Time for a Change - The Year 2000 Problem

by George James

For reasons that have been well reported in the general computing press many computer systems will have problems coping with dates that are in the next century. This article addresses the issue of whether and to what extent M systems will suffer from this problem.

Will M applications suffer from the year 2000 problem?

Yes. Over the past few months we have examined many M applications for year 2000 compliance and, yes, there are systems that do have a problem. The bottom line concerning this problem is that there is nothing special about M applications that makes them immune from this problem. In general, any programmed system that is involved in date-related processing could exhibit a year 2000 problem.

How big is the problem?

In our inspection of M applications we have seen problems that range from trivial to serious. Here are some examples:

A database containing dates stored in YYMMDD format.

A date conversion algorithm that incorrectly treats the year 2000 as a normal 365-day year. N.B. The year 2000 is a leap year.

Input validation that contains a hard coded default of 19 for the century, whatever the current system date.

Sequential files for interfaces to third party systems with a YYMMDD format date as part of their file name.

A date conversion algorithm that incorrectly returns 01/01/20 instead of 01/01/00 for 1 January 2000.

In some cases the problems are very significant, requiring projects costing in excess of $250,000 and displacing other planned projects for up to a year. In other cases the problems are trivial and require nothing more than a few simple bug fixes which can be implemented as part of normal system maintenance.

The Corned Beef Story

In 1995, a well-known British supermarket ordered a consignment of long life corned beef. On the same day that the consignment was delivered to their depot the automated stock control systems flagged the whole lot for disposal because it was past its "sell by" date. The expiration date was actually in the year 2000. Fortunately the actual disposal process was manual and the error was spotted before 50 tons of corned beef were destroyed.

How big the problem is for your application will depend upon factors such as how well your application was engineered in the first place, how well it has been maintained, how tight the programming standards are and how well they have been enforced.

Size is also an important consideration. We have recently examined a very large M application (comprising over 20,000 routines) which, although very well engineered, is known to contain a few isolated problems. How can they be sure that they have found all the problems? The task of correcting the problems in this particular case is trivial. But without exhaustive testing of the whole application how can they be absolutely sure that all problems have been identified? This is why industry estimates for the cost of year 2000 compliance is often so high. Up to 80% of the cost is actually going to be for testing.

Do M implementations suffer from the year 2000 problem?

This is really a question that should be addressed directly to the implementors. If you have a software product that is supplied by a third party, the recommended practice is to contact them and obtain a compliance statement. Therefore, you should not only contact your M supplier but also your operating system supplier and the suppliers of all other software that you use.

Our own investigations and testing have so far not uncovered any major problems with current versions of the main M implementations.

It is clear that if you are running old versions of M, then your supplier may not be prepared to give any guarantees of compliance and may advise you to upgrade. If your business is dependent upon your M implementation working correctly through the year 2000 then this is good advice and should ensure that you get the reassurance and support that you will need.

Are M systems in a better or worse position than other technologies?

On balance I believe that M systems are in a much better position than systems developed using other technologies.

Given two similar applications written to the same software engineering standard, one written using M and one using some other comparable technology, the number of year 2000 problems within each application might be expected to be about the same.

A large number of traditional systems were historically implemented using a two-digit year to save disk space. By contrast many M applications used the same date format as the built-in $H date function and therefore avoid many of the problems altogether. For this reason alone many M applications will have significantly fewer problems.

Another factor that mitigates against many applications is that they do not need to accommodate a large date range. For example, MS-DOS only supports the date range 1980 through 2099. Because the origins of M are rooted in the medical field there has always been a requirement to support dates that are more that 100 years in the past (for patient date of birth). This has meant that, at least for medical applications, there has been tradition to use more than just two digits to represent the year. Even where $H is not used as the basis for the date format (for example in FileMan) the date format used is still able to support dates that span centuries.

Table 1 lists some examples of the effort involved in correcting typical year 2000 problems for an M-based application compared to an application written using traditional technology. The two major differences are that traditional technology consists of separate source, object and executable files, and that the databases are normally fixed length fields in a rigid record and file structure. Both of these differences make year 2000 projects more difficult for traditional technologies. For very old and large systems these can be major problems. By contrast, it is quite difficult to accidentally lose the source code for an M program, and database conversions by virtue of their dynamic nature are comparatively straightforward (although by the same token, accurate file layout / data dictionary documentation may be less forthcoming for an M database).

Despite all this, it should not just be assumed that your M application is OK. Many M applications use two-digit year date formats for input and output and for interfacing to third party applications. And even where $H style dates are used there is no guarantee that the date conversion algorithms are correct. The only way you can have any guarantee that your application will work is to test it.

Are there any M-specific problems?

Most of the year 2000 problems we have seen are independent of the underlying technology. These problems are many and varied and include problems with the user interface design, ambiguity in the presentation of dates on

Comparison of Year 2000 Corrections between M and other Applications.		
Type of Problem	M Applications	Other Applications
Error in date library function. large	Simple correction.	Simple correction but may require scale recompilation. Source code may not be available or previously compiled with an earlier version of the compiler etc.
Database conversion.	a) Identify all date fields instances within database. b) Write and execute database conversion program.	The solution will vary depending upon the technology used but typically: a) Export the whole database. b) Update the database schema. c) Import the whole database. d) Recompile all programs.
Error in ad-hoc date manipulation within a program.	Simple correction.	Simple correction.

Table 1

reports, errors in leap year calculation algorithms and interfaces to third party systems.

There are however, a small number of problems that are only to be found in M applications. For example, one feature of M which may well give rise to a number of problems is the way that the $ZD function and other vendor-supplied date functions operate.

$ZD is available in most M implementations and exhibits similar characteristics in each implementation. It is a function that converts a date in $H format into a date in human readable format. For dates between 1900 and 1999 it returns a date in MM/DD/YY format. For dates after 1999 it returns the date in MM/DD/CCYY format, i.e. the year includes the century. It should be stressed that this is a documented feature of the way that the $ZD function operates and not a bug. Unfortunately it is not an obvious feature and many programmers who have used $ZD in their applications will have been unaware of its behavior. The following code example, which was found in a real application, illustrates what can happen:

```
s date=$zd($h)
s yymmdd=$tr("123456","34/56/12",date)
```

When this code is run on 1 January 2000 the variable yymmdd will contain 200101 and not 000101 which was expected by the programmer.

The Credit Card Story

Many of the banks that issue Visa cards cannot process cards with an expiration date after 1999. Everyone knows that credit cards have an expiration date stamped on them in MM/YY format, so you would not expect such an obvious thing to cause a problem. Each issuing bank has developed its own card authorization system and some of them reject cards with an expiration year of 00. Until such time as Visa is confident that its cards can be processed, it is not issuing cards with an expiration date later than 12/99. As an incentive, any bank that cannot process Visa cards correctly after 1 March 1997 could be fined up to $100,000 by Visa.

Another example of how problems can arise with M concerns the treatment of leading zeros. If dates are represented in YY format, then performing arithmetic calculations when the year is between 00 and 09 (for example s yy=yy+1) will result in the loss of the leading zero.

A third class of problems with M applications relates to the conversion of dates from external format to internal format (e.g., MM/DD/CCYY to $H). Such conversions need to take account of leap years. Table 2 summarizes the rules for which years are a leap year. For most (non-medical) applications that deal with contemporary dates (say 1980 through 1997) a simple leap year algorithm is sufficient. The formula Y#4 is adequate to determine a leap year for all dates in this range. As it turns out, because the year 2000 is a leap year, this formula will actually work right up to 2099.

Most M applications, however, need to deal with a much greater date range (say 1850 through 1997). Because 1900 is an exception to the general leap year rule a slightly more complex leap year algorithm is required, typically using Y#4 and Y#100. This will result in correct dates for all years from 1601 through 1999. However, to work correctly for the year 2000 a third condition is required based on Y#400. Without this condition a date conversion algorithm will not correctly identify the year 2000 as a leap year.

Because of the need to handle dates prior to 1901 in medical applications, most M date algorithms contain both the Y#4 and Y#100 clause but do not necessarily contain the Y#400 clause. In some cases they do contain the Y#400 clause, but because it has never been executed, it has not been tested and does not work correctly. This is clearly an area of potential trouble for M applications. Testing for correct treatment of the year 2000 as a leap year should be part of every M application's year 2000 test plan.

Leap Year Rules

The following three rules define which years are a leap year.

1. Every fourth year is a leap year (e.g. 1988, 1992, 1996)

2. Except every hundredth year which is not a leap year (e.g. 1700, 1800, 1900)

3. Except every four hundredth year which is a leap year (e.g. 1600, 2000, 2400)

Table 2

What are the issues affecting VARs?

For VARs the year 2000 issue is magnified by the fact that they support multiple customers and because software is a key component of their business, the down side of being non-year 2000 compliant can be much greater. The following list indicates some of the areas where VARs may be affected by the year 2000 problem:

• **Support**
There is likely to be a large support hit at the end of the century as year-end processing coincides with a multitude of large and small year 2000 problems both with the VARs' own supplied software and with third party interfaces, operating systems, etc. This will be compounded by the fact that all support staff will either be on vacation or hung over.

• **Contractual Problems**
For customers who are not under a maintenance contract and perhaps purchased a system several years ago, do you have any liability if the software is not year 2000 compliant? Do you have any obligations to correct possible defects either before or after the event?

• **Upgrades**
If your applications are not currently year 2000 compliant and you need to upgrade your customer base, do you have the time and resources to upgrade all your customers between now and then? You may need to upgrade your customers sooner than you think if your application deals with future dates (for example, patient appointments may be made up to a year in advance).

• **Litigation Risk**
How confident are you that your applications do not have a critical year 2000 bug? If your application failed, could your business survive litigation from multiple customers? Can you afford not to perform year 2000 testing?

• **Internal Systems**
While the applications you supply are obviously of primary importance, have you examined your own internal systems? You should check your fault logging and tracking systems, configuration management systems, project planning and scheduling systems, software licensing systems, accounting and billing systems, etc.

• **Customer Assurance**
Are you prepared to respond to inquiries from your customer base concerning year 2000 compliance? Many organizations are starting to send out preliminary questionnaires to their suppliers requesting compliance information. Will you be able to reassure your customers? Will you be able to back up this reassurance with documentary evidence in the form of test plans and results?

• **Competitive Advantage**
If your competitors are able to claim year 2000 compliance and you cannot, then you may be at a competitive disadvantage. Conversely, if you have a good story to tell then you may have the advantage. Many organizations are now including year 2000 compliance as a mandatory requirement in their procurement process.

Conclusion

M applications are probably in much better shape than applications developed using traditional technology. However, this does not mean that the problem can be ignored. The fact that many M users are VARs means that the problem must be taken seriously. There are very real commercial consequences of not being able to respond well to their established and prospective customer bases.

Any M application could contain a year 2000 problem. The only way to know whether your application has a problem is to test it thoroughly. **M**

George James is president of George James Software, a London-based company that supplies re-engineering and configuration management tools and services to M users. The company is currently working with a number of organizations to help them address their year 2000 problems.

First there was MUMPS

then there was M

Now coming soon to an M shop near you

Nathan Wheeler & Co., Ltd.
THE UNAPROGRAMMERS
RATED M

M Programmers and Information Systems Consultants

Must be over 21 or have your manager's approval
(773) 862-1092
Fax: (773) 862-6117

Nathan Wheeler & Co., Ltd.
1351 N. Wolcott
2nd Floor
Chicago, IL 60622

We'll be back