Integer Representation

The use of integer data types in RPG code has increased, not only in procedure prototypes, but even creeping into the definitions of work fields. The emergence of integer data types may be generally attributed to two different factors. One major factor, of course, is the increasing use of interfaces to C, or Java, which necessitate using integer representations, since neither language supports fixed format decimal representations. And, the other factor is an increase in the number of system API's finding their way into RPG programs, where the use of integers as parameters has been a practice for a long number of years.

In past releases of OS/400, it was common to represent integer data as 4B 0 fields (4 byte binary integer). Many of the IBM API manuals still reference a 2-byte, or 4-byte binary representation, such as this example showing the List Job Log (QMHLJOBL) API required parameter group.

Table 1

Required Parameter Group:

1	Qualified user space name	Input	Char(20)
2	Format name	Input	Char(8)
3	Message selection information	Input	Char(*)
4	Size of message selection information	Input	Binary(4)
5	Format of message selection information	Input	Char(8)
6	Error code	I/O	Char(*)

If integers are used in a data structure, programmers should make sure the offset for an integer is aligned correctly. In general, subfields defined using length notation, developers can automatically integer or unsigned subfields by specifying the keyword ALIGN on the data structure. However, the ALIGN keyword is not allowed for a file information data structure (INFDS) or a program status data structure (PSDS).

2 bytes for 5-digit integer or unsigned subfields
4 bytes for 10-digit integer or unsigned subfields or 4-byte float subfields
8 bytes for 20-digit integer or unsigned subfields

The following RPG D spec example shows typical representations of integer variables, (a 4-byte binary integer).

```
0001.00 D binint          ds
0002.00 D   binfield                1      4b 0
0003.00 D bininteger      S                10i 0
```

Typically, an integer, as in either D spec above is analogous to one of the primitive integer types found in other languages such as C and Java, like the Java variable definitions of the integer type, illustrated below.

```
int         scale = 10;
int         maxLabelWidth = 0;
```

I have no objection to integers used in procedure prototypes, or any other place within an RPG application, but since I have had to address several problems where programmers were attempting to assign integer values to fixed decimal representations, it is apparent not everybody understands the nature of integer data types.

The example, Table 2, lists Java integer representations. The variable type of INT corresponds to the integer definition on the RPG D specs above. Note the minimum and maximum values each integer type may represent.

Table 2

type	size(bits)	default value	minimum value	maximum value
byte	8	0	-128	+127
short	16	0	-32768	+32767
int	32	0	-2147483648	+2147483647
long	64	0	-9223372036854775808	+9223372036854775807

Table 3 shows a list of the values consistent with the DB2 database representations of integer data types. Though IBM integer type terminology varies somewhat from the Java primitive type designations, where SMALLINT corresponds to short integer and BIGINT corresponds to long integer representation, the value representation is the same. (IBM does not support the byte type integer representation.)

Table 3

type	size(bits)	default value	minimum value	maximum value
SMALLINT	16	0	-32768	+32767
INTEGER	32	0	-2147483648	+2147483647
BIGINT	64	0	-9223372036854775808	+9223372036854775807

Referring to either table, you can see the numeric value represented by an integer (INT/INTEGER) data type can range from -2,147,483,648 to +2,147,483,647. Even though the integer length may be represented as 4 bytes (32 bits), the precision extends to ten digits. **A 4-byte integer may represent a much larger numeric value than a 4-byte fixed decimal field**.

I have had to revise ILE applications where a developer has used integer values for parameters in a procedure prototype, and after invoking the procedure, the returned integer value has been assigned to a fixed (packed or zoned) decimal field. In many instances there was no check of the value, or monitor block surrounding the assignment.

The attempt to add or set the value of a fixed decimal field from an integer value can lead to a decimal overflow error, ***receiver to small to represent value***. If the assignment operation is not nested in a monitor block, the application program will give the operator no option but to cancel the program. If the intent of a program operation is to move integer values into fixed decimal fields, steps should be taken to insure that the integer value does not exceed the minimum and maximum values for the fixed decimal field.

Sample 1

```
0021.00 D TotDiff         s               4B 0

0292.00    IF TotDiff > 99999 or TotDiff < -99999;
0293.00       Iddlyh = 99999;
0294.00    ELSE;
0295.00       Iddlyh = TotDiff;
0296.00    ENDIF;
```

The code in Sample 1 shows IDDLYH, a 5-digit, zero decimal position, field, assigned a value from the field TOTDIFF. The field, TOTDIFF, happens to be defined as an integer. The simple assignment IDDLYH = TOTDIFF is enough to lead to problems in the program.

The largest value you can represent in a 5-digit fixed decimal field (with no decimal positions) is 99,999. As you can see from the tables, a 4-byte integer may contain a value as large as 2,147,483,647, which cannot be represented in a 5-digit, fixed decimal field. The problem is obvious; you cannot put 10-lbs of fertilizer, in a 5-lb bag, (organic, or otherwise).To prevent the overflow error, test the integer value prior to assignment and avoid the situation that produces the overflow. Ideally, then code a routine within the application to report the overflow/exception.

Avoid the decimal overflow condition with program logic. If integers are used for calculations, or parameters, and the value of integer may be assigned to a fixed numeric field, either make sure the fixed decimal field is large enough to contain the integer value, or test the value of the integer against the minimum/maximum value of the receiving field prior to assigning the value.

(Note: There is a compiler option to tell the program to ignore the numeric overflow when running the program. But this option does not apply to results of calculations performed within expressions, thus you may still get overflow errors even when TRUNCNBR(*YES) was specified on the CRTBNDRPG command.)

Fields that are defined as binary-type fields do contain integers but RPG programs don't necessarily consider binary and integer representation to be synonymous. This is important! A binary field on a physical file will be represented as '9B 0', occupying the same 4-bytes as a '10I 0' integer. So what is the problem? The problem arises when reading a record with a binary field and trying to move it to an internal integer variable. A 'field mapping error' will be generated.

Compile Options

```
ctl-Opt DEBUG(*YES) OPTION(*SRCSTMT : *NODEBUGIO)
      DFTACTGRP(*NO) Main(CSG078RP) EXTBININT(*YES)
      ActGrp('DLYRPT') BndDir( 'QC2LE' );

     D MsgInfo         DS                  qualified
     D  BytesRtn                     10I 0
     D  BytesAvail                   10I 0
     D  Severity                     10I 0
     D  MsgID                         7A
     D  Type                          2A
     D  Key                           4A
     D                                7A
     D  CCSID_st                     10I 0
     D  CCSID                        10I 0
     D  DataLen                      10I 0
     D  DataAvail                    10I 0
     D  Data                       1024A
.
.
.
     D DEC                            4B 0
     D  BIN                           1    OVERLAY(DEC:2)
```

When is an integer not an integer? The answer, at least for IBM's Power i OS is: when it is a binary decimal. Binary-decimal format fields contain a sign (positive or negative) is in the leftmost bit of the field and the numeric value is in the remaining bits of the field. Positive numbers will have a zero in the sign bit; negative numbers are denoted by a one in the sign bit and are in twos complement form. Both of the data structures shown above contain integer representations. The 10I 0 subfield occupies the same number of bytes as does the 4B 0 subfield. It is confusing.

Adding to the confusion is the fact a DDS field defined as 4B 0 may contain a value in 10 digit precision, **the same as for signed integers: -2147483648 to +2147483647. However,** every input field read into the program in binary format is converted by the compiler to a decimal field. The length of 4 is assigned to a 2-byte binary field; a length of 9 is assigned to a 4-byte binary field. Because of the fields are treated as decimals, the

highest decimal value that can be assigned to a 2-byte binary field is 9999 (32767 is greater than 9999, right?) and the highest decimal value that can be assigned to a 4-byte binary field is 999 999 999.

This explains why trying to push an external binary decimal into an integer subfield causes a mapping error to occur. There is a way to avoid the mapping error—refer to the Compiler Options example. The EXTBININT keyword may be used to process externally-described fields with binary external format and zero decimal positions as if they had an external integer format. If the keyword is not specified or specified with a value of *NO, an externally-described binary field is processed with an external binary format.

- Steve Croy