Here's a small routine (Function Coerce) to answer the implied problems with Visual Basic's Currency Data Type. The syntax is:

Coerce(VariableToFix@, 0|1|2|3|1000|1000000)

```
0       Dollar rounding with no significant digits in decimal
1       Tenths rounding with 1 significant decimal digit
2       Pennies rounding - 2 significant decimal digits
3       Mils rounding - 3 significant decimal digits
1000        Thousands of dollars
1000000   Millions of dollars
```

Since Currency is automatically 4 digits, this allows you to coerce the value to a specified number of digits with proper rounding. It will probably also work with other data types but for financial data, I strongly recommend the Currency data type to avoid binary conversion errors that might be present in other types such as Single and Double.

Each of us has convictions about what should and should not be. The answer to the problems inherent in the storage of decimal numbers in binary format can be addressed using BCD but BCD is slow and must still be manipulated to display or print documents that are acceptable in the business world. Such documents include checks, financial statements, payment schedules, and hundreds of others in which the amounts printed or displayed must reflect accuracy to the reader. If the amounts presented show errors when checked with a calculator or by human addition, subtraction, multiplication, and division then the entire document becomes suspect.

As a result, the real test of computer generated calculations rests in the programmer's ability to ensure that numbers and calculations come out correct to pennies in the case of American financial operations.

Some on the Visual Basic thread say that two decimal places of accuracy is all that's needed and that may be true but if I want to talk about 8 1/4% tax rates I must represent that number as .0825 (four decimal places) and if I want to stick with the Currency data type, it must be four places. I don't need to tell you what the State of California would do if taxes on a $15,000 automobile were collected at 8%. If I did it on a number of such transactions it would cost $37.50 on each one. Even worse would be calculating 10.8% interest on

a $500,000 mortgage but you get the point.

Microsoft says:

"Currency Data Type

A currency number is stored as an **8-byte, two's complement integer**, scaled by 10,000 to give a fixed-point number with 15 digits to the left of the decimal point and 4 digits to the right. This representation provides a range from -922337203685477.5808 to 922337203685477.5807.

The currency data type is extremely useful for calculations involving money, or for any fixed-point calculation where accuracy is particularly important."

In fact, many business application programmers use scaled integers to avoid problems and it looks like that is also what Microsoft is doing. The implication is that the accuracy is greater than Single or Double and as a **scaled integer** it may not be subject to the subtle errors that can creep into floating point calculations.

So, for the moment, let's assume that we need the accuracy (4 decimal digits) and that we do avoid the subtle errors that can occur when doing multiplication and division in floating point. With that assumption, it would still be nice to force (coerce) the amounts to 0, 1, 2, or 3 digits as required for calculations that required no greater degree of accuracy and that you can do with COERCE.BAS.

Even with this, you must still use care. Let's take a payroll check as an example.

1) Make all calculations for amount of pay, deductions, and similar information using 4 places of accuracy (Currency). Be particularly sure that multiplications and divisions are all done before the next step.

2) Coerce the calculated values to 2 places of accuracy (pennies) using the routine supplied here, someone else's routine, or one of your own devising.

3) Add total income, total deductions, and other sub total amounts using these penny rounded figures, then subtract as needed for Net Pay.

In other words, regardless of what Microsoft supplies you are still responsible for doing part of the work. That's a programmer's job. This much is certain - whatever MS provides some will fault it, some will work around it, and some will provide add-on functionality. COERCE.BAS may not be the fastest and it certainly isn't the most elegant but I think it will work for a large number of cases.

Bud Aaron - author "Visual Basic Programmer's Reference" Que Corp (due out in March 92 or earlier if possible)