QuantityWare Working Paper

Fixed-Point Arithmetic Calculations

Implementation of measurement standards using fixed-point arithmetic

Version History

Version	Date	Description
00	2009-09-22	Initial Version
01	2017-08-02	Editorially revised and confirmed
02	2021-09-24	Modern QW document style applied
03	2023-11-01	Editorially revised and confirmed

Contents

1. Introduction

The results of oil, gas and chemical product quantity value calculations should be accurate and reproducible, allowing their use in business transactions, within a heterogeneous company landscape and across system boundaries. Such calculations should provide the same values under different environments as operating systems, programming languages, run time libraries, etc.

Some advanced industry standards such as ASTM D1250-80 and ASTM D 1250-04/07/08/19 provide implementation instructions which must be followed step by step to fulfil such requirements.

Despite the wealth of information available, difficulties, however, still arise when implementing older standard versions in specific environments.

In this paper, targeted at technically oriented measurement experts and software architects, we detail some issues that arise from the utilization of different computing environments and how these different environments influence the comparability of results.

A good place to start is with the implementations of the superseded, historic, but still globally used ASTM D1250-80 measurement standard (and all its various mutations) which are all based on a fixed-point arithmetic implementation instruction.

We also provide a technical overview as to how QuantityWare was able to build an implementation of ASTM D1250-80 that provides identical results when comparing more the 60 million single calculations of our implementation with the results of an API C-code implementation.

The detailed documentation of this impressive result is documented in a separate working paper, targeted towards business decision makers and legal authorities and available in the the QuantityWare Knowledge base.

2. Implementation of Numbers and Functions

To represent numbers, different types of numerical data type implementations are used, and of course each of them has its own arithmetic.

For the implementation of current measurement standards three types of implementations are important:

- Fixed-point arithmetic
- Floating-point arithmetic single-precision
- Floating-point arithmetic double-precision

The accuracy and implementation details of any type depend on the CPU, arithmetic implementation, and software characteristics, such as the compiler, runtime libraries, etc.

In addition, there are different implementation standards of data types and the applied arithmetic rules available, which will not always produce the same results under different dependencies as noted above.

Decimal arithmetic (available in programming languages PL1, COBOL and SAP ABAP IV) has not yet been used in measurement standards calculations.

2.1. Floating-Point Arithmetic

Different standards for floating-point implementations are defined, but the most used is the IEEE Standard for Binary floating-point arithmetic (ANSI/IEEE Standard 754-1985) (see also http://en.wikipedia.org/wiki/IEEE_754 for details).

It can also be found under the original reference number IEC 559:1989.

Among all the varying implementations and formats of floating-point arithmetic, two are typically used in the implementation of oil & gas measurement standards:

- Single-Precision (32-bit)
- Double-Precision (64-bit)

Although accuracy is dependent on the number of bits, the number value range is huge, owing to the possibility of exponential representation.

In bulk-product orientated businesses, we do not have to worry about such ranges, or the position of a decimal point, as these details are handled by the floating-point data type implementation and the supported arithmetical operations.

Such implementation can be realized inside a processor (CPU), a floating-point unit (FPU) or in a coprocessor. Some older hardware does not support floating-point at all – in this case, special software programs must be used for emulation.

All the above already starts to indicate that different implementations will probably not necessarily provide exactly the same results. This includes the different data bases used in SAP implementations, as each database provider may show differing levels of accuracy when storing floating point numbers. However, the results from such systems are typically accurate enough with the relevant digits for bulk-product orientated business (i.e., we deal with large quantity values) and after rounding, which is also usually defined in measurement standards, we should obtain the same results. Nevertheless, accuracy is (from a theoretical viewpoint) limited and correct rounding is a crucial part of any measurement standard implementation. Some standards do not require a special implementation of floating-point arithmetic, some do. For example, ASTM D 1250-04 requires a 64-bit floating-point implementation. Some standards recommend calculation results rounding, whereas others require it (e.g., ASTM D 1250-04). By following a standards' implementation instructions, the same results should always be attained, independent of the hard- or software used.

Floating-point representations are easier to use than fixed-point representations, because they can handle a wider dynamic range of numeric representations and do not require programmers to specify the number of digits after the decimal point. However, there are also difficulties in using floating-point representations. These must be taken in consideration when applying arithmetic operations, such as the comparison of two floating-point numbers.

2.2. Fixed-Point Arithmetic

Historically, before the floating-point arithmetic was introduced, the only way to develop software calculations was to use fixed-point arithmetic, which is a form of limited precision arithmetic.

Fixed-point numbers have a fixed number of digits after the decimal point and sometimes also a fixed number of digits before the decimal point.

This arithmetic is very fast and easy to install.

However, a programmer has to “handle” the decimal point.

In this document, we will discuss the fixed-point arithmetic based on an Integer format, as it is used (e.g.) in the implementation guide for ASTM D1250-80 (FORTRAN implementations are described for this standard and a widely-used implementation in C is available).

The integer format may be numbers of 8, 16, 32 or 64 bit that have no decimal point and no defined decimal part. In addition, integer definitions can allow negative and positive values or only positive values (e.g., a counter or a pointer).

With measurement standards implementations, we must deal with two integer data types (we will not consider counters, pointers, indexes, etc):

Integer Data Type	Number of bits	Range (decimal)
Short	16	-32768 through 32767
Long	32	-2147483648 through 2147483647

As noted above, the decimal point does not exist in these integer formats and in order to represent numbers that are typical for the oil & gas business processes, the programmer is forced to (e.g.) multiply a density of 1021.4 by 1000000 and store these values in the available integer format. All other values must be translated in a similar fashion.

Example: a few common numbers of our business, aligned by the decimal point.

Parameter	Value	Value aligned	Value Integer
Density (kg/m³)	1021.4	1021.400000	1021400000
Density rel. to water	1.2345	0001.234500	0001234500
Temperature	103.15	0103.150000	0103150000
Therm. exp. coefficient	0.000789	0000.000789	0000000789
Volume corr. factor	1.23456	0001.234560	0001234560

As can be seen, we are already at the limit of 32-bit integer numbers' significant digits. In the previous examples, we have a "virtual decimal point" always in the same position that makes it easy to handle mathematical operations, but we also can see that we waste digits. In this scenario, if we perform some simple calculations (arithmetical operations), we will run into the following issues:

- We must manage decimal fractions
- We must take care of the numbers' ranges
- We must shift the numbers in order to work with the maximum precision

  (i.e.: during a multiplication, the number of decimals behave like nn.nn x n.nnn = nnn.nnnnn)
- At the end of the arithmetic operation chain, the programmer must bring the number back into the data type of the calling application.

Thus, the programmer must shift, scale, truncate and round almost all numbers during the calculation to get the job done – an overheard that can have serious qualitative consequences during programming and maintenance thereafter.

Differences in the implementation of the fixed-point arithmetic will lead to slightly different results, thus the programmer must depend on hardware and software to collaborate to obtain the same accurate results in different environments. Thus, without the following two major prerequisites, consistency is nigh-on impossible to achieve:

- The standard must provide an implementation guideline
- The programmer has to implement the procedures in a way to obtain the results as described in the standard in any environment

3. ASTM D 1250-80 Implementation in Different Programming Languages

3.1. Overview and Issues

The ASTM D1250 standard has a long history.

In 1980 ASTM D1250-80 was released, replacing the printed tables from 1952 with equations - this was a big step forward. However, at that time, the available hardware and software was not as mature as it is today, and the standard had to work on almost all platforms used within the industry. The only way to achieve this goal was to rely on fixed-point arithmetic operations for the provision of a generally compatible implementation procedure.

The most suitable programming language available at that time for such an implementation was FORTRAN.

Thus, FORTRAN and the fixed-point arithmetic (Integer data type) were used for the development of an abstract implementation procedure and an example implementation, which is described in:

API MPMS Chapter 11.1. -

Volume Correction Factors

Volume X – Background, Development and Program Documentation

The standard provides an extended description of the calculations, implementation procedures, test examples and coding in FORTRAN.

Additionally, the American Petroleum Institute (API) then developed source code of the functions that represented the former tables from 1952 in the programming language C. These function source codes can be integrated in business process systems.

At the beginning of the SAP IS-OIL development, SAP provided an interface [the QCI (Quantity Conversion Interface)] and a frame program for the API 1980 C codes. The frame program must be compiled and linked together with the API source codes. The resulting executable works with the SAP interface and uses the API table functions. The main challenge of this approach was, and still is, that an external executable does not fit into the SAP environment very well as serious security, performance and handling problems cannot be avoided. Thus, in S4/HANA systems, this approach is forbidden.

QuantityWare implemented the original standard (FORTRAN) into the SAP language (ABAP IV) by using the 32-bit integer format, but the comparison with the API C codes showed differences in the last relevant digits of the results (densities and volume correction factors) for approximately 1 in every 100 calculation results.

QuantityWare investigated this issue in detail. Two reasons for these differences could be defined:

- The C code implementation enhanced some of the original standard recommendations for the scaling of numbers to obtain more accurate (but not standard-conform) results.
- Differences in the implementation of the available C compilers and the SAP implementation of the fixed-point arithmetic can occur

Thus, the challenge we were presented with was how to handle the differences while obtaining the same results for both implementations. This will be explained in the following points.

3.2. Different Behaviour of Floating-Point Implementations

Even if operations such as the shifting and scaling of numbers are developed in a very careful manner for each implementation, it is not always possible to avoid different behaviour of an arithmetic operation, as different hardware- and operating- systems implement fixed-point arithmetic in different ways.

The following questions must be answered before development can begin:

- How will floating-point values be converted to fixed-point values and back?
- How will the result of an operation be provided? Via rounding or truncating?

  (i.e., how do we avoid aborts and dumps in the calculation process)
- Do we have to provide customizable tools for rounding?

Examples of these questions will be shown in the following sections.

3.2.1. Multiplication and Ranges

This is easy to understand and important! Let’s assume we must multiply two numbers:

e.g., C = A * B.

A = 812.34

B = 67.8904

We calculate the “normal” way and use any appropriate calculator, or do it all manually:

C = 55150.087536

Now let’s try this in integer format.

A) The first attempt:

Firstly, we must get rid of the decimal point by via multiplication:

A = A * 100	=	81234
B = B * 10000	=	678901

We are now ready for the **multiplication**:

C = A * B = 81234 *678901 = 55150087536

That looks fine and means C = 55150087536 / 1000000 = 55150.87536

Wonderful! **Where’s the problem?**

Remember the range of 32-bit floating-point numbers:

-2147483648 through 2147483647.

We have a result of 55150087536 and are therefore far **outside the ranges**.

Limit of the range:	**2147483647**
Result:	**55150087536**

Our system cannot perform this calculation. Dependent on the installation, we could get an abort without any information, or we can get the sort of abort that we are familiar with from our SAP system – a short dump and some information, helpful for the programmer, but not for the business user 😉

Although a good programmer could catch such an error (arithmetic exception), the calculation remains impossible.

Measurement standards developers must consider such system behavior.

So, we are stuck on 32-bit and still must present a solution. We must stay inside the supported mathematical ranges during the complete and typically complex calculation, thus we may have to accept a loss of accuracy.

B) The second attempt - we try it “smarter”:

We know that we cannot attain the maximum accuracy within our system limits, but we need a result that is as accurate as possible. There are ways in which to calculate the ranges and discover the most appropriate method however, in our example we will use an ad hoc approach to show how, in principle, the solution works:

A = 812.34

B = 67.8901

We round B (0.001 accuracy level):

A = 812.34

B = 67.890

Normalize A and B:

A = A * 100 = 81234

B = B * 1000 = 67890

We are ready for the multiplication:

C = A * B = 81234 * 67890 = 5514976260

Which means: C = 5514976260 / 100000 = 55149.76260

Unfortunately, the expected result (5514976260) **is still outside of our upper range limit** of 2147483647.

Limit of the range:	**2147483647**
Result:	**5514976260**

C) The third attempt, unfortunately, we must reduce the accuracy:

We must round the input by one more digit.

Of all possible methods, let's try the following:

A = 812.34	round to 0.1	-> 812.3
B = 67.8901	round to 0.001	-> 67.890

Normalize A and B:

A = A * 10	=	8123
B = B * 1000	=	67890

We are ready for the **multiplication**:

C = A * B = 8123 * 67890 = 551470470

Which means: C = 551470470 / 10000 = 55147.0470

Check the ranges:

Range: 2147483647

Result: 0551470470

That works 😉

To summarize:

The calculation with maximum accuracy was outside of our possible ranges.

We reduced the accuracy by rounding the input data and performed the calculation inside the supported ranges, but with less accuracy.

Question:

- Is our heuristic approach by rounding both input values the best one?
- Or could we do it better and archive a higher accuracy?

This very simple example shows how difficult it is to design a “simple” operation such as a “normal” multiplication required by our business, in fixed-point arithmetic.

Correct result:	**55150.087536**
Calculated result:	**55147.0470**

3.2.2. Division and Rounding

The division of two numbers is also affected by the limited accuracy of our data format; two major concerns are:

- Integer data type ranges
- Result rounding

Let's try the "simple" division 10 / 4.

I think that we can all agree that the result should be 2.5.

That's true - but not in the integer world.

Integer numbers have neither fractions nor decimal points. The arithmetic implementation must handle this and provide the expected result; thus, the result depends on the individual implementation of the integer arithmetic operation.

Based on the integer data type definition, there are two ways to perform such a division:

Truncate the result:

- 10 / 4 = 2.5 -> 2

Or round the result:

- 10 / 4 = 2.5 -> 3

We can manipulate that operation to obtain a higher accuracy by a simple scaling:

C = A / B

A = 10

B = 4

Now:

C * 10 = A * 10 / B

And we get:

C * 10 = 10 * 10 / 4 = 25

C * 10 = 25 stands for C = 2.5

Division results are commonly rounded via the truncation method. This is exactly how the ASTM D1250-80 implementation in FORTRAN was developed.

The developer of the FORTRAN standard implementation used the special implementation of the fixed-point arithmetic in FORTRAN.

Here are two examples for rounding to "1" in integer when the result has been truncated as in the FORTRAN, and some C implementations:

- 1.3 + 0.5 = 1.8 truncated to 1
- 1.6 + 0.5 = 2.1 truncated to 2

Now, what happens when the result has been rounded (as in SAP ABAP IV):

- 1.3 + 0.5 = 1.8 rounded to 2
- 1.6 + 0.5 = 2.1 rounded to 2

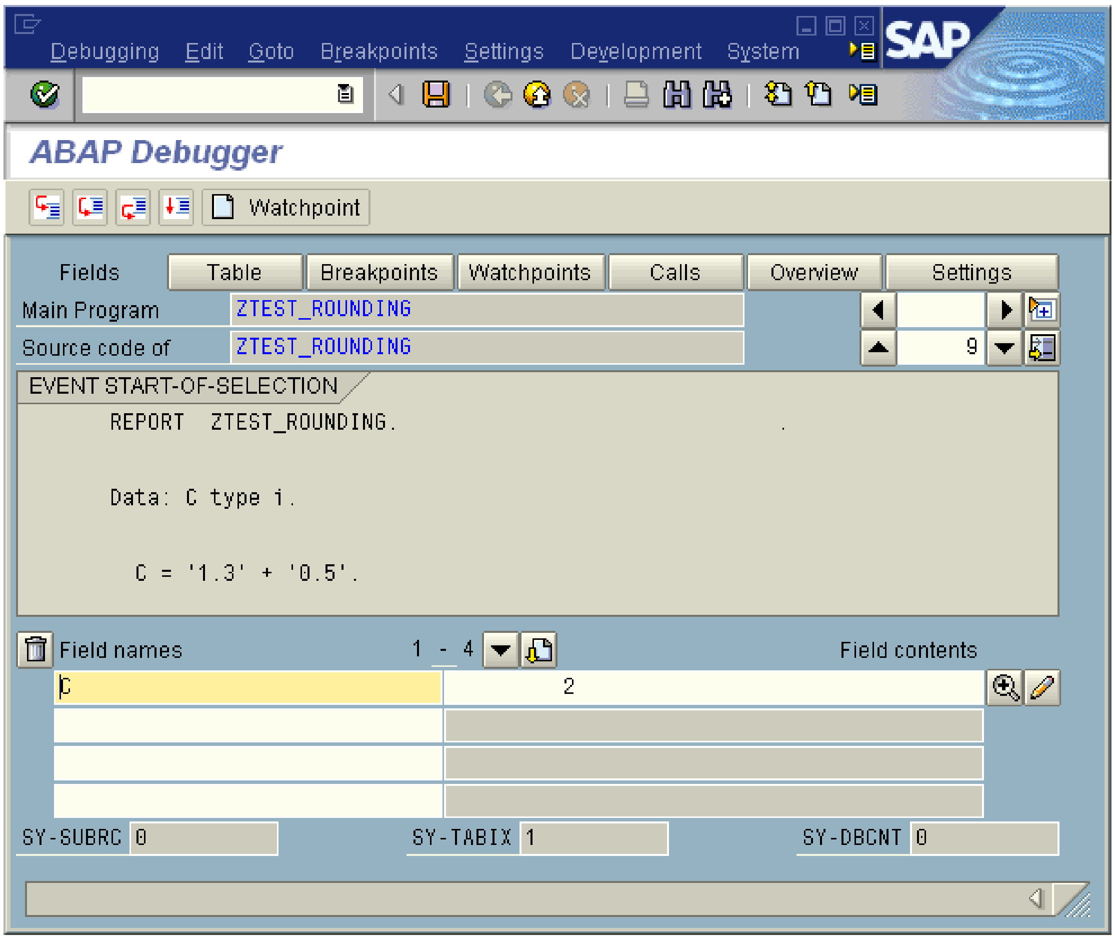
The results are accurate. Division result rounding is a good and widespread practice.

Rounding has been implemented in SAP ABAP IV and provides more accurate results than the truncation implemented in FORTRAN. It is possible for both languages to produce the same degree of accuracy and consistent results; however, this is generally dependent on the skills of the developer who writes the standard implementation.

Each language has a unique design and therefore a different implementation of arithmetic operations.

Let's look at two examples of two different implementations for the same calculation:

First example: SAP ABAP IV with results rounding:



Main Program ZTEST_ROUNDING
Source code of ZTEST_ROUNDING

EVENT START-OF-SELECTION

```
      REPORT  ZTEST_ROUNDING.

      Data: C type i.

        C = '1.3' + '0.5'.
```

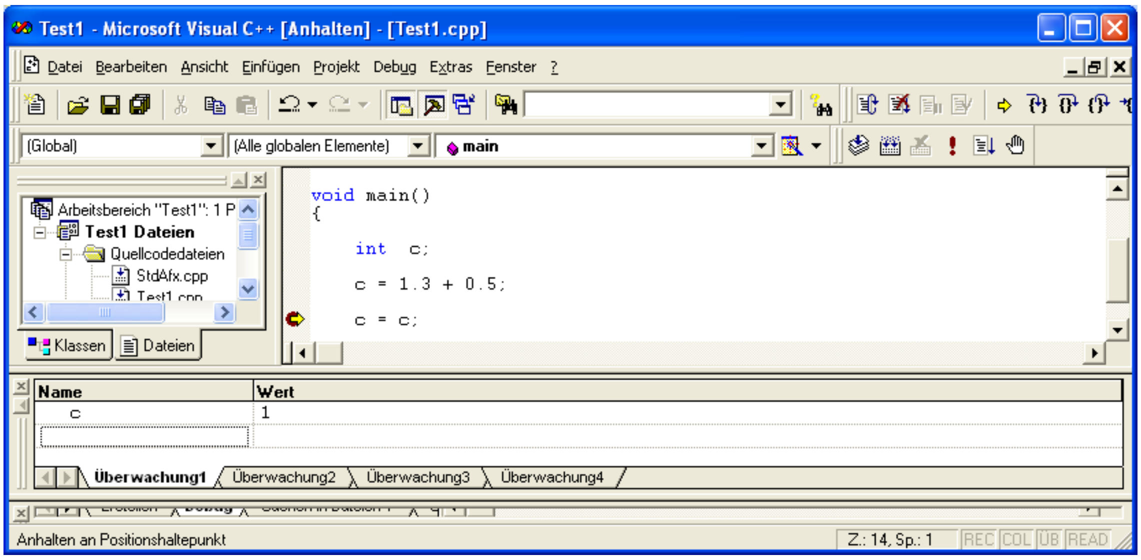
Field names 1 - 4	Field contents
C	2

SY-SUBRC 0   SY-TABIX 1   SY-DBCNT 0

What we see is: **1.3 + 0.5 = 1.8 rounded to 2**

This is a very good way to handle fractions of integer operations however it cannot be used for rounding as performed by ASTM D1250-80 using FORTRAN.

Second example: Microsoft C++ Visual Studio 6.0 (a different implementation of fixed-point arithmetic)


Test1 - Microsoft Visual C++ [Anhalten] - [Test1.cpp]
Datei Bearbeiten Ansicht Einfügen Projekt Debug Extras Fenster ?
(Global)
(Alle globalen Elemente)
main
Arbeitsbereich "Test1": 1 P
Test1 Dateien
Quellcodedateien
StdAfx.cpp
Test1.cpp
Klassen
Dateien
Überwachung1
Überwachung2
Überwachung3
Überwachung4
Anhalten an Positionshaltepunkt
Z.: 14, Sp.: 1
REC COL ÜB READ

```
void main()
{

    int  c;

    c = 1.3 + 0.5;

    c = c;
```

Name	Wert
c	1

We see: **1.3 + 0.5 = 1.8 truncated to 1**

This type of arithmetic is used for rounding in ASTM D1250-80 and of course for all other arithmetic operations.

The standard ASTM D1250-80 is based on a very special fixed-point arithmetic implementation. These procedures cannot be transferred in an identical manner to any other environment, thus it does not matter what environment is used, “even” C code implementations will not work in the same way, as they too are dependent on their fixed-point arithmetic implementation.

3.3. Consequences for Standard Implementations

The previous sections show some of the differences that must be dealt with when implementing fixed-point arithmetic operations.

There are two key issues to be considered when implementing a standard:

- How has the arithmetic been implemented in the system in which the standard has been developed, coupled with how are the standards' procedures making use of that implementation?
- How has the arithmetic been installed in the target system and how must it be used to achieve the same results required by the standard.

As previously shown, fixed-point arithmetic as a basis for measurement standards implementation can be executed in diverse ways, varying between programming languages and operating systems. Thus, a measurement standards' implementation can produce different calculation results when executed in different systems. The major points to consider are:

- An implementation of a specific measurement standard may have been developed under complex conditions within a special development system
- The procedures of the standard may make use of unique parts of special environments
- The target environment may have a slightly different system environment, leading to differing results between the different systems

For these reasons, we can safely say that standard selection and implementation is not as easy as it first appears!

In this working paper, we have considered only two special features of the fixed-point arithmetic. It is not the goal of this document to describe how a standard implementation should be executed, but to show a few mathematical issues as examples of the difficulties involved. As shown by the two examples, such differences are real and can have an impact on the implementation of a standard –leading to differing results. Such differences may not appear very often and may be small, but they must be expected and can lead to serious business disruption. When considering the financial and reputational values at stake, this is not acceptable for any implementation that calculates bulk goods quantity values.

3.4. QuantityWare Implementation of ASTM D1250-80

Among the many standards implemented by QuantityWare in SAP ABAP is the legacy ASTM D1250-80.

For quality assurance reasons, QuantityWare has also purchased a licensed version of the API c-codes (1980 and 2004 version).

We analyzed the given FORTRAN implementation guidelines and developed an implementation based on these instructions. Since we know in detail how the SAP language ABAP IV has been implemented, our task was to find a way to implement this standard in SAP ABAP IV while achieving exactly the same results as the other available implementations, where the historic C implementation provided by the API was the “de facto” industry standard.

The results of our effort are satisfactory.

The first comparison we ran was across all petroleum tables, over the full supported ranges, between our implementation and the API C-code implementation. More than 60 million calculations were performed. We found approximately 60 0000 differences for the volume correction factors at the fifth decimal place when compared to the original implementation.

We analyzed all differences step-by-step and found out that our implementation exactly reflected the FORTRAN standard. Eventually, the differences could be explained by advanced implementation procedures used within the c-code solution – owing to fixed point arithmetic issues like those described previously in this paper.

We also found out that in the case of such differences, the FORTRAN standard did not achieve the best possible ‘physical’ accuracy whereas the API C-codes did. Thus, the implementation method for the ASTM D1250-80 VCF calculations is software dependent, based on the fixed-point arithmetic limitations. The current ASTM D1250-04 VCF implementation procedure relies on double precision floating-point arithmetic and avoids this dependency.

We changed our implementation and ran all 60 million calculations again.

Our “tweaked” SAP ABAP IV implementation now provided the same results as the legacy API C-codes implementation (when compiled with a Microsoft 32-bit C++ compiler).

Since the API C codes are widely used within the industry, we decided to offer the modified version of our implementation as the default solution for the ASTM D1250-80 standard within our BCP product, thus supporting this “de facto” implementation.

Finally, we also decided to solve some reoccurring issues which the industry has when using the ASTM D1250-80 standard:

Many customers need to use the German rounding rules – these are supported in our standard BCP package.

The API C implementation of standard ASTM D1250-80 follows the defined validity ranges and does not allow calculations outside of those ranges (an error is returned). This is not practicable when related to bulk goods industries’ business requirements – we allow our customers to customize density and temperature ranges in a controlled manner, with a flag set in the corresponding material movement document.

4. Summary

In this paper, we focused on issues arising from fixed point arithmetic software implementations with a limited data value range and how these issues influence the programming of petroleum measurement implementations.

In detail, we used our findings and detailed analysis tools to build an SAP ABAP implementation of the ASTM D1250-80 API MPMS Chapter 11.1 volume correction factor calculation routines, which provides identical results when compared with a legacy API c-code implementation – for more than 60 million single calculations.

Since there are many operating system, compiler, and hardware combinations in use within the industry, there are a correspondingly large number of ASTM D1250-80 API MPMS Chapter 11.1 volume correction factor calculation routine implementations in use. As we have shown, differences between such implementations will occur.

ASTM D1250-04 has been available now since 2004. The oil and petrochemical industries are in a long transformation process towards this new and improved standard which eliminates fixed point arithmetic issues. Based on current observations, the transformation process could well take another 5 to 15 years (depending on e.g., legal requirements and national measurement standard adjustments). As ASTM D1250-80 is still in use, the findings documented in this paper can be relevant and helpful for organizations during this transition time. We hope that in some small way, we can also help accelerate the process for acceptance and business usage of modern, accurate and transparent measurement standards.

QuantityWare offers implementations of all available ASTM D1250 versions to ensure a smooth and individual transition process for our customers – we understand the reasons for prolonged usage of expired standards and strive to accommodate our customers' specific requirements within our standard solutions – BCP and BCG.

Legal Notices