IBM Library Server Print Preview

DOCNUM = GC26-9324-01
DATETIME = 05/27/98 01:56:56
BLDVERS = 1.2
TITLE = PL/I Millennium Language Extensions Guide
AUTHOR =
COPYR = © Copyright IBM Corp. 1998
PATH = /home/webapps/epubs/htdocs/book

COVER Book Cover

PL/I

Millennium Language Extensions Guide

Document Number GC26-9324-01

Program Number
5648-MLX
5686-MLX

ABSTRACT Abstract

The *MLE Guide* contains conceptual information about using millennium language extensions that are now available to help you locate and correct potential date-related fields.

NOTICES Notices

Note!

Before using this information and the product it supports, be sure to read the general information under "Notices" in topic FRONT_1.

EDITION Edition Notice

| **Second Edition (June 1998)**

This edition applies to:

- VisualAge PL/I Millennium Language Extensions for MVS & VM, Version 1 Release 1 (Program number 5648-MLX)

| ° VisualAge PL/I Millennium Language Extensions for VSE/ESA, Version
| 1 Release 1 (Program number 5686-MLX)

and to any subsequent releases until otherwise indicated in new editions or technical newsletters. Make sure you are using the correct edition for the level of the product.

Order publications through your IBM representative or the IBM branch office serving your locality. Publications are not stocked at the address below.

A form for readers' comments is provided at the back of this publication. If the form has been removed, address your comments to:

IBM Corporation, W92/H3
P.O. Box 49023
San Jose, CA 95161-9023
U.S.A.

When you send information to IBM, you grant IBM a nonexclusive right to use or distribute the information in any way it believes appropriate without incurring any obligation to you.

CONTENTS Table of Contents

FRONT_1 Notices

References in this publication to IBM products, programs, or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service can be used. Any functionally equivalent product, program, or service that does not infringe any of the intellectual property rights of IBM might be used instead of the IBM product, program, or service. The evaluation and verification of operation in conjunction with other products, except those expressly designated by IBM, are the responsibility of the user.

IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
500 Columbus Avenue
Thornwood, NY 10594
U.S.A.

Subtopics:

FRONT_1.1 Programming Interface Information

This book primarily documents intended Programming Interfaces that allow the customer to write programs to obtain services of:

- IBM PL/I for MVS & VM when used in conjunction with VisualAge PL/I Millennium Language Extensions for MVS & VM
- IBM PL/I for VSE/ESA when used in conjunction with VisualAge PL/I Millennium Language Extensions for VSE/ESA.

This book also documents information that is NOT intended to be used as Programming Interfaces of VisualAge PL/I Millennium Language Extensions for MVS & VM and VisualAge PL/I Millennium Language Extensions for VSE/ESA. This information is identified where it occurs, either by an introductory statement to a chapter or section or by the following marking:

---------- NOT Programming Interface information ----------

|---------- End of NOT Programming Interface information ----------|

FRONT_1.2 Trademarks

The following terms are trademarks of the IBM Corporation in the United States or other countries or both:

CICS	IMS
DB2	MVS
DFSORT	VisualAge
IBM	VSE/ESA

PREFACE About This Book

This book provides information on the millennium language extensions that have been incorporated into IBM PL/I to assist with Year 2000 processing. PL/I millennium language extensions are provided by the following combinations of products:

- IBM PL/I for MVS & VM and VisualAge PL/I Millennium Language Extensions for MVS & VM
- IBM PL/I for VSE/ESA and VisualAge PL/I Millennium Language Extensions for VSE/ESA

The Year 2000 problem has been documented in many places and it is not the intent of this book to describe the problem in detail. It is sufficient to say that the potential for logic errors can occur at the turn of the century when processing with two-digit year dates in your applications. Files and data bases that hold date fields in this form, and programs that act on those date fields, generally "assume" that the date occurs within the 1900-1999 range, so that "98" for example, really means the year 1998. When the year changes from 1999 to 2000, this assumption is no longer valid.

Again, this book is not designed to provide a comprehensive solution to this problem. Generally, plans to address this situation include a *find* stage to identify applications that might be affected by the Year 2000 problem. Then, the *fix* stage provides steps to change and test those applications.

Assuming that affected programs have been identified, PL/I millennium language extensions can be placed in the *fix* stage of the plan. They provide one method of assisting with the Year 2000 challenge and should be considered, along with other methods, when formulating a solution.

| CHANGES Summary of Changes

| This section lists the major changes that have been made to this book.

Subtopics:

| CHANGES.1 Second Edition, June 1998

| Information has been added about using PL/I millennium language extensions
| with the IBM PL/I for VSE/ESA compiler in conjunction with VisualAge PL/I
| Millennium Language Extensions for VSE/ESA.

1.0 Chapter 1. Finding Solutions to the Year 2000 Challenge

What is this year 2000 challenge? Currently, a majority of programs and databases use two digits rather than four to represent the year when using date values. Program logic assumes the first two digits of the date are "19" and so only the last two digits are explicitly represented.

Until the turn of the century becomes a factor, this compact date format works well. With the year 2000 swiftly approaching, however, applications which once produced valid results can potentially introduce computational errors. For example, two-digit years which fall after the year 2000 such as 01 and 02 are interpreted as numerically smaller than, and therefore logically preceding, two-digit years which occur prior to the year 2000 such as 97 and 98.

Subtopics:

1.1 Choosing the Right Approach

Before you can fix your two-digit date data, you must select which approach to use for the given data. The solution you choose depends on the current structure of your applications and how much time you have, in addition to other considerations. There are two methods generally offered to handle two-digit dates:

° Expand two-digit year fields to four digits
° Infer the correct century through a process called *windowing*.

The expansion solution involves explicitly expanding two-digit year dates

to contain four digits assuring reliable date processing, but is labor intensive. Windowing can often be achieved with few changes to your program.

Expansion, then, is a *process* that you must perform, while windowing is a *method* of viewing dates. The intent of this book is to explain PL/I's implementation of windowing through the introduction of millennium language extensions.

1.2 Defining MLE

| MLE is short for "millennium language extensions," IBM's patent-pending
| technology that provides support for automated date windowing in the
| following PL/I compilers:

| ° PL/I for MVS & VM when used in conjunction with VisualAge PL/I
| Millennium Language Extensions for MVS & VM

| ° PL/I VSE when used in conjunction with VisualAge PL/I Millennium
| Language Extensions for VSE/ESA

In short, it is a compiler-assisted solution to help you tackle the Year 2000 challenge.

Subtopics:

1.2.1 What MLE Does for You

Using MLE, you can change the data declarations in your application to indicate which items represent dates to be windowed and how you want the window to be defined. The compiler then implements the windowing changes, in many cases without you having to code the associated logic changes. MLE has the potential to reduce programming time and effort to make your applications Year 2000 ready and also simplify maintenance beyond the turn of the century.

1.2.2 What MLE Does Not Do for You

It is important to recognize that MLE is not designed to be a single solution to the Year 2000 challenge. Though it can relieve you of logic changes should you decide on a windowing solution, it does not take the place of careful assessment, planning, analysis, and testing activities.

A windowing solution might not be an acceptable approach for all applications. There could be cases where the limitations of MLE (size of the window, for example) require that you make manual logic changes. Hopefully, these situations will not occur frequently. The following section should help you identify when windowing is the right solution.

1.3 Implementing a Windowing Solution

Date windowing is a method used to determine in which century a two-digit year falls. With PL/I MLE, a century window is a period of exactly 100 years which you define to the compiler. The compiler, in turn, applies this window to two-digit year date values in your application.

| For example, for a century window covering from 1930 to 2029, the PL/I
| compiler (either PL/I for MVS & VM or PL/I VSE with the appropriate PTFs
| applied and the corresponding PL/I MLE product installed) evaluates dates
in the following manner:

```
Years between 30 and 99 are interpreted as 1930-1999
Years between 00 and 29 are interpreted as 2000-2029.
```

Subtopics:

1.3.1 When to Use Windowing

Date windowing is an effective solution to the year 2000 problem for an application that fits the following profile:

- The application files contain date fields with two-digit years
- The application programs perform simple operations on dates, such as comparing two dates to determine which is earlier
- The life expectancy of the application is such that its date fields go beyond 1999.

MLE provides an effective tool to implement date windowing for applications such as this. However, there are other factors to consider before commencing an implementation project. This section helps you identify those factors that are relevant in the MLE context.

Date range
If the application contains dates that span more than 100 years, it is not eligible for windowing. For example, a publishing house could not realistically use date windowing because the publication dates of its books could go back more than 100 years. But a manufacturing company could use date windowing if the dates of orders, deliveries, and invoices all fell within a limited range.

Century window
You must be able to define a century window that is appropriate for all variables in the application. For example, a children's hospital can record dates of birth using a window that starts 20 years ago, and this would be adequate for admissions and discharges, but may not be for historical records.

Other languages
If the same variable is used by other programs written in other languages such as Assembler, it is probably easier with these other programs to implement date expansion rather than date windowing. If this is the case, you should also implement date expansion in your PL/I programs.

Other uses of the date field
Both the PL/I millennium language extensions and other products impose limits on where you can use windowed variables. For example, if a variable is used in a context where its binary value is important, it probably cannot be windowed, this includes:

- A key on a VSAM file
- A search field in a data base system such as IMS or DB2
- A key field in a CICS command.

1.3.2 How to Apply Windowing

This section describes how you can use MLE to implement a date windowing solution, and suggests some other methods that you could use instead of, or in addition to, MLE.

Subtopics:

1.3.2.1 The MLE Method

To implement automatic date windowing using PL/I MLE, you need to complete the following tasks:

1. Add the DATE attribute to those variables that are to be windowed (see "DATE Attribute" in topic 2.1.1).

2. If necessary, use the DAYS and DAYSTODATE built-in functions for date conversion (see "Using Built-in Functions with MLE" in topic 2.3).

3. Use the RESPECT compile-time option to enable MLE (see "RESPECT Compile-time Option" in topic 2.1.2).

4. Use the WINDOW compile-time option to set the century window (see "WINDOW Compile-time Option" in topic 2.1.3).

5. If necessary, use the RULES(LAXCOMMENT) compile-time option to have the compiler honor DATE attributes enclosed between special comment delimiters (see "RULES Compile-time Option" in topic 2.1.4).

1.3.2.2 Other Methods

With MLE, it is often possible to implement a solution in which the windowing process is fully automatic; that is, you simply identify the fields that contain windowed dates, and you do not need any extra program logic to interpret the window. However, you can also implement date windowing using other methods, some of which might require additional program logic.

In some cases, you might need to use one or more of these other methods in addition to MLE in order to achieve your goal. For example, if a date field is not in a format supported by MLE, you need to use an alternative method of windowing for that field.

Some other methods of implementing date windowing are:

PL/I coding
: You can insert IF statements around the references to date fields in your program, to determine how to apply a century component. For example, the following code implements a century window of

```
1940-2039:

    if YY_1 < 40 then
       CC_1 = 20;
    else
       CC_1 = 19;
```

Converting dates

Use the DAYS and DAYSTODATE built-in functions to help you manipulate and convert dates. See "Using Built-in Functions with MLE" in topic 2.3 for more details.

Though not its primary purpose, you can use MLE and windowing as a first step in expanding two-digit to four-digit years.

1.3.2.3 Subsystems with No Windowing

The Millennium Language Extensions permit the specification of windowed dates for sorting and merging, in conjunction with sort products such as DFSORT. However, the following subsystems do not support windowed dates:

- DB/2
- IMS
- CICS
- VSAM

This can cause a variety of problems, including errors from mismatched ordering or compilation failure.

1.3.3 Preliminary Testing with MLE

A pilot project is always something to consider for your year 2000 work. You should plan to do some preliminary testing with MLE to determine its usefulness. If you are interested in taking the windowing approach, you must first migrate to a year 2000 ready compiler which supports MLE.

In your analysis, if you determine that a century window approach and MLE are viable solutions, these steps can help you take things in order:

- Choose a fixed-window or sliding-window solution
- Choose a century-window start year or offset for each program
- Modify appropriate data declarations
- Compile the program with the appropriate options.

To test an application, use the following steps:

- Run regression tests with 1900 as the start year (the default); this should give the same results as those prior to the changes.

- Run date simulator tests with your selected start year.

- After each executable is tested, it can be moved into production with the year 2000 support enabled.

1.4 Getting Started with MLE

To get some practical application of the items discussed in this chapter, consider this short example which presents a problem and uses MLE in the solution.

Subtopics:

1.4.1 A Simple Date Problem

Our example consists of a program which compares the date that a video tape was returned with the date it was due back to see whether to impose a fine. The two dates are stored in the file as 6-digit Gregorian dates; that is, YYMMDD.

```
dcl
 1 loan_record,
   2 member_number   pic '99999999',
   2 tape_id         pic '99999999',
   2 date_due_back   pic '999999',
   2 date_returned   pic '999999';
      .
      .
      .
if date_returned >date_due_back then
   call fine_him;
```

If the tape was due back on 14 September 1998, the contents of date_due_back are 980914. If it was returned on 12 September, the contents of date_returned are 980912. Therefore, no fine is imposed, because date_returned is less than date_due_back

If we go forward in time, we can see how this program behaves in January 2000. If the tape is due back on 2 January 2000, but is returned on 31 December 1999, the contents of the fields are:

```
date_due_back is 000102
date_returned is 991231
```

In this case, date_returned is much larger than date_due_back, and the program imposes a hefty fine for being 100 years late.

1.4.2 Solving the Problem

If the program recognized that the year 00 was in fact 2000, not 1900, the result would be different. In the new scenario, the If ... Then ... statement would behave properly, viewing 991231 as 19991231 and 000102 as 20000102, resulting in correct logic path and no fine for the tape that was returned early, not late.

PL/I MLE gives the flexibility you need to do this. Again, you have to do a few things to make it work.

1. Tell the compiler which fields to treat as date fields with two-digit years, so it knows when to apply the century window and when not to. You do this by changing the PL/I source program, adding a DATE attribute to the date fields that you want handled in this way. So the record layout now looks like this:

```
dcl
 1 loan_record,
   2 member_number   pic '99999999',
```

```
        2 tape_id         pic '99999999',
        2 date_due_back   pic '999999' date('YYMMDD'),
        2 date_returned   pic '999999' date('YYMMDD');
```

2. Decide on a *century window*, and tell the PL/I compiler about it. Take into account how far back your historic data goes and how far into the future your date data is. If you decide to use 1950-2049 as your window, then PL/I evaluates dates like this:

   Years between 50 and 99 are really 1950-1999.
   Years between 00 and 49 are really 2000-2049.

   Use the following compile-time options to tell the compiler about the century window:

   RESPECT
   Tells the compiler to enable the windowing process.

   You specify this as either RESPECT(DATE), to have the compiler honor any specification of the DATE attribute and to apply the DATE attribute to the result of DATE built-in, or RESPECT() to have the compiler ignore any specification of the DATE attribute and to not apply the DATE attribute to the result of DATE built-in.

   WINDOW
   Defines whether a fixed or sliding century window is used and the beginning date for that window.

   You specify this as WINDOW(w), where w is either an unsigned integer between 1582 and 9999 for a fixed window, or a negative integer between -1 and -99 for a sliding century window set to w years before the current system date. The value for w can also be zero for the current year.

3. Test the program.

To summarize, then, you can introduce a windowed date concept into a PL/I program by deciding on a century window, changing some data items, and recompiling the program. Chapter 2, "Using PL/I MLE in Your Applications" in topic 2.0 contains more details on MLE syntax and how to use it.

2.0 Chapter 2. Using PL/I MLE in Your Applications

With the introduction of MLE, PL/I provides support for a number of new language features. The purpose of this chapter is for you to become familiar with the new attribute, compile-time options, date patterns, and built-in functions. As you follow the sequence of the chapter, you should have an idea about how to apply these to your existing applications.

Subtopics:

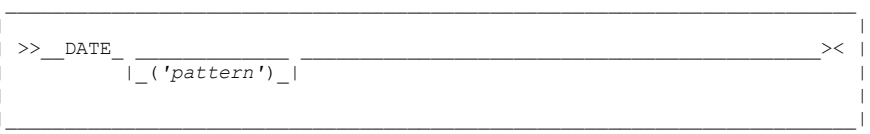
2.1 Applying Attributes and Options

The language features introduced in these sections are not part of your PL/I compiler documentation. When the next release of your compiler becomes available, descriptions of these features will be integrated into the Language Reference and Programming Guide.

Subtopics:

2.1.1 DATE Attribute

Implicit date comparisons and conversions are made by the compiler if the two operands have the DATE attribute. The DATE attribute specifies that a variable, argument, or returned value holds a date with a specified pattern. PL/I MLE supports a number of date patterns as described in "Understanding Date Patterns" in topic 2.2.



pattern
: One of the supported date patterns. If you do not specify a pattern, YYMMDD is the default. For a list of supported date patterns, see Table 1 in topic 2.2.

The DATE attribute is valid only with variables having one of the following sets of attributes:

- CHAR(n) nonVARYING
- PIC'(n)9' REAL
- FIXED DEC(n,0) REAL

The length or precision, *n*, must be a constant equal to the length of the date pattern or default pattern.

When the RESPECT compile-time option (discussed later in this chapter) has been specified, the DATE built-in function returns a value that has the attribute DATE('YYMMDD'). This allows DATE() to be assigned to a variable with the attribute DATE('YYMMDD') without an error message being generated. If DATE() is assigned to a variable not having the DATE attribute, however, an error message is generated.

Here are a few examples using the DATE attribute:

```
dcl gregorian_Date char(6) date;

dcl julian_Date    pic'(5)9' date ('YYDDD');

dcl year           fixed dec(2) date('YY');
```

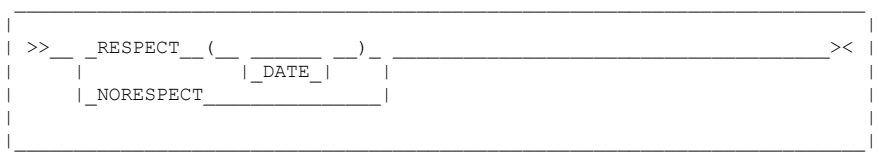
The DATE attribute is useful even if you have no year 2000 problems in

your applications. You can use it to manipulate differing dates as shown in these examples:

```
dcl gregorian_Date   char(8) date ('YYYYMMDD');

dcl julian_Date      pic'(7)9' date ('YYYYDDD');

if julian_Date > gregorian_Date then ...
```

2.1.2 RESPECT Compile-time Option

Use the RESPECT option to specify which attributes the compiler should recognize. Currently, DATE is the only selection possible for this compile-time option.



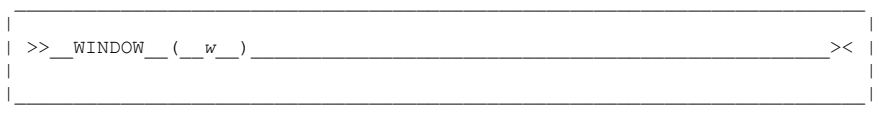
The default is RESPECT() and causes the compiler to ignore any specification of the DATE attribute. Therefore, the DATE attribute is not applied to the result of DATE built-in. NORESPECT is a synonym for RESPECT()

Specifying RESPECT(DATE), on the other hand, causes the compiler to honor any specification of the DATE attribute and to apply the DATE attribute to the result of DATE built-in.

TSO/MVS users, note that RESPECT() is not accepted when compiling with the PLI command on TSO/MVS.

2.1.3 WINDOW Compile-time Option

By default, all dates with two-digit years are viewed as falling in a window starting with 1950 and ending in 2049. You can use the WINDOW option to change the value for your century window.



As previously mentioned, the default for this option is WINDOW(1950). You can specify the value for *w* as one of the following:

- An unsigned integer between 1582 and 9999 (inclusive) that represents the start of a fixed century window
- A negative integer between -1 and -99 (inclusive) that creates a "sliding" century window
- Zero, indicating the value for w is the current year.

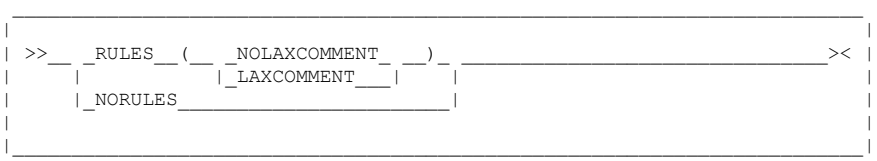
To create a fixed window, you could specify WINDOW(1900) and all two-digit years would be assumed to occur in the 20th century.

If the current year were 1998, and you wanted to create a sliding window, you could specify WINDOW(-5). The resulting century window would span the years 1993 through 2092, inclusive. When the year changes to 1999, the window would also move forward by one year.

If you set a value for the century window using the WINDOW compile-time option, that value is used for the window argument in the built-in functions which allow it, unless otherwise specified in that built-in. See "Using Built-in Functions with MLE" in topic 2.3 for more details.

2.1.4 RULES Compile-time Option

In general, the RULES option allows or disallows certain language capabilities and allows you to choose semantics when alternatives are available. Currently, LAXCOMMENT is the only selection available for this option.



The default is RULES(NOLAXCOMMENT). LAXCOM and NOLAXCOM are acceptable abbreviations for the suboptions.

If you specify RULES(LAXCOMMENT), the compiler ignores the special characters /*/; therefore, whatever comes between the sets of characters is interpreted as part of the syntax instead of as a comment. If you specify RULES(NOLAXCOMMENT), the compiler treats /*/ as the start of a comment which continues until a closing */ is found.

If you happen to have workstation code that you are porting to the mainframe and uses /*/ around the DATE attribute, you need to use the RULES(LAXCOMMENT) option so that the compiler honors the attribute.

2.2 Understanding Date Patterns

PL/I MLE supports a series of date patterns as shown in the following table.

Table 1. Date patterns supported by PL/I MLE

	4-digit year	**Example**	**2-digit year**	**Example**
Year first	YYYY	1999	YY	99
	YYYYMM	199912	YYMM	9912
	YYYYMMDD	19991225	YYMMDD	991225
	YYYYMMM	1999DEC	YYMMM	99DEC
	YYYYMMMDD	1999DEC25	YYMMMDD	99DEC25
	YYYYMmm	1999Dec	YYMmm	99Dec
	YYYYMmmDD	1999Dec25	YYMmmDD	99Dec25
	YYYYDDD	1999359	YYDDD	99359
Month first	MMYYYY	121999	MMYY	1299
	MMDDYYYY	12251999	MMDDYY	122599
	MMMYYYY	DEC1999	MMMYY	DEC99
	MMMDDYYYY	DEC251999	MMMDDYY	DEC2599
	MmmYYYY	Dec1999	MmmYY	Dec99
	MmmDDYYYY	Dec251999	MmmDDYY	Dec2599
Day first	DDMMYYYY	25121999	DDMMYY	251299
	DDMMMYYYY	25DEC1999	DDMMMYY	25DEC99

	DDMmmYYYY	25Dec1999	DDMmmYY	25Dec99
	DDDYYYY	3591999	DDDYY	35999

When the day or month is omitted from one of these patterns, the compiler assumes it has a value of 1.

Subtopics:

- 2.2.1 Patterns and Windowing

2.2.1 Patterns and Windowing

To define how a date with a two-digit year (YY) is interpreted, a century window is defined using the WINDOW compile-time option. As described previously, the century window defines the beginning of a 100-year span to which the two-digit year applies.

Without the help of the PL/I millennium language extensions, you would have to implement something like the following logic which converts y2 from a two-digit year to a four-digit year with a window (w).

```
dcl y4 pic'9999';
dcl cc pic'99';

cc = w/100;

if y2 < mod(w,100) then
  y4 = (100 * cc) + 100 + y2;
else
  y4 = (100 * cc) + y2;
```

Using this example, if you were to specify WINDOW(1900), 19 would be interpreted as the year 1919. If you were to specify WINDOW(1950), however, 19 would be interpreted as the year 2019.

Conversely, this logic calculates the two-digit year (y2) when converting from a four-digit year.

```
dcl y4 pic'9999';

if y4 < w | y4 >= w + 100 then
    signal error;

y2 = mod(y4,100);
```

2.3 Using Built-in Functions with MLE

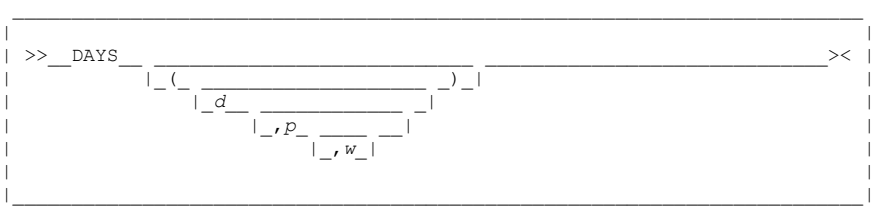
The date patterns for PL/I MLE are supported by the DAYS and DAYSTODATE built-in functions. These built-ins both accept the optional argument (w) that specifies a window to be used in handling two-digit year patterns. If you specify w as part of DAYS or DAYSTODATE, the value you enter overrides the value as defined by the WINDOW compile-time option.

Subtopics:

- 2.3.1 DAYS
- 2.3.2 DAYSTODATE

2.3.1 DAYS

DAYS returns a FIXED BINARY(31,0) value which is the number of days (in Lilian format) corresponding to the date *d*.



d String expression representing a date. If omitted, it is assumed to be the value returned by DATETIME().

The value for *d* should have character type. If not, *d* is converted to character.

p One of the supported date patterns shown in Table 1 in topic 2.2. If omitted, the compiler assumes that *p* is the default pattern returned by the DATETIME built-in function (YYYYMMDDHHMISS999).

p should have character type. If not, it is converted to character.

w An integer expression that defines a century window to be used to handle any two-digit year formats.

- If the value is positive, such as 1950, it is treated as a year.
- If negative or zero, the value specifies an offset to be subtracted from the current, system-supplied year.
- If omitted, *w* defaults to the value specified in the WINDOW compile-time option.

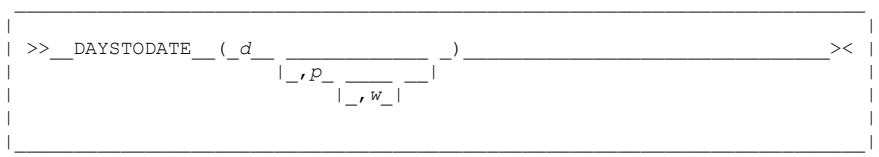
The following example shows uses of both the DAYS and DAYSTODATE built-in functions:

```
dcl date_format char(8) static init('MMDDYYYY');
dcl todays_date char(8);
dcl sep2_1993 char(8);
dcl days_of_july4_1993 fixed bin(31);
dcl msg char(100) varying;
dcl date_due char(8);


todays_date = daystodate(days(),date_format);


days_of_july4_1993 = days('07041993','MMDDYYYY');
sep2_1993 = daystodate(days_of_july4_1993 + 60, Date_format);
             /*  09021993    */


date_due = daystodate(days() + 60, date_format);
        /* assuming today is July 4, 1993, this would be Sept. 2, 1993


msg = 'Please pay amount due on or before ' ||
```

```
            substr(date_due, 1, 2) || '/' ||
            substr(date_due, 3,2)  || '/' ||
            substr(date_due, 5);
```

2.3.2 DAYSTODATE

DAYSTODATE returns a nonvarying character string containing the date in the form *p* that corresponds to *d* days (in Lilian format).



d The number of days (in Lilian format).

d must have a computational type and is converted to FIXED BINARY(31,0) if necessary.

p One of the supported date patterns shown in Table 1 in topic 2.2. If omitted, the compiler assumes that *p* is the default pattern returned by the DATETIME built-in function (YYYYMMDDHHMISS999).

p should have character type. If not, it is converted to character.

w An integer expression that defines a century window to be used to handle any two-digit year formats.

- If the value is positive, such as 1950, it is treated as a year.
- If negative or zero, the value specifies an offset to be subtracted from the current, system-supplied year.
- If omitted, *w* defaults to the value specified in the WINDOW compile-time option.

2.4 Performing Date Calculations and Comparisons

Once you understand what the PL/I millennium language features are and you have made the appropriate syntax changes, you can use MLE to perform calculations and comparisons in your applications.

Subtopics:

2.4.1 Explicit Date Calculations

You can use the DAYS and DAYSTODATE built-in functions to make date

comparisons and calculations manually.

Subtopics:

2.4.1.1 Comparing Dates

To compare two dates d1 and d2 which have the date pattern YYMMDD, you can use the following code:

```
DAYS (d1, 'YYMMDD', w) < DAYS(d2, 'YYMMDD', w)
```

2.4.1.2 Converting Dates

You can convert between a two-digit date (d1) with the pattern YYMMDD and a four-digit date (d2) with the pattern YYYYMMDD using assignments:

```
d2 = DAYSTODATE(DAYS(d1,'YYMMDD',w), 'YYYYMMDD');
d1 = DAYSTODATE(DAYS(d2,'YYYYMMDD'), 'YYMMDD' ,w);
```

2.4.1.3 Subtracting Dates

To subtract 2 two-digit years, y1 and y2, you need to calculate the imposing difference:

```
DAYSTODATE(DAYS(y1,'YY',w), 'YYYY') -
        DAYSTODATE(DAYS(y2,'YY',w),'YYYY')
```

2.4.2 Implicit Date Calculations

You can use MLE to take advantage of implicit date comparisons and conversions if you first complete the following steps:

- Give the two operands the DATE attribute
- Specify the RESPECT compile-time option

2.4.3 Implicit Date Comparisons

The DATE attribute causes implicit *commoning* when two variables declared with the DATE attribute are compared. Comparisons where only one variable has the DATE attribute are flagged, and the other comparand is generally treated as if it had the same DATE attribute, although some exceptions apply which are discussed later.

Implicit commoning means that the compiler generates code to convert the dates to a common, comparable representation. This process converts 2-digit years using the *window* you specify in the WINDOW compile-time option.

In the following code fragment, if the DATE attribute is honored, then the comparison in the second display statement is 'windowed'. This means that if the window started at 1900, the comparison would return false. However, if the window started at 1950, the comparison would return true.

```
dcl a   pic'(6)9' date;
dcl b   pic'(6)9' def(a);
dcl c   pic'(6)9' date;
dcl d   pic'(6)9' def(c);


b = '670101';
d = '010101';


display( b || ' < ' || d || ' ?' );
display( a < c );
```

Date comparisons can also occur in the following places:

- IF and SELECT statements
- WHILE or UNTIL clauses
- Implicit comparisons caused by a TO clause.

Subtopics:

2.4.3.1 Comparing Dates with Like Patterns

The compiler does not generate any special code to compare dates with identical patterns under the following conditions:

- The comparison operator of = or ¬= is used
- The pattern is equal to YYYY, YYYYMM, YYYYDDD, or YYYYMMDD.

2.4.3.2 Comparing Dates with Differing Patterns

For comparisons involving dates with unlike patterns, the compiler generates code to convert the dates to a common comparable representation. Once the conversion has taken place, the compiler compares the two values.

2.4.3.3 Comparisons Involving the DATE Attribute and a Literal

If you are making comparisons in which one comparand has the DATE attribute and the other is a literal, the compiler issues a W-level message. Further compiler action depends on the value of the literal as follows:

- If the literal appears to be a valid date, it is treated as if it had the same date pattern and window as the comparand with the DATE attribute.

- If the literal does not appear to be a valid date, the DATE attribute is ignored on the other comparand.

```
dcl start_date char(6) date;
if start_date >= '' then /* no windowing */
...
if start_date >= '851003' then /* windowed */
...
```

2.4.3.4 Comparisons Involving the DATE Attribute and a Non-Literal

In comparisons where one comparand has the DATE attribute and the other is not a date and not a literal, the compiler issues an E-level message. The non-date value is treated as if it had the same date pattern as the other comparand and as if it had the same window.

```
dcl start_date char(6) date;
dcl non_date char (6);

if start_date >= non_date then /* windowed */
...
```

2.4.4 Implicit DATE Assignments

The DATE attribute can also cause implicit conversions to occur in assignments of two variables declared with date patterns.

° If the source and target have the same DATE and data attributes, then the assignment proceeds as if neither had the DATE attribute.

° If the source and target have differing DATE attributes, then the compiler generates code to convert the source date before making the assignment.

° In assignments where the source has the DATE attribute but the target does not, the compiler issues an E-level message and ignores the DATE attribute.

° In assignments where the target has the DATE attribute but the source does not (and the source IS NOT a literal), the compiler issues an E-level message and ignores the DATE attribute.

° In assignments where the target has the DATE attribute but the source does not (and the source IS a literal), the compiler issues a W-level message and ignores the DATE attribute.

```
dcl start_date char(6) date;
start_date = '';
...
```

° The DATE attribute is ignored in:

- The debugger
- Assignments performed in record I/O statements
- Assignments and conversions performed in stream I/O statements (such as GET DATA).

Even if you do not choose a windowing solution, you might have some code that needs to manipulate both two- and four-digit years. You can use multiple date patterns to help you in these situations:

```
dcl old_date char(6) date('YYMMDD');
dcl new_date char(8) date('YYYYMMDD');


new_date = old_date;
```

2.5 Summarizing Date Diagnostics

In PL/I, *effective* assignments occur when

- An expression is passed as an argument to an entry that has described that argument
- An expression is used in a RETURN statement.

The following uses of variables with the DATE attribute are flagged:

- Assignments (explicit or effective) which include either
  - A date to a non-date
  - A non-date to a date
- Any arithmetic operation applied to a date
- Use of a date in a BY clause (since this implies an arithmetic operation)
- Use of a date in any mathematical built-in function
- Use of a date in any arithmetic built-in function except BINARY, DECIMAL, FIXED, FLOAT, or PRECISION
- Use of a date in the built-in functions SUM, PROD, or POLY.

In all of these cases, code is produced but no windowing occurs. In effect, the DATE attribute is ignored.

2.6 Using MLE with the SQL Preprocessor

The SQL preprocessor objects to the DATE attribute. However, if you enclose the attribute between /*/ and /*/, the SQL preprocessor ignores it (as part of a comment that stretches from the first /* to the last */). In order for the compiler to honor the DATE attribute between these special characters, you must specify RULES(LAXCOMMENT), see "RULES Compile-time Option" in topic 2.1.4 for more details.

2.7 The MLE Objective

To summarize then, the primary objective of PL/I MLE is to extend the useful life of applications (as they are currently specified) into the twenty-first century. Source changes to accomplish this can hopefully be held to a minimum, preferably limited to making additions to declarations of date-related variables. In many cases, you should not be required to make any changes to the program logic.

A secondary objective is to support limited kinds of maintenance and enhancement, especially to allow dates with two-digit years to be used in conjunction with expanded year dates. This can assist you with incrementally introducing expanded dates into your applications, if desired.

3.0 Chapter 3. PL/I for MVS & VM MLE Messages

This appendix documents information that is NOT intended to be used as Programming Interfaces.

The following descriptions are for the PL/I for MVS & VM compile-time messages (see also "PL/I TSO Prompter Messages" in topic 3.2) and codes that are either changed or new as a result of VisualAge PL/I Millennium Language Extensions for MVS & VM. In the case of message IEL02301, a new restriction (299) exists as shown.

Subtopics:

3.1 Compile-time messages

IEL0230I U COMPILER ERROR OR RESTRICTION NUMBER 299 DURING PHASE 'II'.

Explanation: The program has too many statements that use variables declared with the DATE attribute. Approximately 10,000 references with the DATE attribute are allowed in program statements. For example, in assignment statements, IF statements, etc.

Programmer Response: Reduce the usage of variables declared with the DATE attribute, particularly those that are based or subscripted. Alternatively, reduce the usage of the DATE builtin or the usage of functions that return a value with the DATE attribute.

IEL0661I W 'DATE' ATTRIBUTE IGNORED IN COMPARISON OF 'DATE' OPERAND WITH NON-DATE CONSTANT.

COMPARISON OF 'DATE' OPERAND WITH NON-DATE CONSTANT IS INVALID. THE 'DATE' ATTRIBUTE HAS BEEN IGNORED.

Example:

```
DCL X CHAR(6) DATE('YYMMDD');
IF X='' THEN ...
```

Explanation: In a comparison, if one operand has the DATE attribute, the other should also. The compiler has corrected this by ignoring the DATE attribute.

IEL0662I W CONSTANT IN COMPARISON IS ASSUMED TO HAVE SAME PATTERN AS 'DATE' OPERAND.

COMPARISON OF 'DATE' OPERAND WITH CONSTANT HAS BEEN ACCEPTED. CONSTANT IS ASSUMED TO HAVE THE SAME PATTERN AS THE OPERAND WITH THE 'DATE' ATTRIBUTE.

Example:

```
DCL X CHAR(6) DATE('YYMMDD');
IF X>'971122' THEN ...
```

Explanation: In a comparison, if one operand has the DATE attribute, the other should also. The compiler has corrected this by assuming that the constant has the same DATE attribute. In the example above, the constant '971122' is assumed to have the DATE pattern 'YYMMDD', the same as X.

IEL0663I E NON-DATE OPERAND IN COMPARISON IS ASSUMED TO HAVE SAME PATTERN AS 'DATE' OPERAND.

COMPARISON OF 'DATE' OPERAND WITH NON-DATE OPERAND IS INVALID. THE NON-DATE OPERAND IS ASSUMED TO HAVE THE SAME PATTERN AS THE OPERAND WITH THE 'DATE' ATTRIBUTE.

Example:

```
DCL X CHAR(6) DATE('YYMMDD');
DCL Y CHAR(6);
IF X>Y THEN ...
```

Explanation: In a comparison, if one operand has the DATE attribute, the other should also. The compiler has corrected this by assuming that the non-DATE operand has the same DATE attribute. In the example above, Y is assumed to have the DATE pattern 'YYMMDD', the same as X.

IEL0664I E [PROLOGUE CODE.] 'DATE' ATTRIBUTE OF TARGET D IGNORED IN ASSIGNMENT.

[PROLOGUE CODE.] SOURCE IN ASSIGNMENT DOES NOT HAVE THE 'DATE' ATTRIBUTE BUT TARGET D DOES. THE 'DATE' ATTRIBUTE HAS BEEN IGNORED FOR THIS ASSIGNMENT.

Example:

```
DCL X CHAR(6) DATE('YYMMDD');
DCL Y CHAR(6);
X=Y;
```

Explanation: If the target in an assignment statement has the DATE attribute, the source should also. The compiler has corrected this by ignoring the DATE attribute in performing this assignment.

For a DO statement, this message may be issued for the initial assignment of the loop control variable or for a 'REPEAT' clause.

Note that if the target is a subscripted variable, this message only shows the array name, without any subscripts.

IEL0665I E [PROLOGUE CODE.] 'DATE' ATTRIBUTE OF SOURCE IGNORED IN ASSIGNMENT TO TARGET D.

[PROLOGUE CODE.] TARGET D IN ASSIGNMENT DOES NOT HAVE THE 'DATE' ATTRIBUTE BUT SOURCE DOES. THE 'DATE' ATTRIBUTE HAS BEEN IGNORED FOR THIS ASSIGNMENT.

Example:

```
DCL X CHAR(6);
X=DATE();
```

Explanation: If the source in an assignment statement has the DATE attribute, the target should also. The compiler has corrected this by ignoring the DATE attribute in performing this assignment.

For a DO statement, this message may be issued for the initial assignment of the loop control variable or for a 'REPEAT' clause.

Note that if the target is a subscripted variable, this message only shows the array name, without any subscripts.

IEL0666I E [PROLOGUE CODE.] 'DATE' ATTRIBUTE OF SOURCE IGNORED IN ASSIGNMENT.

[PROLOGUE CODE.] TARGET IN ASSIGNMENT DOES NOT HAVE THE 'DATE' ATTRIBUTE BUT SOURCE DOES. THE 'DATE' ATTRIBUTE HAS BEEN IGNORED FOR THIS ASSIGNMENT.

Example:

```
DCL X CHAR(8);
SUBSTR(X,3)=DATE();
```

Explanation: If the source in an assignment statement has the DATE attribute, the target should also. The compiler has corrected this by ignoring the DATE attribute in performing this assignment.

For a DO statement, this message may be issued for the initial assignment of the loop control variable or for a 'REPEAT' clause.

IEL0667I W [PROLOGUE CODE.] 'DATE' ATTRIBUTE OF TARGET D IGNORED IN ASSIGNMENT.

[PROLOGUE CODE.] TARGET D IN ASSIGNMENT HAS THE 'DATE' ATTRIBUTE BUT SOURCE CONSTANT DOES NOT. THE 'DATE' ATTRIBUTE HAS BEEN IGNORED FOR THIS ASSIGNMENT.

Example:

```
DCL X CHAR(6) DATE('YYMMDD');
X='';
X='971027';
```

Explanation: If a constant is used as the source in an assignment, the constant should be assigned to a target that does not have DATE attribute. The compiler has corrected this by ignoring the DATE attribute in performing this assignment.

For a DO statement, this message may be issued for the initial assignment of the loop control variable or for a 'REPEAT' clause.

Note that if the target is a subscripted variable, this message only shows the array name, without any subscripts.

IEL0668I E 'DATE' ATTRIBUTE IGNORED IN COMPARISON OF 'DATE' OPERAND WITH

NON-DATE OPERAND.

COMPARISON OF 'DATE' OPERAND WITH NON-DATE OPERAND IS INVALID. THE 'DATE' ATTRIBUTE HAS BEEN IGNORED.

Example:

```
DCL X CHAR(6) DATE('YYMMDD');
DCL Y FIXED BIN(31);
IF X=Y  THEN ...
```

Explanation: In a comparison, if one operand has the DATE attribute, the other should also. The compiler has corrected this by ignoring the DATE attribute.

IEL0678I E 'DATE' ATTRIBUTE OF ARGUMENT N TO BUILTIN T HAS BEEN IGNORED.

ARGUMENT NUMBER N TO BUILTIN FUNCTION T MUST NOT HAVE THE 'DATE' ATTRIBUTE. THE 'DATE' ATTRIBUTE HAS BEEN IGNORED.

Example:

```
DCL X FIXED DEC(6) DATE;
DCL Y FIXED DEC(6);
DCL ADD BUILTIN;
X=ADD(X,Y,6);
```

Explanation: Variables with the DATE attribute may only be used in comparisons or assignments. The compiler does not support builtin arithmetic that takes into account the DATE pattern of a variable.

IEL0679I E 'DATE' ATTRIBUTE HAS BEEN IGNORED IN ARITHMETIC OPERATION.

A 'DATE' OPERAND IS NOT VALID IN AN ARITHMETIC OPERATION. THE 'DATE' ATTRIBUTE HAS BEEN IGNORED.

Example:

```
DCL X FIXED DEC(6) DATE;
X=X+1;
```

Explanation: Variables with the DATE attribute may only be used in comparisons or assignments. The compiler does not support arithmetic that takes into account the DATE pattern of a variable.

IEL0696I E [PROLOGUE CODE.] 'DATE' PARAMETER ATTRIBUTE FOR STRUCTURE ELEMENT T IN ARGUMENT N TO ENTRY D HAS BEEN IGNORED.

[PROLOGUE CODE.] ARGUMENT NUMBER N TO ENTRY D HAS STRUCTURE ELEMENT T THAT DOES NOT HAVE THE 'DATE' ATTRIBUTE BUT THE CORRESPONDING PARAMETER DOES. THE 'DATE' ATTRIBUTE HAS BEEN IGNORED.

Example:

```
DCL X ENTRY(1, 2 CHAR(6) DATE, 2 CHAR(1));
DCL 1 ST, 2 STA CHAR(6), 2 STB CHAR(1);
```

```
CALL X(ST);
```

Explanation: If a parameter descriptor has the DATE attribute, then the corresponding argument should also have the DATE attribute. In the example above, element STA will be diagnosed.

Note that if the ENTRY is a subscripted variable, this message only shows the array name, without any subscripts. Also, if the structure element name is not available, for example, because it is a structure expression, the name T shown above is '****'.

IEL0697I E [PROLOGUE CODE.] 'DATE' PARAMETER ATTRIBUTE IGNORED FOR ARGUMENT N TO ENTRY D.

[PROLOGUE CODE.] ARGUMENT NUMBER N TO ENTRY D DOES NOT HAVE THE 'DATE' ATTRIBUTE BUT THE CORRESPONDING PARAMETER DOES. THE 'DATE' ATTRIBUTE HAS BEEN IGNORED.

Example:

```
DCL X ENTRY(CHAR(6) DATE, CHAR(6) DATE);
DCL (A,B) CHAR(6);
CALL X(A,B);
```

Explanation: If a parameter descriptor has the DATE attribute, then the corresponding argument should also have the DATE attribute. In the example above, both arguments will be diagnosed.

Note that if the ENTRY is a subscripted variable, this message only shows the array name, without any subscripts.

IEL0698I E [PROLOGUE CODE.] 'DATE' ATTRIBUTE OF STRUCTURE ELEMENT T IN ARGUMENT N TO ENTRY D HAS BEEN IGNORED.

[PROLOGUE CODE.] ARGUMENT NUMBER N TO ENTRY D HAS STRUCTURE ELEMENT T THAT HAS THE 'DATE' ATTRIBUTE BUT THE CORRESPONDING PARAMETER DOES NOT. THE 'DATE' ATTRIBUTE HAS BEEN IGNORED.

Example:

```
DCL X ENTRY(1, 2 CHAR(6), 2 CHAR(1));
DCL 1 ST, 2 STA CHAR(6) DATE, 2 STB CHAR(1);
CALL X(ST);
```

Explanation: If an argument has the DATE attribute, then the corresponding parameter should also have the DATE attribute. In the example above, element STA will be diagnosed.

Note that if the ENTRY is a subscripted variable, this message only shows the array name, without any subscripts. Also, if the structure element name is not available for example because it is a structure expression, the name T shown above is '****'.

IEL0699I E [PROLOGUE CODE.] 'DATE' ATTRIBUTE OF ARGUMENT N TO ENTRY D HAS BEEN IGNORED.

[PROLOGUE CODE.] ARGUMENT NUMBER N TO ENTRY D HAS THE 'DATE' ATTRIBUTE BUT THE CORRESPONDING PARAMETER DOES NOT. THE 'DATE'

ATTRIBUTE HAS BEEN IGNORED.

Example:

```
DCL X ENTRY(CHAR(6), CHAR(6));
DCL (A,B) CHAR(6) DATE;
CALL X(A,B);
```

Explanation: If an argument has the DATE attribute, then the corresponding parameter descriptor should also have the DATE attribute. In the example above, both arguments will be diagnosed.

Note that if the ENTRY is a subscripted variable, this message only shows the array name, without any subscripts.

IEL0700I W [PROLOGUE CODE.] 'DATE' PARAMETER ATTRIBUTE IGNORED FOR ARGUMENT N TO ENTRY D.

[PROLOGUE CODE.] ARGUMENT NUMBER N TO ENTRY D DOES NOT HAVE THE 'DATE' ATTRIBUTE BUT THE CORRESPONDING PARAMETER DOES. THE 'DATE' ATTRIBUTE HAS BEEN IGNORED.

Example:

```
DCL X ENTRY(CHAR(6) DATE);
CALL X('971231');
```

Explanation: If a parameter descriptor has the DATE attribute, then the corresponding argument should also have the DATE attribute. In the example above, the character constant argument will be diagnosed with this warning message.

Note that if the ENTRY is a subscripted variable, this message only shows the array name, without any subscripts.

IEL0744I S 'DATE' PATTERN OF D IS INVALID.

'DATE' PATTERN IN DECLARATION OF D IS INVALID. 'DATE' ATTRIBUTE HAS BEEN IGNORED.

Example:

```
DCL X CHAR(6) DATE('YYDDDD');
DCL Y CHAR(6) DATE('yymmdd');
```

Explanation: In the example above, 'YYDDDD' and 'yymmdd' will be diagnosed.

IEL0745I S 'DATE' ATTRIBUTE OF D IS ONLY VALID WITH NON-VARYING CHARACTER OR ARITHMETIC PICTURE OR FIXED DECIMAL.

'DATE' ATTRIBUTE IS ONLY VALID WITH NON-VARYING CHARACTER OR ARITHMETIC PICTURE OR FIXED DECIMAL. 'DATE' ATTRIBUTE IGNORED IN DECLARATION OF D.

Example:

```
  DCL W CHAR(6) DATE VARYING;
  DCL X FIXED BIN(31) DATE;
  DCL Y FIXED DEC(6)  DATE COMPLEX;
  DCL P PIC'999999' DATE COMPLEX;
```

Explanation: This message may be issued when incorrect attributes are specified for a variable declaration, parameter descriptor, or a function reference return.

IEL0746I S CHARACTER OR PICTURE LENGTH OF D MUST BE A CONSTANT AND MUST MATCH DATE PATTERN LENGTH.

CHARACTER OR PICTURE LENGTH MUST BE A CONSTANT AND MUST MATCH THE LENGTH OF THE DATE PATTERN. 'DATE' ATTRIBUTE IGNORED IN DECLARATION OF D.

Example:

```
  DCL X CHAR(*) CONTROLLED DATE;
  DCL Y CHAR(N) DATE;
  DCL Z PIC'9999' DATE('YY');
```

IEL0747I S DECIMAL PRECISION OF D MUST MATCH DATE PATTERN LENGTH.

DECIMAL PRECISION MUST MATCH THE LENGTH OF THE DATE PATTERN. 'DATE' ATTRIBUTE IGNORED IN DECLARATION OF D.

Example:

```
  DCL X FIXED DEC(8) DATE('YYDDD');
  DCL Y FIXED DEC(9) DATE;
```

IEL0748I S DECIMAL SCALING FACTOR OF D MUST BE ZERO WITH 'DATE' ATTRIBUTE.

DECIMAL SCALING FACTOR MUST BE ZERO WHEN 'DATE' ATTRIBUTE IS SPECIFIED. 'DATE' ATTRIBUTE IGNORED IN DECLARATION OF D.

Example:

```
  DCL X FIXED DEC(5,3) DATE('YYDDD');
  DCL Y FIXED DEC(6,2) DATE;
```

IEL0749I S PICTURE SPECIFICATION OF D MUST BE ALL 9'S WITH 'DATE' ATTRIBUTE.

PICTURE SPECIFICATION MUST BE ALL 9'S WHEN 'DATE' ATTRIBUTE IS SPECIFIED. 'DATE' ATTRIBUTE IGNORED IN DECLARATION OF D.

Example:

```
DCL X PIC'99XXX' DATE('YYDDD');
DCL Y PIC'AAAAAA' DATE;
```

IEL0750I S DATE PATTERN OF D WITH ALPHABETIC CHARACTERS IS ONLY VALID WITH CHARACTER DATA.

A DATE PATTERN WITH ALPHABETIC CHARACTERS IS ONLY VALID WITH THE CHARACTER DATA TYPE. 'DATE' ATTRIBUTE IGNORED IN DECLARATION OF D.

Example:

```
DCL X FIXED DEC(7) DATE('YYMmmDD');
DCL Y PIC'9999999' DATE('YYMMMDD');
```

Explanation: A DATE pattern with alphabetic characters can only be declared with the character data type versus the arithmetic data type.

IEL0780I E 'DATE' ATTRIBUTE IGNORED IN ARITHMETIC OPERATION GENERATED FOR 'BY' CLAUSE.

'DATE' OPERAND IS NOT VALID IN ARITHMETIC OPERATION GENERATED FOR 'BY' CLAUSE. THE 'DATE' ATTRIBUTE HAS BEEN IGNORED.

Example:

```
DCL D FIXED DEC(6) DATE;
DO I=1 TO N BY D;
DO D=1 TO N BY 1;
```

Explanation: Variables with the DATE attribute may only be used in comparisons or assignments. The compiler does not support arithmetic that takes into account the DATE pattern of a variable. In particular, this applies to the arithmetic operation implicitly generated for the BY clause of a DO statement. The compiler has corrected this by ignoring the DATE attribute.

IEL0781I W 'DATE' ATTRIBUTE OF CONTROL VARIABLE IGNORED IN 'TO' CLAUSE COMPARISON.

COMPARISON OF CONTROL VARIABLE WITH NON-DATE CONSTANT IS INVALID. THE 'DATE' ATTRIBUTE OF THE CONTROL VARIABLE HAS BEEN IGNORED IN THE 'TO' CLAUSE COMPARISON.

Example:

```
DCL N FIXED DEC(6) DATE;
DO N=1 TO 20;
```

Explanation: In a comparison, if one operand has the DATE attribute, the other should also. In particular, this applies to the comparison implicitly generated for the TO clause of a DO statement control variable. The compiler has corrected this by ignoring the DATE attribute of the control variable.

IEL0782I W CONSTANT IN 'TO' CLAUSE IS ASSUMED TO HAVE SAME 'DATE' PATTERN AS CONTROL VARIABLE.

COMPARISON OF CONTROL VARIABLE WITH CONSTANT HAS BEEN ACCEPTED. CONSTANT IN 'TO' CLAUSE IS ASSUMED TO HAVE THE SAME 'DATE' PATTERN AS THE CONTROL VARIABLE.

Example:

```
DCL N FIXED DEC(5) DATE('YYDDD');
DO N=1 TO 97030;
```

Explanation: In a comparison, if one operand has the DATE attribute, the other should also. In particular, this applies to the comparison implicitly generated for the TO clause of a DO statement control variable. The compiler has corrected this by assuming that the constant in the TO clause has the same DATE attribute as the DO statement control variable. In the example above, the constant 97030 is assumed to have the DATE pattern 'YYDDD', the same as N.

IEL0783I E NON-DATE OPERAND IN 'TO' CLAUSE COMPARISON IS ASSUMED TO HAVE SAME PATTERN AS 'DATE' OPERAND.

COMPARISON OF 'DATE' OPERAND WITH NON-DATE OPERAND IS INVALID. THE NON-DATE OPERAND IS ASSUMED TO HAVE THE SAME PATTERN AS THE 'DATE' OPERAND IN THE 'TO' CLAUSE COMPARISON.

Example:

```
DCL N FIXED DEC(5) DATE('YYDDD');
DCL M FIXED DEC(5);
DO N=1 TO M;
DO M=1 TO N;
```

Explanation: In a comparison, if one operand has the DATE attribute, the other should also. In particular, this applies to the comparison implicitly generated for the TO clause of a DO statement control variable. The compiler has corrected this by assuming that the non-DATE operand has the same DATE attribute as the DATE operand. In the preceding examples, M is assumed to have the DATE pattern 'YYDDD', the same as N.

IEL0784I E 'DATE' ATTRIBUTE IGNORED IN 'TO' CLAUSE COMPARISON.

COMPARISON OF 'DATE' OPERAND WITH NON-DATE OPERAND IS INVALID. THE 'DATE' ATTRIBUTE HAS BEEN IGNORED IN THE 'TO' CLAUSE COMPARISON.

Example:

```
DCL N FIXED DEC(6) DATE;
DCL M FIXED BIN(31);
DO N=1 TO M;
```

Explanation: In a comparison, if one operand has the DATE attribute, the other should also. In particular, this applies to the comparison implicitly generated for the TO clause of a DO statement control variable. The compiler has corrected this by ignoring the DATE attribute.

IEL0793I E 'DATE' ATTRIBUTE OF PROCEDURE 'RETURNS' OPTION HAS BEEN IGNORED.

EXPRESSION IN RETURN STATEMENT DOES NOT HAVE THE 'DATE' ATTRIBUTE BUT 'RETURNS' OPTION OF THE PROCEDURE DOES. 'DATE' ATTRIBUTE HAS BEEN IGNORED.

Example:

```
P: PROC RETURNS(CHAR(6) DATE);
   DCL C CHAR(6);
   RETURN(C);
   END;
```

Explanation: In the example above, C does not have the DATE attribute, but the procedure P does.

IEL0794I E 'DATE' ATTRIBUTE OF RETURNED EXPRESSION HAS BEEN IGNORED.

EXPRESSION IN RETURN STATEMENT HAS THE 'DATE' ATTRIBUTE BUT 'RETURNS' OPTION OF THE PROCEDURE DOES NOT. 'DATE' ATTRIBUTE HAS BEEN IGNORED.

Example:

```
P: PROC RETURNS(CHAR(6));
   RETURN(DATE());
   END;
```

Explanation: In the example above, builtin DATE() has the DATE attribute, but the procedure P does not.

IEL0795I W RETURNED EXPRESSION DOES NOT HAVE THE 'DATE' ATTRIBUTE BUT 'RETURNS' OPTION OF AN 'ENTRY' IN THIS BLOCK DOES.

EXPRESSION IN RETURN STATEMENT DOES NOT HAVE THE 'DATE' ATTRIBUTE BUT 'RETURNS' OPTION OF AN 'ENTRY' IN THIS BLOCK DOES. 'DATE' ATTRIBUTE WILL BE IGNORED IF THE INVALID COMBINATION OF 'RETURN' AND 'ENTRY' IS USED.

Example:

```
P: PROC(N)  RETURNS(CHAR(6) DATE);
E: ENTRY(N) RETURNS(CHAR(6));
   DCL N FIXED;
   DCL C CHAR(6);
   IF N=0 THEN RETURN('970704');
          ELSE RETURN(C);
     .
     .
     .
   END;
```

Explanation: In the example above, both RETURN statements will be flagged with this message.

IEL0796I W RETURNED EXPRESSION HAS THE 'DATE' ATTRIBUTE BUT 'RETURNS' OPTION OF AN 'ENTRY' IN THIS BLOCK DOES NOT.

EXPRESSION IN RETURN STATEMENT HAS THE 'DATE' ATTRIBUTE BUT 'RETURNS' OPTION OF AN 'ENTRY' IN THIS BLOCK DOES NOT. 'DATE' ATTRIBUTE WILL BE IGNORED IF THE INVALID COMBINATION OF 'RETURN' AND 'ENTRY' IS USED.

Example:

```
P: PROC(N)  RETURNS(CHAR(6) DATE);
E: ENTRY(N) RETURNS(CHAR(6));
   DCL N FIXED;
   IF N=0 THEN RETURN(DATE());
     .
     .
     .
   END;
```

Explanation: In the example above, the DATE() builtin has the DATE attribute but entry E does not.

IEL0899I U PL/I MILLENNIUM LANGUAGE EXTENSIONS (MLE) PRODUCT MUST BE INSTALLED AND ACCESSIBLE TO THE COMPILER TO USE RULES(LAXCOMMENT).

THE PL/I MILLENNIUM LANGUAGE EXTENSIONS (MLE) PRODUCT MUST BE INSTALLED AND ACCESSIBLE TO THE COMPILER TO USE THE RULES(LAXCOMMENT) COMPILER OPTION. COMPILATION TERMINATED.

Explanation: The use of the RULES(LAXCOMMENT) compiler option requires that the VisualAge PL/I Millennium Language Extensions (MLE) product be installed and accessible to the PL/I compiler. This product provides Year 2000 support.

IEL0900I U PL/I MILLENNIUM LANGUAGE EXTENSIONS (MLE) PRODUCT MUST BE INSTALLED AND ACCESSIBLE TO THE COMPILER TO USE RESPECT(DATE).

THE PL/I MILLENNIUM LANGUAGE EXTENSIONS (MLE) PRODUCT MUST BE INSTALLED AND ACCESSIBLE TO THE COMPILER TO USE THE RESPECT(DATE) COMPILER OPTION. COMPILATION TERMINATED.

Explanation: The use of the RESPECT(DATE) compiler option requires that the VisualAge Pl/I Millennium Language Extensions (MLE) product be installed and accessible to the PL/I compiler. This product provides Year 2000 support.

IEL0901I U PL/I MILLENNIUM LANGUAGE EXTENSIONS (MLE) PRODUCT MUST BE INSTALLED AND ACCESSIBLE TO THE COMPILER TO USE 'DAYS' OR 'DAYSTODATE'.

THE PL/I MILLENNIUM LANGUAGE EXTENSIONS (MLE) PRODUCT MUST BE INSTALLED AND ACCESSIBLE TO THE COMPILER TO USE THE 'DAYS' OR 'DAYSTODATE' BUILTIN. COMPILATION TERMINATED.

Explanation: The use of the DAYS or DAYSTODATE builtins requires that the VisualAge PL/I Millennium Language Extensions (MLE) product be installed and accessible to the PL/I compiler. This product provides Year 2000 support.

3.2 PL/I TSO Prompter Messages

IKJ65084I RESPECT SUBFIELD IS MISSING OR INVALID. VALID ARGUMENT IS DATE.

Example:

```
RESPECT(DATES)
```

Explanation: The argument for the RESPECT option must be DATE.

Programmer Response: You should reenter a correct value. If you enter a null line, the default value is assumed.

IKJ65085I RESPECT DEFAULT ASSUMED

Explanation: You have entered a null line in response to a request to reenter the argument for the RESPECT option.

IKJ65086I RULES SUBFIELD IS MISSING OR INVALID. VALID ARGUMENTS ARE LAXCOM, NOLAXCOM, LAXCOMMENT, OR NOLAXCOMMENT.

Example:

```
RULES(LAXCMT)
```

Explanation: The argument for the RULES option must be LAXCOM, NOLAXCOM, LAXCOMMENT, or NOLAXCOMMENT. LAXCOM and LAXCOMMENT allow comments of the form /*/.

Programmer Response: You should reenter a correct value. If you enter a null line, the default value is assumed.

IKJ65087I RULES DEFAULT ASSUMED

Explanation: You have entered a null line in response to a request to reenter the argument for the RULES option.

IKJ65088I WINDOW SUBFIELD IS MISSING OR INVALID. VALID VALUES ARE -1 THROUGH -99, 0, 1582 THROUGH 9999.

Example:

```
WINDOW(1)
```

Explanation: The argument for the WINDOW option must be a two-digit negative integer, zero, or a four-digit positive integer between 1582 and 9999, inclusive.

Programmer Response: You should reenter a correct value. If you enter a null line, the default value is assumed.

IKJ65089I WINDOW DEFAULT ASSUMED

Explanation: You have entered a null line in response to a request to reenter the argument for the WINDOW option.

| 4.0 Chapter 4. PL/I VSE MLE Messages

| This appendix documents information that is NOT intended to be used as
| Programming Interfaces.

| The following descriptions are for the PL/I VSE compile-time messages and
| codes that are either changed or new as a result of VisualAge PL/I
| Millennium Language Extensions for VSE/ESA. In the case of message
| IEL02301, a new restriction (299) exists as shown.

Subtopics:

| 4.1 Compile-time messages

| **IEL0230I U COMPILER ERROR OR RESTRICTION NUMBER 299 DURING PHASE 'II'.**

| **Explanation:** The program has too many statements that use variables
| declared with the DATE attribute. Approximately 10,000 references with
| the DATE attribute are allowed in program statements. For example, in
| assignment statements, IF statements, etc.

| **Programmer Response:** Reduce the usage of variables declared with the DATE
| attribute, particularly those that are based or subscripted.
| Alternatively, reduce the usage of the DATE builtin or the usage of
| functions that return a value with the DATE attribute.

| **IEL0661I W 'DATE' ATTRIBUTE IGNORED IN COMPARISON OF 'DATE' OPERAND WITH**
| **NON-DATE CONSTANT.**

| **COMPARISON OF 'DATE' OPERAND WITH NON-DATE CONSTANT IS INVALID.**
| **THE 'DATE' ATTRIBUTE HAS BEEN IGNORED.**

| **Example:**

```
|   DCL X CHAR(6) DATE('YYMMDD');
|   IF X='' THEN ...
```

| **Explanation:** In a comparison, if one operand has the DATE attribute, the
| other should also. The compiler has corrected this by ignoring the DATE

| attribute.

| **IEL0662I W CONSTANT IN COMPARISON IS ASSUMED TO HAVE SAME PATTERN AS**
| **'DATE' OPERAND.**

| **COMPARISON OF 'DATE' OPERAND WITH CONSTANT HAS BEEN ACCEPTED.**
| **CONSTANT IS ASSUMED TO HAVE THE SAME PATTERN AS THE OPERAND WITH**
| **THE 'DATE' ATTRIBUTE.**

| **Example:**

```
|   DCL X CHAR(6) DATE('YYMMDD');
|   IF X>'971122' THEN ...
```

| **Explanation:** In a comparison, if one operand has the DATE attribute, the
| other should also. The compiler has corrected this by assuming that the
| constant has the same DATE attribute. In the example above, the constant
| '971122' is assumed to have the DATE pattern 'YYMMDD', the same as X.

| **IEL0663I E NON-DATE OPERAND IN COMPARISON IS ASSUMED TO HAVE SAME PATTERN**
| **AS 'DATE' OPERAND.**

| **COMPARISON OF 'DATE' OPERAND WITH NON-DATE OPERAND IS INVALID.**
| **THE NON-DATE OPERAND IS ASSUMED TO HAVE THE SAME PATTERN AS THE**
| **OPERAND WITH THE 'DATE' ATTRIBUTE.**

| **Example:**

```
|   DCL X CHAR(6) DATE('YYMMDD');
|   DCL Y CHAR(6);
|   IF X>Y THEN ...
```

| **Explanation:** In a comparison, if one operand has the DATE attribute, the
| other should also. The compiler has corrected this by assuming that the
| non-DATE operand has the same DATE attribute. In the example above, Y is
| assumed to have the DATE pattern 'YYMMDD', the same as X.

| **IEL0664I E [PROLOGUE CODE.] 'DATE' ATTRIBUTE OF TARGET D IGNORED IN**
| **ASSIGNMENT.**

| **[PROLOGUE CODE.] SOURCE IN ASSIGNMENT DOES NOT HAVE THE 'DATE'**
| **ATTRIBUTE BUT TARGET D DOES. THE 'DATE' ATTRIBUTE HAS BEEN**
| **IGNORED FOR THIS ASSIGNMENT.**

| **Example:**

```
|   DCL X CHAR(6) DATE('YYMMDD');
|   DCL Y CHAR(6);
|   X=Y;
```

| **Explanation:** If the target in an assignment statement has the DATE
| attribute, the source should also. The compiler has corrected this by
| ignoring the DATE attribute in performing this assignment.

| For a DO statement, this message may be issued for the initial assignment
| of the loop control variable or for a 'REPEAT' clause.

| Note that if the target is a subscripted variable, this message only shows

| the array name, without any subscripts.

| **IEL0665I E [PROLOGUE CODE.] 'DATE' ATTRIBUTE OF SOURCE IGNORED IN**
| **ASSIGNMENT TO TARGET D.**

| **[PROLOGUE CODE.] TARGET D IN ASSIGNMENT DOES NOT HAVE THE**
| **'DATE' ATTRIBUTE BUT SOURCE DOES. THE 'DATE' ATTRIBUTE HAS BEEN**
| **IGNORED FOR THIS ASSIGNMENT.**

| **Example:**

```
|   DCL X CHAR(6);
|   X=DATE();
```

| **Explanation:** If the source in an assignment statement has the DATE
| attribute, the target should also. The compiler has corrected this by
| ignoring the DATE attribute in performing this assignment.

| For a DO statement, this message may be issued for the initial assignment
| of the loop control variable or for a 'REPEAT' clause.

| Note that if the target is a subscripted variable, this message only shows
| the array name, without any subscripts.

| **IEL0666I E [PROLOGUE CODE.] 'DATE' ATTRIBUTE OF SOURCE IGNORED IN**
| **ASSIGNMENT.**

| **[PROLOGUE CODE.] TARGET IN ASSIGNMENT DOES NOT HAVE THE 'DATE'**
| **ATTRIBUTE BUT SOURCE DOES. THE 'DATE' ATTRIBUTE HAS BEEN**
| **IGNORED FOR THIS ASSIGNMENT.**

| **Example:**

```
|   DCL X CHAR(8);
|   SUBSTR(X,3)=DATE();
```

| **Explanation:** If the source in an assignment statement has the DATE
| attribute, the target should also. The compiler has corrected this by
| ignoring the DATE attribute in performing this assignment.

| For a DO statement, this message may be issued for the initial assignment
| of the loop control variable or for a 'REPEAT' clause.

| **IEL0667I W [PROLOGUE CODE.] 'DATE' ATTRIBUTE OF TARGET D IGNORED IN**
| **ASSIGNMENT.**

| **[PROLOGUE CODE.] TARGET D IN ASSIGNMENT HAS THE 'DATE'**
| **ATTRIBUTE BUT SOURCE CONSTANT DOES NOT. THE 'DATE' ATTRIBUTE**
| **HAS BEEN IGNORED FOR THIS ASSIGNMENT.**

| **Example:**

```
|   DCL X CHAR(6) DATE('YYMMDD');
|   X='';
|   X='971027';
```

| **Explanation:** If a constant is used as the source in an assignment, the

| constant should be assigned to a target that does not have DATE attribute.
| The compiler has corrected this by ignoring the DATE attribute in
| performing this assignment.

| For a DO statement, this message may be issued for the initial assignment
| of the loop control variable or for a 'REPEAT' clause.

| Note that if the target is a subscripted variable, this message only shows
| the array name, without any subscripts.

**| IEL0668I E 'DATE' ATTRIBUTE IGNORED IN COMPARISON OF 'DATE' OPERAND WITH
| NON-DATE OPERAND.**

**| COMPARISON OF 'DATE' OPERAND WITH NON-DATE OPERAND IS INVALID.
| THE 'DATE' ATTRIBUTE HAS BEEN IGNORED.**

| Example:

```
|   DCL X CHAR(6) DATE('YYMMDD');
|   DCL Y FIXED BIN(31);
|   IF X=Y  THEN ...
```

| Explanation: In a comparison, if one operand has the DATE attribute, the
| other should also. The compiler has corrected this by ignoring the DATE
| attribute.

| IEL0678I E 'DATE' ATTRIBUTE OF ARGUMENT N TO BUILTIN T HAS BEEN IGNORED.

**| ARGUMENT NUMBER N TO BUILTIN FUNCTION T MUST NOT HAVE THE 'DATE'
| ATTRIBUTE. THE 'DATE' ATTRIBUTE HAS BEEN IGNORED.**

| Example:

```
|   DCL X FIXED DEC(6) DATE;
|   DCL Y FIXED DEC(6);
|   DCL ADD BUILTIN;
|   X=ADD(X,Y,6);
```

| Explanation: Variables with the DATE attribute may only be used in
| comparisons or assignments. The compiler does not support builtin
| arithmetic that takes into account the DATE pattern of a variable.

| IEL0679I E 'DATE' ATTRIBUTE HAS BEEN IGNORED IN ARITHMETIC OPERATION.

**| A 'DATE' OPERAND IS NOT VALID IN AN ARITHMETIC OPERATION. THE
| 'DATE' ATTRIBUTE HAS BEEN IGNORED.**

| Example:

```
|   DCL X FIXED DEC(6) DATE;
|   X=X+1;
```

| Explanation: Variables with the DATE attribute may only be used in
| comparisons or assignments. The compiler does not support arithmetic that
| takes into account the DATE pattern of a variable.

IEL0696I E [PROLOGUE CODE.] 'DATE' PARAMETER ATTRIBUTE FOR STRUCTURE ELEMENT T IN ARGUMENT N TO ENTRY D HAS BEEN IGNORED.

[PROLOGUE CODE.] ARGUMENT NUMBER N TO ENTRY D HAS STRUCTURE ELEMENT T THAT DOES NOT HAVE THE 'DATE' ATTRIBUTE BUT THE CORRESPONDING PARAMETER DOES. THE 'DATE' ATTRIBUTE HAS BEEN IGNORED.

Example:

```
  DCL X ENTRY(1, 2 CHAR(6) DATE, 2 CHAR(1));
  DCL 1 ST, 2 STA CHAR(6), 2 STB CHAR(1);
  CALL X(ST);
```

Explanation: If a parameter descriptor has the DATE attribute, then the corresponding argument should also have the DATE attribute. In the example above, element STA will be diagnosed.

Note that if the ENTRY is a subscripted variable, this message only shows the array name, without any subscripts. Also, if the structure element name is not available, for example, because it is a structure expression, the name T shown above is '****'.

IEL0697I E [PROLOGUE CODE.] 'DATE' PARAMETER ATTRIBUTE IGNORED FOR ARGUMENT N TO ENTRY D.

[PROLOGUE CODE.] ARGUMENT NUMBER N TO ENTRY D DOES NOT HAVE THE 'DATE' ATTRIBUTE BUT THE CORRESPONDING PARAMETER DOES. THE 'DATE' ATTRIBUTE HAS BEEN IGNORED.

Example:

```
  DCL X ENTRY(CHAR(6) DATE, CHAR(6) DATE);
  DCL (A,B) CHAR(6);
  CALL X(A,B);
```

Explanation: If a parameter descriptor has the DATE attribute, then the corresponding argument should also have the DATE attribute. In the example above, both arguments will be diagnosed.

Note that if the ENTRY is a subscripted variable, this message only shows the array name, without any subscripts.

IEL0698I E [PROLOGUE CODE.] 'DATE' ATTRIBUTE OF STRUCTURE ELEMENT T IN ARGUMENT N TO ENTRY D HAS BEEN IGNORED.

[PROLOGUE CODE.] ARGUMENT NUMBER N TO ENTRY D HAS STRUCTURE ELEMENT T THAT HAS THE 'DATE' ATTRIBUTE BUT THE CORRESPONDING PARAMETER DOES NOT. THE 'DATE' ATTRIBUTE HAS BEEN IGNORED.

Example:

```
  DCL X ENTRY(1, 2 CHAR(6), 2 CHAR(1));
  DCL 1 ST, 2 STA CHAR(6) DATE, 2 STB CHAR(1);
  CALL X(ST);
```

Explanation: If an argument has the DATE attribute, then the corresponding parameter should also have the DATE attribute. In the example above, element STA will be diagnosed.

Note that if the ENTRY is a subscripted variable, this message only shows the array name, without any subscripts. Also, if the structure element name is not available for example because it is a structure expression, the name T shown above is '****'.

IEL0699I E [PROLOGUE CODE.] 'DATE' ATTRIBUTE OF ARGUMENT N TO ENTRY D HAS BEEN IGNORED.

[PROLOGUE CODE.] ARGUMENT NUMBER N TO ENTRY D HAS THE 'DATE' ATTRIBUTE BUT THE CORRESPONDING PARAMETER DOES NOT. THE 'DATE' ATTRIBUTE HAS BEEN IGNORED.

Example:

```
DCL X ENTRY(CHAR(6), CHAR(6));
DCL (A,B) CHAR(6) DATE;
CALL X(A,B);
```

Explanation: If an argument has the DATE attribute, then the corresponding parameter descriptor should also have the DATE attribute. In the example above, both arguments will be diagnosed.

Note that if the ENTRY is a subscripted variable, this message only shows the array name, without any subscripts.

IEL0700I W [PROLOGUE CODE.] 'DATE' PARAMETER ATTRIBUTE IGNORED FOR ARGUMENT N TO ENTRY D.

[PROLOGUE CODE.] ARGUMENT NUMBER N TO ENTRY D DOES NOT HAVE THE 'DATE' ATTRIBUTE BUT THE CORRESPONDING PARAMETER DOES. THE 'DATE' ATTRIBUTE HAS BEEN IGNORED.

Example:

```
DCL X ENTRY(CHAR(6) DATE);
CALL X('971231');
```

Explanation: If a parameter descriptor has the DATE attribute, then the corresponding argument should also have the DATE attribute. In the example above, the character constant argument will be diagnosed with this warning message.

Note that if the ENTRY is a subscripted variable, this message only shows the array name, without any subscripts.

IEL0744I S 'DATE' PATTERN OF D IS INVALID.

'DATE' PATTERN IN DECLARATION OF D IS INVALID. 'DATE' ATTRIBUTE HAS BEEN IGNORED.

Example:

```
DCL X CHAR(6) DATE('YYDDDD');
DCL Y CHAR(6) DATE('yymmdd');
```

Explanation: In the example above, 'YYDDDD' and 'yymmdd' will be diagnosed.

IEL0745I S 'DATE' ATTRIBUTE OF D IS ONLY VALID WITH NON-VARYING CHARACTER OR ARITHMETIC PICTURE OR FIXED DECIMAL.

'DATE' ATTRIBUTE IS ONLY VALID WITH NON-VARYING CHARACTER OR ARITHMETIC PICTURE OR FIXED DECIMAL. 'DATE' ATTRIBUTE IGNORED IN DECLARATION OF D.

Example:

```
  DCL W CHAR(6) DATE VARYING;
  DCL X FIXED BIN(31) DATE;
  DCL Y FIXED DEC(6)  DATE COMPLEX;
  DCL Z GRAPHIC(6) DATE;
  DCL P PIC'999999' DATE COMPLEX;
```

Explanation: This message may be issued when incorrect attributes are specified for a variable declaration, parameter descriptor, or a function reference return.

IEL0746I S CHARACTER OR PICTURE LENGTH OF D MUST BE A CONSTANT AND MUST MATCH DATE PATTERN LENGTH.

CHARACTER OR PICTURE LENGTH MUST BE A CONSTANT AND MUST MATCH THE LENGTH OF THE DATE PATTERN. 'DATE' ATTRIBUTE IGNORED IN DECLARATION OF D.

Example:

```
  DCL X CHAR(*) CONTROLLED DATE;
  DCL Y CHAR(N) DATE;
  DCL Z PIC'9999' DATE('YY');
```

IEL0747I S DECIMAL PRECISION OF D MUST MATCH DATE PATTERN LENGTH.

DECIMAL PRECISION MUST MATCH THE LENGTH OF THE DATE PATTERN. 'DATE' ATTRIBUTE IGNORED IN DECLARATION OF D.

Example:

```
  DCL X FIXED DEC(8) DATE('YYDDD');
  DCL Y FIXED DEC(9) DATE;
```

IEL0748I S DECIMAL SCALING FACTOR OF D MUST BE ZERO WITH 'DATE' ATTRIBUTE.

DECIMAL SCALING FACTOR MUST BE ZERO WHEN 'DATE' ATTRIBUTE IS SPECIFIED. 'DATE' ATTRIBUTE IGNORED IN DECLARATION OF D.

Example:

```
  DCL X FIXED DEC(5,3) DATE('YYDDD');
```

```
|   DCL Y FIXED DEC(6,2) DATE;
```

IEL0749I S PICTURE SPECIFICATION OF D MUST BE ALL 9'S WITH 'DATE' ATTRIBUTE.

PICTURE SPECIFICATION MUST BE ALL 9'S WHEN 'DATE' ATTRIBUTE IS SPECIFIED. 'DATE' ATTRIBUTE IGNORED IN DECLARATION OF D.

Example:

```
|   DCL X PIC'99XXX' DATE('YYDDD');
|   DCL Y PIC'AAAAAA' DATE;
```

IEL0754I S DATE PATTERN OF D WITH ALPHABETIC CHARACTERS IS ONLY VALID WITH CHARACTER DATA.

A DATE PATTERN WITH ALPHABETIC CHARACTERS IS ONLY VALID WITH THE CHARACTER DATA TYPE. 'DATE' ATTRIBUTE IGNORED IN DECLARATION OF D.

Example:

```
|   DCL X FIXED DEC(7) DATE('YYMmmDD');
|   DCL Y PIC'9999999' DATE('YYMMMDD');
```

Explanation: A DATE pattern with alphabetic characters can only be declared with the character data type versus the arithmetic data type.

IEL0780I E 'DATE' ATTRIBUTE IGNORED IN ARITHMETIC OPERATION GENERATED FOR 'BY' CLAUSE.

'DATE' OPERAND IS NOT VALID IN ARITHMETIC OPERATION GENERATED FOR 'BY' CLAUSE. THE 'DATE' ATTRIBUTE HAS BEEN IGNORED.

Example:

```
|   DCL D FIXED DEC(6) DATE;
|   DO I=1 TO N BY D;
|   DO D=1 TO N BY 1;
```

Explanation: Variables with the DATE attribute may only be used in comparisons or assignments. The compiler does not support arithmetic that takes into account the DATE pattern of a variable. In particular, this applies to the arithmetic operation implicitly generated for the BY clause of a DO statement. The compiler has corrected this by ignoring the DATE attribute.

IEL0781I W 'DATE' ATTRIBUTE OF CONTROL VARIABLE IGNORED IN 'TO' CLAUSE COMPARISON.

COMPARISON OF CONTROL VARIABLE WITH NON-DATE CONSTANT IS

| **INVALID. THE 'DATE' ATTRIBUTE OF THE CONTROL VARIABLE HAS BEEN**
| **IGNORED IN THE 'TO' CLAUSE COMPARISON.**

| **Example:**

```
|   DCL N FIXED DEC(6) DATE;
|   DO N=1 TO 20;
```

| **Explanation:** In a comparison, if one operand has the DATE attribute, the
| other should also. In particular, this applies to the comparison
| implicitly generated for the TO clause of a DO statement control variable.
| The compiler has corrected this by ignoring the DATE attribute of the
| control variable.

| **IEL0782I W CONSTANT IN 'TO' CLAUSE IS ASSUMED TO HAVE SAME 'DATE' PATTERN**
| **AS CONTROL VARIABLE.**

| **COMPARISON OF CONTROL VARIABLE WITH CONSTANT HAS BEEN ACCEPTED.**
| **CONSTANT IN 'TO' CLAUSE IS ASSUMED TO HAVE THE SAME 'DATE'**
| **PATTERN AS THE CONTROL VARIABLE.**

| **Example:**

```
|   DCL N FIXED DEC(5) DATE('YYDDD');
|   DO N=1 TO 97030;
```

| **Explanation:** In a comparison, if one operand has the DATE attribute, the
| other should also. In particular, this applies to the comparison
| implicitly generated for the TO clause of a DO statement control variable.
| The compiler has corrected this by assuming that the constant in the TO
| clause has the same DATE attribute as the DO statement control variable.
| In the example above, the constant 97030 is assumed to have the DATE
| pattern 'YYDDD', the same as N.

| **IEL0783I E NON-DATE OPERAND IN 'TO' CLAUSE COMPARISON IS ASSUMED TO HAVE**
| **SAME PATTERN AS 'DATE' OPERAND.**

| **COMPARISON OF 'DATE' OPERAND WITH NON-DATE OPERAND IS INVALID.**
| **THE NON-DATE OPERAND IS ASSUMED TO HAVE THE SAME PATTERN AS THE**
| **'DATE' OPERAND IN THE 'TO' CLAUSE COMPARISON.**

| **Example:**

```
|   DCL N FIXED DEC(5) DATE('YYDDD');
|   DCL M FIXED DEC(5);
|   DO N=1 TO M;
|   DO M=1 TO N;
```

| **Explanation:** In a comparison, if one operand has the DATE attribute, the
| other should also. In particular, this applies to the comparison
| implicitly generated for the TO clause of a DO statement control variable.
| The compiler has corrected this by assuming that the non-DATE operand has
| the same DATE attribute as the DATE operand. In the preceding examples, M
| is assumed to have the DATE pattern 'YYDDD', the same as N.

| **IEL0784I E 'DATE' ATTRIBUTE IGNORED IN 'TO' CLAUSE COMPARISON.**

| **COMPARISON OF 'DATE' OPERAND WITH NON-DATE OPERAND IS INVALID.**
| **THE 'DATE' ATTRIBUTE HAS BEEN IGNORED IN THE 'TO' CLAUSE**

COMPARISON.

Example:

```
DCL N FIXED DEC(6) DATE;
DCL M FIXED BIN(31);
DO N=1 TO M;
```

Explanation: In a comparison, if one operand has the DATE attribute, the other should also. In particular, this applies to the comparison implicitly generated for the TO clause of a DO statement control variable. The compiler has corrected this by ignoring the DATE attribute.

IEL0793I E 'DATE' ATTRIBUTE OF PROCEDURE 'RETURNS' OPTION HAS BEEN IGNORED.

EXPRESSION IN RETURN STATEMENT DOES NOT HAVE THE 'DATE' ATTRIBUTE BUT 'RETURNS' OPTION OF THE PROCEDURE DOES. 'DATE' ATTRIBUTE HAS BEEN IGNORED.

Example:

```
P: PROC RETURNS(CHAR(6) DATE);
   DCL C CHAR(6);
   RETURN(C);
   END;
```

Explanation: In the example above, C does not have the DATE attribute, but the procedure P does.

IEL0794I E 'DATE' ATTRIBUTE OF RETURNED EXPRESSION HAS BEEN IGNORED.

EXPRESSION IN RETURN STATEMENT HAS THE 'DATE' ATTRIBUTE BUT 'RETURNS' OPTION OF THE PROCEDURE DOES NOT. 'DATE' ATTRIBUTE HAS BEEN IGNORED.

Example:

```
P: PROC RETURNS(CHAR(6));
   RETURN(DATE());
   END;
```

Explanation: In the example above, builtin DATE() has the DATE attribute, but the procedure P does not.

IEL0795I W RETURNED EXPRESSION DOES NOT HAVE THE 'DATE' ATTRIBUTE BUT 'RETURNS' OPTION OF AN 'ENTRY' IN THIS BLOCK DOES.

EXPRESSION IN RETURN STATEMENT DOES NOT HAVE THE 'DATE' ATTRIBUTE BUT 'RETURNS' OPTION OF AN 'ENTRY' IN THIS BLOCK DOES. 'DATE' ATTRIBUTE WILL BE IGNORED IF THE INVALID COMBINATION OF 'RETURN' AND 'ENTRY' IS USED.

Example:

```
P: PROC(N)  RETURNS(CHAR(6) DATE);
E: ENTRY(N) RETURNS(CHAR(6));
   DCL N FIXED;
```

```
|       DCL C CHAR(6);
|       IF N=0 THEN RETURN('970704');
|              ELSE RETURN(C);
|         .
|         .
|         .
|       END;
```

| **Explanation:** In the example above, both RETURN statements will be flagged
| with this message.

| **IEL0796I W RETURNED EXPRESSION HAS THE 'DATE' ATTRIBUTE BUT 'RETURNS'**
| **OPTION OF AN 'ENTRY' IN THIS BLOCK DOES NOT.**

| **EXPRESSION IN RETURN STATEMENT HAS THE 'DATE' ATTRIBUTE BUT**
| **'RETURNS' OPTION OF AN 'ENTRY' IN THIS BLOCK DOES NOT. 'DATE'**
| **ATTRIBUTE WILL BE IGNORED IF THE INVALID COMBINATION OF 'RETURN'**
| **AND 'ENTRY' IS USED.**

| **Example:**

```
|    P: PROC(N)  RETURNS(CHAR(6) DATE);
|    E: ENTRY(N) RETURNS(CHAR(6));
|       DCL N FIXED;
|       IF N=0 THEN RETURN(DATE());
|         .
|         .
|         .
|       END;
```

| **Explanation:** In the example above, the DATE() builtin has the DATE
| attribute but entry E does not.

| **IEL0899I U PL/I MILLENNIUM LANGUAGE EXTENSIONS (MLE) PRODUCT MUST BE**
| **INSTALLED AND ACCESSIBLE TO THE COMPILER TO USE**
| **RULES(LAXCOMMENT).**

| **THE PL/I MILLENNIUM LANGUAGE EXTENSIONS (MLE) PRODUCT MUST BE**
| **INSTALLED AND ACCESSIBLE TO THE COMPILER TO USE THE**
| **RULES(LAXCOMMENT) COMPILER OPTION. COMPILATION TERMINATED.**

| **Explanation:** The use of the RULES(LAXCOMMENT) compiler option requires
| that the VisualAge PL/I Millennium Language Extensions (MLE) product be
| installed and accessible to the PL/I compiler. This product provides Year
| 2000 support.

| **IEL0900I U PL/I MILLENNIUM LANGUAGE EXTENSIONS (MLE) PRODUCT MUST BE**
| **INSTALLED AND ACCESSIBLE TO THE COMPILER TO USE RESPECT(DATE).**

| **THE PL/I MILLENNIUM LANGUAGE EXTENSIONS (MLE) PRODUCT MUST BE**
| **INSTALLED AND ACCESSIBLE TO THE COMPILER TO USE THE**
| **RESPECT(DATE) COMPILER OPTION. COMPILATION TERMINATED.**

| **Explanation:** The use of the RESPECT(DATE) compiler option requires that
| the VisualAge Pl/I Millennium Language Extensions (MLE) product be
| installed and accessible to the PL/I compiler. This product provides Year
| 2000 support.

```
| IEL0901I U PL/I MILLENNIUM LANGUAGE EXTENSIONS (MLE) PRODUCT MUST BE
|           INSTALLED AND ACCESSIBLE TO THE COMPILER TO USE 'DAYS' OR
|           'DAYSTODATE'.


|           THE PL/I MILLENNIUM LANGUAGE EXTENSIONS (MLE) PRODUCT MUST BE
|           INSTALLED AND ACCESSIBLE TO THE COMPILER TO USE THE 'DAYS' OR
|           'DAYSTODATE' BUILTIN.    COMPILATION TERMINATED.
```

| **Explanation:** The use of the DAYS or DAYSTODATE builtins requires that the
| VisualAge PL/I Millennium Language Extensions (MLE) product be installed
| and accessible to the PL/I compiler. This product provides Year 2000
| support.

BIBLIOGRAPHY Bibliography

Subtopics:

BIBLIOGRAPHY.1 PL/I for MVS & VM Publications

- *Licensed Program Specifications*, GC26-3116
- *Installation and Customization under MVS*, SC26-3119
- *Compiler and Run-Time Migration Guide*, SC26-3118
- *Programming Guide*, SC26-3113
- *Language Reference*, SC26-3114
- *Reference Summary*, SX26-3821
- *Compile-Time Messages and Codes*, SC26-3229
- *Diagnosis Guide*, SC26-3149

| BIBLIOGRAPHY.2 PL/I VSE Publications

| *Fact Sheet*, GC26-8052

| *Licensed Program Specifications*, GC26-8055

| *Installation and Customization Guide*, SC26-8057

| *Migration Guide*, SC26-8056

| *Programming Guide*, SC26-8053

| *Language Reference*, SC26-8054

| *Reference Summary*, SX26-3836

| *Compile-Time Messages and Codes*, SC26-8059

| *Diagnosis Guide*, SC26-8058

BIBLIOGRAPHY.3 VisualAge PL/I Millennium Language Extensions for MVS & VM Publications

- *Licensed Program Specifications*, GC26-9323
- *Installation and Customization under MVS*, SC26-3119
- *PL/I Millennium Language Extensions Guide*, GC26-9324

| BIBLIOGRAPHY.4 VisualAge PL/I Millennium Language Extensions for VSE/ESA Publications

| ° *Licensed Program Specifications*, GC26-9418

| ° *Installation and Customization Guide*, SC26-8057

| ° *PL/I Millennium Language Extensions Guide*, GC26-9324

BIBLIOGRAPHY.5 Language Environment for MVS & VM Publications

- *Fact Sheet*, GC26-4785
- *Concepts Guide*, GC26-4786
- *Licensed Program Specifications*, GC26-4774
- *Installation and Customization under MVS*, SC26-4817
- *Programming Guide*, SC26-4818
- *Programming Reference*, SC26-3312
- *Debugging Guide and Run-Time Messages*, SC26-4829
- *Writing Interlanguage Communication Applications*, SC26-8351
- *Run-Time Migration Guide*, SC26-8232
- *Master Index*,SC26-3427

BIBLIOGRAPHY.6 OS/390 Language Environment Publications

- Concepts Guide, GC28-1945
- Programming Guide, SC28-1939
- Programming Reference, SC28-1940
- Customization, SC28-1941
- Debugging Guide and Run-Time Messages, SC28-1942
- Run-Time Migration Guide, SC28-1944
- Writing Interlanguage Applications, SC28-1943

| BIBLIOGRAPHY.7 Language Environment for VSE/ESA Publications

| *Fact Sheet*, GC33-6679

| *Concepts Guide*, GC33-6680

| *Licensed Program Specifications*, GC33-6683

| *Installation and Customization Guide*, SC33-6682

| *Programming Guide*, SC33-6684

| *Programming Reference*, SC33-6685

| *Debugging Guide and Run-Time Messages*, SC33-6681

| *Writing Interlanguage Communication Applications*, SC33-6686

| *Run-Time Migration Guide*, SC33-6687

| *C Run-Time Library Reference*, SC33-6689

| *C Run-Time Programming Guide*, SC33-6688

BIBLIOGRAPHY.8 VisualAge PL/I Enterprise (OS/2 and Windows)

- *Programming Guide*, GC26-9177
- *Language Reference*, GC26-9178
- *Messages and Codes*, GC26-9179
- *Building GUIs on OS/2*, GC26-9180

BIBLIOGRAPHY.9 Softcopy Publications

Online publications are distributed on CD-ROMs and can be ordered from Mechanicsburg through your IBM representative. PL/I books are distributed on the following collection kit:

- *MVS Collection Kit*, SK2T-0710
- *OS/390 Collection Kit*, SK2T-6700
- *VM Collection Kit*, SK2T-2067
- *VSE Collection Kit*, SK2T-0060
- *Messages & Codes Collection Kit*, SK2T-2068

INDEX Index

A

B

C

D

E

M

P

R

S

U

W

Y

BACK_1 We'd Like to Hear from You

PL/I
Millennium Language Extensions Guide

Publication No. GC26-9324-01

Please use one of the following ways to send us your comments about this book:

° Mail--Print and use the Readers' Comments form on the next page. To print the form, select **Print** or **Copy** from the **Services** pull-down menu. Enter *COMMENTS* as the topic to be printed or copied. Mail the completed form to:

IBM Corporation, Department W92/H3
P.O. Box 49023
San Jose, CA 95161-9023
U.S.A.

If you are sending the form from a country other than the United States, give it to your local IBM branch office or IBM representative for mailing.

° Fax--Print and use the Readers' Comments form on the next page and fax it to this U.S. number: 800-426-7773. To print the form, follow the instructions under "Mail."

° Electronic mail--Use the following network ID:

Internet: COMMENTS@VNET.IBM.COM

Be sure to include the following with your comments:

- Title and publication number of this book
- Your name, address, and telephone number if you would like a reply

Your comments should pertain only to the information in this book and the way the information is presented. To request additional publications, or

to comment on other IBM information or the function of IBM products, please give your comments to your IBM representative or to your IBM authorized remarketer.

IBM may use or distribute your comments without obligation.

COMMENTS Readers' Comments

PL/I
Millennium Language Extensions Guide

Publication No. GC26-9324-01

How satisfied are you with the information in this book?

Legend:
1 Very satisfied
2 Satisfied
3 Neutral
4 Dissatisfied
5 Very dissatisfied

Please circle the number that corresponds to the level of your satisfaction.

Technically accurate	1	2	3	4	5
Complete	1	2	3	4	5
Easy to find	1	2	3	4	5
Easy to understand	1	2	3	4	5
Well organized	1	2	3	4	5
Applicable to your tasks	1	2	3	4	5
Grammatically correct and consistent	1	2	3	4	5
Graphically well designed	1	2	3	4	5
Overall satisfaction	1	2	3	4	5

Please tell us how we can improve this book:

May we contact you to discuss your comments? Yes No

Name . . . . . _____
Company or Organization _____
Address . . . . . _____
_____
_____
Phone No. . . . . _____

IBM Library Server Print Preview

DOCNUM = GC26-9324-01
DATETIME = 05/27/98 01:56:56
BLDVERS = 1.2
TITLE = PL/I Millennium Language Extensions Guide
AUTHOR =

PATH = /home/webapps/epubs/htdocs/book
