generic function might not help the performance of your executed code, it--or examples like it--can help you code faster. It can also serve as a useful reminder of some of the less frequently mentioned, but still very powerful features of PL/I

IBM's Technical Client/Server Solutions...Featuring PL/I

by Harry Stewart, IBM PL/I Technical Marketing
hstewart@vnet.ibm.com

IBM sponsored a technical conference and exposition at the San Jose Convention Center October 9 through October 13, 1995. On the agenda were more than 80 technical sessions, including a special CIO conference. Over 200 vendors participated in the exposition.

IBM TechCon was designed specifically to demonstrate how to integrate legacy systems with client/server solutions in a heterogeneous environment. The conference featured in-depth coverage of technical subjects including application development, data management, interoperability, transaction processing, systems management, and workgroup solutions (including, of course, Lotus Notes). There were eight conference tracks designed to serve the divergent needs of CIOs, data processing professionals, and end users.

I presented *The PL/I Family Overview* which included what we have in PL/I today and IBM's strategic direction for language products. The session included an overview of how the PL/I family of products provides access to DB2, VSAM, IMS, and CICS for application development on mainframes and workstations. I discussed the PL/I suite of products that addresses requirements for rapid application development, maintenance, reuse, downsizing, and tools support and indicated that all of these products are available for purchase today. I explained how new functions and tools provide flexibility and increase programmer productivity and stability while protecting the PL/I investment by offering an opportunity to develop a better competitive advantage for the customer.

Several key PL/I developers attended the exposition to demonstrate our workstation products. If you stopped by the booth, thanks for coming by. If not, see you at the next exposition presented in your local area.

Don't Forget!!

If you know someone who uses PL/I and might be interested in receiving this newsletter, tell them about us! Have them send their name, address, phone and fax numbers, and electronic mail address to one of the following addresses:

PL/I Newsletter
IBM - Santa Teresa Lab
555 Bailey Avenue, B3T/D284
P.O. Box 49023
San Jose, CA 95161-9023
U.S.A.

Fax (408) 463-4820
teampli@vnet.ibm.com
USIB5RLG at IBMMAIL

A Tip from the Trenches...

by Don Yazurlo, ISSC Public Sector Services
yazurlo@vnet.ibm.com

...Or, get a good night's sleep without letting table boundaries wake you up

SCENARIO: It's 3:00 a.m. "Hi, Jack. This is Paul from the data center. Job AA6040 just stopped with the following error message:

```
Program AA60 has stopped, the internal territory table
is not large enough to handle the current territory file.
Please contact development."
```

How many times have problems with hard coded table boundaries provoked a phone call in the middle of the night? How many times have developers had to change a program because an external file that is being read into an internal table is increasing in size?

There is more than one way to solve this problem, but the following routine uses controlled storage as the solution.

```
/**************************************************************/
/* The following routine will build an internal table without*/
/* hard coding the boundaries.  This is accomplished by      */
/* using PL/I controlled storage, allocate and free          */
/* instructions.                                              */
/**************************************************************/
dyntab1: proc options (main) reorder;
dcl inaad file record input;
dcl sysprint file stream output;
dcl 1 aaar based (p),
      3 aaoffno        char (3),
      3 aadporg        char (1),
      3 aamka          char (2),
      3 aadiv          char (2),
      3 left_over      char (72);
dcl time               builtin;
dcl p                  pointer;
dcl i                  fixed bin (15) init (0);
dcl rec_cnt            fixed bin (15) init (0);
dcl d5_eof             bit (1) aligned init ('0'b);
/**************************************************************/
/* The following controlled storage area will be allocated   */
/* for each record read.                                      */
/**************************************************************/
dcl 1 aa_temp          controlled,
      3 b              char (3),
      3 org            char (1),
      3 area           char (2),
      3 div            char (2);
/**************************************************************/
/* The following table is allocated once and is based on the  */
/* number of records read.                                     */
/**************************************************************/
dcl 1 aa_temp_table (rec_cnt) controlled,
      3 b              char (3),
      3 org            char (1),
      3 area           char (2),
      3 div            char (2);
on endfile (inaad)
  begin;
    d5_eof = '1'b;
  end;
```

```
/****************************************************************/
/* The following routine utilizes PL/I controlled storage to    */
/* build an internal table from a flat file.                    */
/*                                                              */
/* For each record read, allocate a storage area to hold the    */
/* fields that will make up the internal table.  Keep a count   */
/* of the number of records read, this will be used to allocate */
/* the table. Each of these allocates is analogous to putting a */
/* entry on a push down stack.  One important note, although    */
/* this appears to be a lifo stack, controlled storage is       */
/* allocated in the non-lifo section of the isa.                */
/****************************************************************/
put skip list ('dynamic array processing start time  ', time);
read file (inaad) set (p);
do while (d5_eof = '0'b);
  rec_cnt = rec_cnt + 1;
  allocate aa_temp;
  aa_temp.b          = aaoffno;
  aa_temp.org        = aadporg;
  aa_temp.area       = aamka;
  aa_temp.div        = aadiv;
  read file (inaad) set (p);
end;
/****************************************************************/
/* Allocate the table and build it from the last occurrence    */
/* to the first.  This is important because the free           */
/* instruction will release the last storage area acquired     */
/* by the allocate instruction when a specific allocation      */
/* is not specified.                                           */
/*                                                             */
/* In other words, the free instruction will take items off    */
/* the top of the referenced push down stack.                  */
/****************************************************************/
  allocate aa_temp_table (rec_cnt);
  do i = rec_cnt to 1 by -1;
    aa_temp_table.b(i)     = aa_temp.b;
    aa_temp_table.org(i)   = aa_temp.org;
    aa_temp_table.area(i)  = aa_temp.area;
    aa_temp_table.div(i)   = aa_temp.div;
    free aa_temp;
  end;
  put skip list ('dynamic array processing end time  ', time);
  put skip list ('records read  ',rec_cnt);
end dyntab1;
```

The following timings were taken on a 75 Mhz Pentium processor with 32 M of memory which was running OS/2 WARP connect. A file containing 5000 records was used.

For a program with a fixed table boundary of 5000 occurrences, these were the timings:

Start time	16:24:02:090
End time	16:24:02:500
Elapsed	00:00:00:410

For a program using controlled storage to allocate the table, these were the timings:

Start time	16:24:04:940
End time	16:24:05:500
Elapsed	00:00:00:560
Delta of	00:00:00:150

Questions and Answers

Question	Answer
Can I compile PL/I application programs as reentrant?	With option REENTRANT in the PROCEDURE statement, PL/I generates code that is reentrant. However, you must make sure that your PL/I code will not store anything into STATIC storage. This will violate the reentrant rule and goes undetected (by PL/I). Also, a PL/I program that calls or is called by a FORTRAN or COBOL program is not reenterable.
Does PL/I on AIX interface with DB2 for AIX?	Yes! PL/I provides an integrated SQL preprocessor so you can embed SQL statements (dynamic and static) in your PL/I programs on either the AIX or OS/2 platforms.
Can anybody tell me if V1.5 programs can run using the V2.3 runtime library?	Yes, V1.5 load modules will run under V2.3

Does anyone have PL/I code fragment that returns the day of the week for a given date? For example, given a date of 1995/11/07 would return 'Tuesday'.

The solution given here would be useful if using mainframe PL/I, but PL/I for OS/2 and PL/I Set for AIX support a built-in function that provides the same outcome.

```
/*****************************************/
/*DayOfWeek routine                      */
/*Converts Gregorian date to day of week */
/*****************************************/
/* -->>>>No error checking is performed! */
/* 19891101 Walter Pachl                 */
/* REXXified algorithm published in CACM */
/* (Fliegel & van Flandern, CACM Vol. 11 */
/* No.10 October 1968)                   */
/* Valid from 10/15/1582 to 2/28/3200    */
/* Day of week returns 0 to 6 for Monday */
/* to Sunday.                            */
/*****************************************/

DayOfWeek: proc(yyyy_mm_dd)
           returns(fixed bin(15));

dcl yyy_mm_dd char(10) nonasgn;

dcl 1 greg def yyyy_mm_dd,
        3 y pic '9999',
        3 s1 char(1),
        3 m pic '99',
        3 s2 char(1),
        3 d pic '99';

dcl (ma,ya,xx,yy,zz,day) fixed bin(31);

if bin(m) < 3 then ma = -1; else ma = 0;
ya = bin(y) + 4800 + ma;
xx = floor((1461 * ya) /4);
yy = floor((367 * (bin(m)-2-(ma*12))) /12);
zz = 3 * floor(floor ((ya+100) /100) /4);
day = bin(d) - 32075 + xx + yy - zz;

return (mod(day,7));

end DayOfWeek;
```

Oops!

In the Questions and Answers section of the last issue of THE PL/I CONNECTION (Issue 4, September 1995), we referred to an HTML-format document, *Migration to Version 2 of the PL/I Compiler.* Unfortunately, we failed to notice that the URL for accessing it is behind a firewall, and cannot be reached from outside the IBM internal network. However, due to the interest in this document, the author has been kind enough to create a flat text version of the article which he can send by electronic mail. If you would like a copy, please contact *MartinF_James@uk.ibm.com.*

Please accept our apologies for any confusion or inconvenience this oversight may have caused.