The Year 2000: M and More

by Chris Bonnici

The Year 2000 (Y2K) problem is one of the most time- and labor-intensive problems the IT (Information Technology) industry has ever had to deal with. While it seems that the need for Y2K-compliant code is something quite recent, there is still a lot of old code in use today (from programming languages such as M, that have been around for a long time) that will require a major rewrite or conversion. The reason, of course, is that there are many systems now in operation that will not recognize year 2000 dates as the year 2000, but rather as the year 1900. And there are many reasons why something so obvious slipped by in such an unnoticed manner:

• no one realized that the code would last so long
• additional costs required to store 4-digit dates could not be justified
• existing YY libraries were used rather than modifying them to YYYY
• conversion was not a management priority

For some organizations, the term Year 2000 is very misleading because it gives the impression that the problems will come about in January 2000. This is far from correct. In some industries such as insurance, the majority of policies span one year, with some that extend beyond that. In banking, one can have dates that are 10 years into the future. Other industries will demonstrate similar situations. This means that the problems will appear before the turn of the century. For those organizations whose transactions are annual, the final date to commence Y2K project initiation was last October. This allows enough time for organizations to start testing the new date system during 1998 before the year 2000 dates are put into use.

Although this paper is directed towards software, the new century may effect anything that contains the words *program* and *using a date*. Computer systems fall under the reign of the IT department, but temperature regulation systems, telephone systems and others may also fall victim to Y2K. For example, equipment that normally generates a request for a service based on date, is prone to issue a false request.

In many organizations, CEOs and boards of directors consider computer departments to be a costly necessity. Top management does not seem to appreciate the fact that computer systems make their organizations tick. With Y2K, IT departments must convince management that additional funding is needed to fix something for which there are yet no visible symptoms and that for the time being, new developments must be put on hold. How does one explain this? Since in the majority of cases, the IT department will initiate the request for additional funding, it would be wise to point out that this particular problem will also effect areas of business (such as those mentioned above) which are not part of the computer department.

As far as software (which cannot be considered independently of hardware), the following stages are necessary when dealing with the Y2K issue:

I. Initial evaluation

Here one must identify the systems that need be checked for Y2K compliance. It must be understood that with large systems, the available resources may not be sufficient to make all available systems Y2K compliant. Thus it becomes important to prioritize systems. One big mistake is for the IT department to assume the sole responsibility for such a decision. Top management should be involved in selecting which systems are more important than others as these people should have a better overall knowledge of the company structure and as managers, are often held accountable for decisions effecting the bottom line.

II. Y2K compliance evaluation

Working down the prioritized list, teams would evaluate each system and ensure Y2K certification of all hardware, OS (operating systems) and attached devices as it would be useless to have functional software performing in a non-functional environment. For example, if a vendor points out that a given product will no longer be supported, some avenues may not be feasible. Evaluation teams must be on the alert for vendors that promise (without any form of binding agreement) that they will be looking into their product "soon." At this stage one must:

• Decide between a system conversion and a new system
• Decide changes in the hardware, OS and devices
• Estimate how long the task will take to complete. This is an area of project management and it may not be possible to produce on-the-spot estimates (though pilot testing may assist). Estimates that might be useful for this task are:

  • Estimates based on the number of date variables in programs (a parsing tool may be useful in this task)
  • Estimates of the lines of code that have to be looked into
  • Estimates based upon the number of debugged lines of code that a typical programmer can produce

A positive thing is that as the task progresses, experience from previous systems will help provide more reliable forecasts.

A point worth noting here is that evaluation (and all stages that follow) should be done on a separate machine having a mirrored backup of the original system. There have been reported cases of passwords expiring, software licenses expiring, erroneous transaction dates and other similar events occurring when the system date was changed on a live system.

III. Conversion commencement

Stage II above is an assessment of the system. If done properly one will gain an understanding of the relevant areas and can commence a strategy to tackle the problem. As stated earlier, investing in a new system for such development would be a wise decision. This paper will not delve into methodologies; there are numerous texts on how to deal with new developments and how to perform corrective maintenance.

A point worth noting here is that there is often an irresistible urge to improve the system. Experience has taught us that all improvements normally come at the cost of new bugs. Even if one focuses exclusively on the date issue, errors will be introduced. Given the possibility of tight schedules, one should avoid augmenting this with other (unnecessary) problems.

IV. Testing

This is the most important aspect of program development and will account for 50% or more of the time dedicated to resolving the Y2K problem. Even if some have skimmed this stage in the past, the penalty for doing it now is total disaster. Imagine, if due to poor testing, corrective patches contain bugs, and all of these bugs surface simultaneously in all systems when they go live.

With the complexity of software, even exhaustive testing may not identify all possible problems but the following conditions will help in evaluating as much of the Y2K problem as possible:

• Dates less than 2000
• Dates greater than 2000
• Dates that span the two centuries
• 31st December 1999
• 1st January 2000
• The First working day in 2000 (1st happens to be a Saturday)
• 29 Feb 2000 (2000 is a leap year)
• Fiscal year end
• Start of next fiscal year

Testing should cover all aspects of a system, namely input, processing and output.

a. Input

• User-entered data fields are enlarged. This may necessitate reformatting of the screen layouts
• Device Input. The dates are being passed correctly and no special marker is misinterpreted as a date
• OS input. The system date can be read and processed

b. Processing

• Calculations based on dates
• Sorting by date
• Date comparisons

c. Output

• Report output is correct: crammed reports may loose formatting or details (e.g., end of page)
• Data passed to devices is not interpreted erroneously
• OS output: a date passed to the OS will work as expected

The above summary gives a broad spectrum of the aspects that must be looked at. One of the major problems here is that computers are very good at processing data and can produce huge amounts of it. Peeking at the data can give the false impression that everything is OK when in reality it is not. It is important to distinguish between Visible System Failures and Invisible System Failures. In programming terminology, this approximates syntax errors vs. logic errors.

M

The fact that M has been around for so long means that one can find systems written long ago that have experienced a multitude of changes by an even greater troop of programmers, who may or may not be part of the organization today. This makes the task even more difficult. What follows is an attempt at classifying how dates are stored in M globals and offering suggestions that may aid others in arriving at a solution to the Y2K problem.

Dates in different systems can be stored in any one of the following formats:

• 4-digit year, human-readable format. Examples of such date formats are YYYYMMDD, DDMMYYYY, MMDDYYYY, with or without separators. These systems are probably already Y2K compliant (although only evaluation can prove this to be a fact) because in many cases, the underlying data structures determine the overall operational philosophy of the system.

• Integer format. M is blessed with routines that convert a human-readable date into an integer. The number one stands for 1st January 1841. The number 58074 is the M integer representation of 1st January 2000 (which in actual fact is the number of days that have passed since 31st December 1840). While date computations, comparisons and sorting should not be effected in such systems, input and output will have to be converted so that the year is YYYY rather than YY. This is because M systems may treat the date 01/01/00 to be 01/01/1900 rather than 01/01/2000. Some of the system-supplied routines may also have a characteristic that would not supply the first two-year digits if these were within the current century.

• 2-digit year, human-readable format. The most obvious problem area is with those dates where the year is represented simply as YY. Going from 99 to 00 could effect all date-related processes.

```
Y2KRTN=>;Date Conversion Routines - Chris Bonnici - Sept 1997
=>;You use these programs at your own risk
=>;
=>;The following assumptions are being made:
=>; Expands the two digit year of the passed date (DD/MM/YY or MM/DD/YY) parameter to
4 digit (DD/MM/YYYY or MM/DD/YYYY) - Chris Bonnici - Sept 1997
=>; If year < 50 then century = 19 else 20
=>; Null Dates will not generate an error but will return a null to calling module.
EXPDT(WHAT)=>N YY
=>Q:WHAT="" WHAT
=>S YY=$P(WHAT,"/",3)
=>Q $P(WHAT,"/",1,2)_"/"_$S(YY<50:"20",1:"19")_YY
=>;*** EOR ***
=>; Compacts a date from DD/MM/YYYY OR MM/DD/YYYY to DD/MM/YY or MM/DD/YY - - Chris
Bonnici - Sept 1997
=>; Routine can only handle dates in the region 1950-2049. If an error occurs user will
be alerted.
=>; Null Dates will not generate an error but will return a null to calling module.
COMPDT(WHAT)=>N YYYY
=>Q:WHAT="" WHAT
=>S YYYY=$P(WHAT,"/",3)
=>I YYYY<1950!(YYYY>2049) D  Q WHAT
=>.W !,"Cannot Compress ",WHAT
=>.R " Enter FULL date to pass to calling module ",WHAT
=>.Q
=>Q $P(WHAT,"/",1,2)_"/"_$E(YYYY,3,4)
```

Figure 1

There are two possible options:

a) A Windowing approach whereby all dates are passed through functions that will expand the date to Y2K and back on-the-fly. The algorithm would, for example, assume all dates greater than 50 pertain to the 20th century, with the range 0 - 50 belonging to the 21st.

The library in Figure 1 does just that. There are two functions that will expand and contract the dates on-the-fly. The advantages of this approach are that there is no need to expand dates, and there is no need to change the dates in globals. The Y2K task simplifies itself quite a bit.

The price for such an approach is that there is a 100 year date span limit. In a field such as a date of birth the presented solution will not function while a "temporary solution" will probably complicate matters so much that it would be more worthwhile going for a full conversion. Another problem is that the use of such an intermediary library might effect overall system performance. Another point to note is that although it would be possible to sort on dates when this process is done in a scratch global, it is impossible to have globals whose key fields are YY dates naturally sorted (i.e., read these globals sequentially using the $O function).

b) A more permanent solution to the Y2K problem is the one-time conversion to 4- digit years (well at least until 9999). Such a task would necessitate more work than solution (a), but does provide a more sturdy and long-term solution. Here are some suggestions as to how this can be tackled in M:

• Change all human-readable dates in the database to the internal $H integer date formats using vendor-supplied routines. For example, Micronetics has %DI,

```
DTRTN=>;Date conversion library from $H to DD/MM/YYYY and back
=>;You use these programs at your own risk
=>;
TRIM(WHAT)=>;Removes Leading and Traiming spaces from the Parameter - ACB - Mar 1996
=>N I
=>S I=1 F  Q:$L(WHAT)=0  Q:$E(WHAT,I)'=" "  S WHAT=$E(WHAT,2,$L(WHAT)) ; remove lead-
ing spaces
=>S I=$L(WHAT) F  Q:$L(WHAT)=0  Q:$E(WHAT,I)'=" "  S WHAT=$E(WHAT,1,$L(WHAT)-1)
=>Q (WHAT)
=>;*** EOR ***
=>;Converts a Date from DD/MM/??YY format to $H format - Chris Bonnici - Sept 1996
=>;With 2 digit Years, the program expands the remaining digits because the %DI rou-
tine will treat 00 as 1900 and not as 2000.
=>;It is assumed that in 2 digit year dates, if the date is < 50 it belongs to the 21st
century, else to the 20th.
INTDT(WHAT)=>N %DN,%DS,%ER,YR
=>Q:$$TRIM(WHAT)=""!(+WHAT=0) WHAT
=>S YR=$P(WHAT,"/",3)
=>Q:$L(YR)<1 WHAT
=>S:$L(YR)=2 YR=$S(YR<50:"20",1:"19")_YR
=>S %DS=$P(WHAT,"/",2)_"/"_$P(WHAT,"/",1)_"/"_YR
=>D ^%DI
=>Q:$D(%ER) ""
=>Q %DN
=>;*** EOR ***
=>;Converts a Date from $H to DD/MM/YYYY format - Chris Bonnici - Sept 1996
=>;The conversion utility will not print the first two year digits if these are cur-
rent century. This has been catered for - Chris Bonnici - Oct 1997.
NORDT(%DN,LONG)∂N %DS,%DA,%ER,YR
=>Q:+%DN=0 ""
=>D 400^%DO
=>S YR=$P(%DS,"/",3)
=>I '$D(LONG) Q ($P(%DS,"/",2)_"/"_$P(%DS,"/",1)_"/"_$S($L(YR)>2:$E(YR,3,4),1:YR))
=>I $L(YR)=2 S %NP="" D ^%D S YR=$E($P(%DAT,"/",3),1,2)_$P(%DS,"/",3)
=>Q ($P(%DS,"/",2)_"/"_$P(%DS,"/",1)_"/"_YR)
```

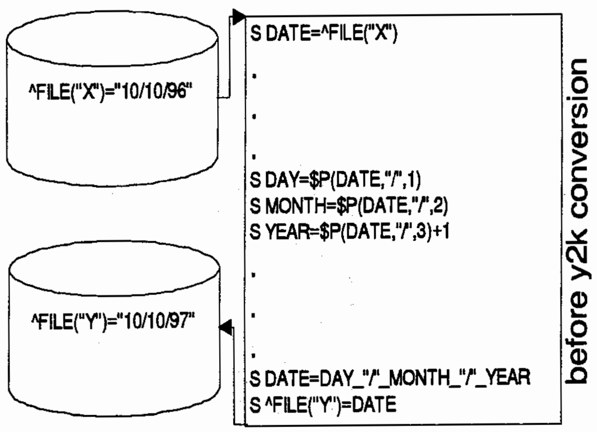
Figure 2

%DO and $ZDATE that will tackle this task.

The library in Figure 2 can help with such a conversion. The main problems associated with such an approach are that such dates cannot go below 1841. Another problem (or if you look at it from a security aspect an improvement) with $H dates is that dumping a global will not give you readable dates. Some additional processing may be necessary.

Routine NORDT is set so that it can return both CCYY and YY dates. This feature has been implemented to alleviate the task of changing reports during the first pass. Once conversion of the permanent data is over with, one can then focus exclusively on the reporting.

One minor benefit is that there will be an overall reduction in the size of globals. Also using a function means the amount of changes that have to be made to programs can also be reduced if dates within globals are converted to normal format as soon as they are read into the program and re-"re-integerized" exactly before they are written out to the file.



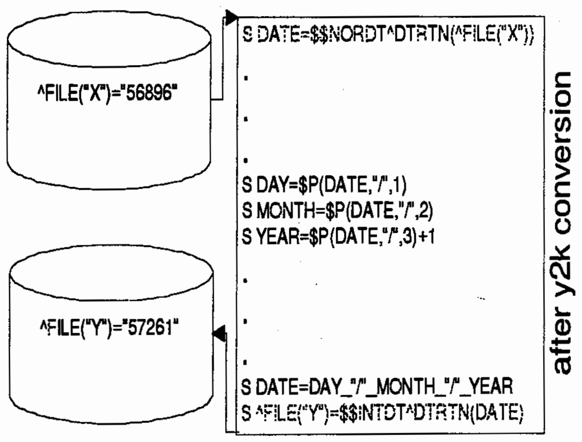
```
S DATE=^FILE("X")
.
.
.
S DAY=$P(DATE,"/",1)
S MONTH=$P(DATE,"/",2)
S YEAR=$P(DATE,"/",3)+1
.
.
.
S DATE=DAY_"/"_MONTH_"/"_YEAR
S ^FILE("Y")=DATE
```

Figure 3

In Figure 3 we have the original program using the functions presented here, one converts the dates to their "traditional" format as they are read.

Figure 4 shows that only two lines of the program need changes.

It should be emphasized that (as in this example) programs must not be effected by a four, rather than a two-digit year (for example using $E). If this is not the case, additional changes would be required. It should be clarified that input- and output-related issues will mean



```
S DATE=$$NORDT^DTRTN(^FILE("X"))
.
.
.
S DAY=$P(DATE,"/",1)
S MONTH=$P(DATE,"/",2)
S YEAR=$P(DATE,"/",3)+1
.
.
.
S DATE=DAY_"/"_MONTH_"/"_YEAR
S ^FILE("Y")=$$INTDT^DTRTN(DATE)
```

Figure 4

additional changes. This approach may result in inefficient programs, for example, when $H dates are read into a program and converted to human-readable format only to be converted back to $H again later in the same program. It would be simpler to avoid both conversions, but due to the urgency of Y2K, such matters should be shelved until programs are toured again for code optimization.

• Expand YY dates to YYYY. This approach is very similar to the Windowing approach discussed above, the only difference being that the changes are permanently written within globals rather than converted on-the-fly.

Besides problems relating to increased year length (mentioned previously), problems specific to this approach are:

- • Globals will increase in size. This will result in greater online / offline storage requirements
- • Globals might have to be remapped if the maximum string length of the global entry is exceeded

• Create your own $H routine. This is only necessary if the lower limit of the $H poses a problem. A library to perform such a task is shown in Figure 5.

The routine CHKDT in this library is a simple check to demonstrate that the algorithm converts dates accurately from integer to human-readable format. It hangs a bit when the date is a leap year (allowing one to note the date).

This can be seen as a more flexible implementation of the $H functions. Other than that what has been said for the standard date conversions applies here.

```
CBDTRTN=>;Chris Bonnici's Date Conversion Functions.
=>;You are using these programs at your own risk
=>;This library addresses the lower limitation of the $H functions.
=>;This conversion will cater for dates between October 15, 1582 and November 25,4046 - Chris Bonni-
ci - Sept 1997
=>;Source: Hewlett-Packard HP-85 Standard Pac Manual
=>;Modified algorithm to handle and return certain leap year dates correctly.
=>;Converts a date passed as DD/MM/??YY to an integer
INTDT(WHAT)=>N DD,MM,YYYY
=>S DD=$P(WHAT,"/",1),MM=$P(WHAT,"/",2),YYYY=$P(WHAT,"/",3)
=>I $L(YYYY)=2 S YYYY=$S(YYYY<50:"20",1:"19")_YYYY
=>I MM<3 S YYYY=YYYY-1,MM=MM+13
=>E  S MM=MM+1
=>Q $P(YYYY*365.25,".",1)-$P(YYYY/100,".",1)+$P(YYYY/400,".",1)+$P(MM*30.6001,".",1)+DD-478164
=>;*** EOR ***
=>;Converts a Integer Date to DD/MM/YYYY - Chris Bonnici - Sept 1997
NORDT(WHAT)=>N DD,MM,YYYY
=>S WHAT=WHAT+478164
=>S YYYY=$P((WHAT-121.5)/365.2425,".",1)
=>S MM=$P((WHAT-$P(YYYY*365.25,".",1)+$P(YYYY/100,".",1)-$P(YYYY/400,".",1))/30.6001,".",1)
=>S DD=WHAT-$P(YYYY*365.25,".",1)+$P(YYYY/100,".",1)-$P(YYYY/400,".",1)-$P(30.6001*MM,".",1)
=>I MM<14 S MM=MM-1
=>E  S MM=MM-13
=>I MM<3 S YYYY=YYYY+1
=>I MM=2 D
=>.I DD>29 S YYYY=YYYY-1
=>.I DD=30 S DD=28 Q
=>.S:DD=31 DD=$$LEAPYR
=>.Q
=>Q ($S($L(DD)=1:"0",1:"")_DD_"/"_$S($L(MM)=1:"0",1:"")_MM_"/"_YYYY)
=>;*** EOR ***
LEAPYR()=>Q:YYYY#4>0 28
=>I YYYY#100=0 Q:YYYY#400>0 28
=>Q 29
=>;*** EOR ***
=>;The loop below can be used to confirm that the two functions convert accurately, but be prepared
for a long wait until it reaches completion - Chris Bonnici - Sept 1997
CHKDT=>N I,DT,ERR
=>S ERR=0
=>F I=100000:1:1000000 S DT=$$NORDT^CBDTRTN(I) W !,I," ",DT D  Q:ERR
H:+$P(DT,"/",2)=2&(+$P(DT,"/",1)>28) 1
=>.I $$INTDT^CBDTRTN(DT)'=I W " error: ",$$INTDT^CBDTRTN(DT) S ERR=1 Q
=>.Q
=>Q
```

Figure 5

The author has used a COBOL version of these algorithms for many years without problems although he has never had to deal with dates that are less than 1950.

Some sources are claiming that Y2K will spell doom for the IT industry. Perhaps the opposite is true. We need the IT industry now, more than ever, to deal with the changes. And certainly change is nothing new. Changes in regulations and new business approaches are what make the business world go around.

We have always lived with deadlines (some of them being more of a deadline than others). This just happens to be of the unshiftable type. The same solid business practices that have brought us through other tough times will prevail here as well. Good project management, realistic time schedules, prioritization, and staff motivation, are all part of the job description that will make Y2K conversion a victorious task. Very soon we will be popping the end of century/new century champagne mildly cooled down to the right temperature. With proper planning and hard work, we will have something to celebrate. Besides, the experience that many of us will get won't be matched again for at least 100 years.

Some sites where you can find Y2K information (and which were referred to by the author for this article):
http://www.year2000.com
htttp://software.ibm.com/year2000

There are many sites with Y2K information. A search engine should provide much more information. **M**

Chris Bonnici is the MIS manager for The Malta Branch of Royal Insurance. He is editor of M Web Magazine *at http://www.mcenter.com/mwm/ and writes regularly for numerous paper and electronic media. He can be reached at chribonn@keyworld.net.*
His virtual office is at:
http://netopia.geocities.com/cbonnici.

The routines mentioned in this article can be downloaded from http://www.geocities.com/SiliconValley/7041/dload.html. Your thoughts and comments are welcome and may be used in another article in M Computing.

NOTE: The programs in this article assume that the conversion of 2-digit dates to 4 is based on the fact that all dates less than 50 belong to the 21st century with all others associated with the 20th century. This must not be taken as a generic rule and different systems must be first analyzed to check for the last date found in 19XX. If the conversion is of a permanent nature, this range assumption will only be a one-time process.

MDC—What's Your Fair Share?

As the MDC moves M Technology toward the millenium, it's looking for broader fiscal support from the M community. The M Technology Association and the Department of Veterans Affairs continue to be actively contributing partners, and other organizations are renewing or initiating a new interest in sponsoring MDC's vital work.

MDC Chair Art Smith notes in this issue's Questing column that the budget crunch is tougher than usual this year—so tough that MDC's "very continuation is in question."

MTA fully endorses the existence and work of MDC as vital for M Technology's future. MTA joins Art in encouraging you to consider what your "fair share" could be, and should be, in sustaining the MDC's—and M Technology's—vitality.

Don't miss this opportunity . . .

to place your ad in the **Annual Conference** issue of *M Computing* coming in June 1998.

Call MTA offices at 301-431-4070 for a contract and ad rate sheet. Hurry, space is limited!

Contracts are due in MTA offices by April 30. Ad copy is due by May 15th.