How to Repair ... *That is the Question*

Addressing the Year 2000 Repair Decision-Making Process

BY PAUL ROSENHEIM

Many discussions focus on the evaluation of tools for the Year 2000 conversion, overlooking the various repair methods and their advantages and disadvantages.

WE'VE all been through the tools evaluation: Tool A finds 70 percent of the dates. Tool B finds 50 percent, but if you run it against the data files, its level goes up to 90 percent. Tool C can do automated code repair, but only for windowing, while tool D can repair code doing both windowing and expansion. We have (that's a generic "we") focused on tools, but haven't really spent much time evaluating the advantages and disadvantages of the various repair methods. There are at least eight major repair methods (windowing, expansion, date encapsulation, using century indicators, packing the date, employing date algorithms, using days from a point in time and using third-party date routine software), all with their associated advantages and disadvantages.

Although testing the repaired applications may generally take 40 percent to 60 percent of the entire project, the decisions made in the choice of repair methods can significantly impact this number. This article focuses on the repair decision-making process and specifically addresses date expansion and windowing, the two most popular remediation methods.

DATE EXPANSION AND WINDOWING DEFINED

Stated succinctly, date expansion means adding the two-digit century to the existing two-digit year thus ending up with a four-digit year. Some companies also consider adding a century indicator to be date expansion. However, for purposes of this article, expansion means adding a two-digit century and expanding the size of the date field and the corresponding data record.

There are two types of windowing: fixed century windowing and windowing with a "sliding" century window. Fixed century windowing means establishing a fixed "pivot" or "base" year. The year portion of the date field in the file (by file, we mean flat file, database, or table), screen, etc., is compared to the base year and depending on the result of the comparison, the century is identified. For example, assuming the base year is 75, years 00 through 74 will be treated as the 21st century and each year will be preceded by 20. Years 75 through 99 will be treated as the 20th century and the year will be preceded by 19.

Sliding century windowing involves specifying the number of years in the past and future relative to the system year (normally the current year) that encompass the century window. Assuming that the window is set at 27 past years and 72 future years and the current year is 1998, your application can accurately deal with dates between 1971 and 2070. The "sliding" portion takes effect in 1999, when the window becomes 1972 through 2071.

WHEN TO MAKE THE DECISION

Before deciding on the repair method or the combination of methods, you should first determine when to make the decision. Many companies make their decision before the project starts for a variety of reasons, with cost being the driving factor in many cases. Business decisions are best made when the most accurate information is available. In a Year 2000 project, this is after the Assessment or Impact Analysis phase, when detailed reports of date impact are available. These reports will give you a strong understanding of how your applications are date impacted, and by taking the following factors into consideration, you will be able to make a more educated decision.

CONSIDERATIONS

Selecting the appropriate repair method may on the surface seem to be a simple decision, but there are many factors that must be considered:

Cost: Based on the choice of the repair method, how much will it cost for you to repair and test the applications? Do you have the

internal staff necessary to repair and test the applications, and if not, can you hire more staff or will you have to use an outside service provider? Based on the repair method, what do you estimate the ongoing maintenance costs of the application to be? Are there any hidden costs (e.g., purchase additional DASD)?

Swiftness of repair: You have less than two years to complete the effort. Realistically, you should complete the project by the end of 1998. This will give you one year to hopefully find all the problems not found during your unit and integrated testing. Which repair method gives you the biggest likelihood of completing the project on time? If you have accounting applications that need repair, you should plan to complete them by third quarter 1998 so you can run a complete year with the repaired code before you close the books for 1999 in early 2000.

Application longevity: Is the application expected to be around for three years or more after the change of the century, or is it in the process of being (or will be) replaced shortly after the turn of the century? For short-lived applications, a less "clean" and less expensive approach might be acceptable.

Permanence of change: An expansion repair is permanent, whereas other repair methods may be less than permanent (e.g., windowing only works for a 100-year range).

Maintainability: Is the repair method chosen one that your maintenance staff will find easy to maintain after the application is repaired? What kind of standards will need to be implemented to ensure new programs and modifications to existing programs will follow the standards implemented during the repair process? To date, how successful have you been in implementing and enforcing standards?

Ease of implementation: Once the repairs are made, how easy or difficult will it be to implement the application? Do you need to build and manage bridges between your external data sources and your applications? Do you need to build and manage bridges between the repaired applications and other unrepaired applications? Can you implement the application in segments or does it require a "big bang" implementation because the number of shared files and programs accessing these shared files preclude a segmented implementation?

> Before deciding on the repair method or the combination of methods, you should first determine when to make the decision.

Database/file keys: You will need to expand files containing dates (with the year) in keys or indices only if you need to access the data in date sequence. How many files/databases/tables have dates in keys or indices that will require expansion, even if windowing is the selected repair method? Dates in non-key ID fields require additional analysis to determine if additional field expansion is necessary.

Bridges: Based on the repair method, will bridges be required? This can be a function of how the application is to be repaired and implemented in production. If the application must be segmented for repair and implementation, then temporary bridges may be required. If all programs accessing a given file/database/table can be repaired and implemented together in production, then these temporary bridges will not be required for production, but may be required for testing. However, many companies with large program portfolios are looking to do segmented repairs and implementation (as a means of reducing implementation risks), thus, there is a need for bridges.

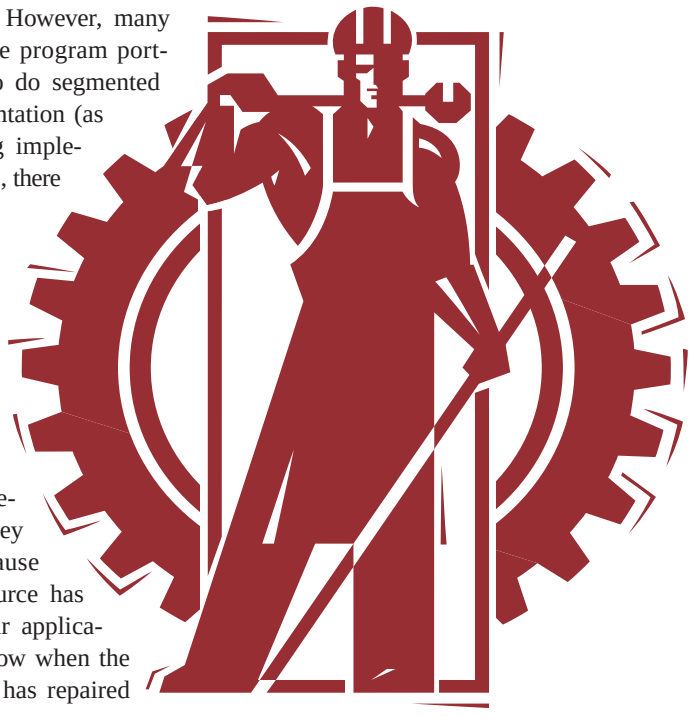
You must also take into account bridges needed for data coming from and going to data sources outside of your company. These bridges can require considerable management, especially if they are temporary because the external data source has not yet repaired their application. You need to know when the external data source has repaired their applications, so the temporary bridge can be eliminated. I know of one large insurance company that has so many external data sources to which it sends and receives data that it has assigned the management of bridges to one of their staff as a full-time job.

Performance: Will there be any potential system performance/throughput impacts? If your system is barely meeting its batch windows, additional statements added for windowing could cause problems.

Testing: Once the application has been repaired, how easy will it be to test? How many files will have to be expanded? How many databases/tables will have to be modified? How many conversion and bridge programs will have to be written?

DASD: Is there an impact on DASD storage needs? If you have large files with records containing many dates additional DASD may be required for storage and processing.

Assigning a high weight to maintainability may be valid for one application, but for another application, maintainability may have little or no relevance. For this reason, I haven't assigned a weight to any of these factors; I'll leave it to individual companies to develop their own weighting factors.



FIELD EXPANSION

If you're looking for a permanent solution, one that will be easy to maintain, relatively easy to implement, and easy to test, then field expansion is your solution. However, like all Y2K solutions, expansion does present some additional issues. File indices will work with the new expanded keys (assuming dates were part of the index). Many companies feel that if they have to expand some files, then why not expand all files? That is a valid question, and the answer depends on the number of files with dates in keys or indices. For example, if your application has 500 files and only 10 percent have dates in keys or indices, do you really want to convert the other 90 percent of the files that are unaffected? If the converse is true, it becomes a much easier decision.

The conversion itself will be a relatively automated one because most tools that do automated repairs support date expansion. The only decision you may have to make is "do I want to expand the date to display the year on screens and reports?" This is a business decision, not a technical one. Internal sorts should be unaffected (unless the language supports offset displacements in the internal sort). It will be easier to test a four-digit year, but more complex when bridges and data conversion problems are involved. You'll notice that I left cost out of the equation. Field expansion has been found to be the most expensive solution to implement. Why? There are several reasons.

Field expansion requires more DASD to store the additional two bytes of data (for every date in every record) that represent the century. You may be able to avoid additional DASD storage by packing the date or using binary representation. However, this may affect other factors, such as maintainability and bridges. By bridges, I mean permanent bridges that may be required for external data sources (sources outside the company with which you send and/or receive files). If you do use two additional bytes per date, do you have sufficient DASD to store the additional data? This is an area to involve your capacity planners. If you need more DASD, you've just increased your costs. Do you have the physical space to add more DASD? I spoke with some companies that physically had no room in their data center to even store additional DASD. For those companies, expansion of all files is not even a consideration. Field expansion also means converting the data in the files, which takes time and costs money (even if the tool being used produces the conversion programs).

Inter-group communications will be the key to reducing chances for error, which adds a level of complexity to the project and the ongoing maintenance. If a single base year is used, the communication problems are greatly reduced.

Another issue to address when field expansion is chosen is the use of temporary bridges as the company phases in the implementation of Year 2000-ready systems. While some files are being expanded, other applications continue to operate with unexpanded dates. When system A (which has its dates expanded) sends a file to system B (which has not yet been repaired) a program (bridge) must be built to convert the expanded file from system A back to its unexpanded format, thus, allowing it to be processed by system B. This bridge must stay in place until system B has its programs repaired to accept an expanded date field. Each bridge program takes time to code and test, even if the conversion tool generates the programs. Furthermore, bridge programs add additional processing to the normal processing cycle. The management of bridges (when to connect, when to disconnect) can be a full-time job for some companies. The batch bridges are relatively easy to deal with, but what about the real-time bridges needed to access databases/tables that are used in multiple applications? Think of what it will take to design and implement real-time bridges! Larger file record sizes mean more data to be processed, which in turn, could impact throughput and online system response time, although the impact will most likely be minimal.

Another issue to deal with when expanding files relates to offline historical data. How often do you have to recover this data? Do you convert it now or do you wait until it's needed? If you convert the files now, how much historical data do you have to convert? If you wait until the historical data is needed, there is the likelihood that someone will forget to convert the data. Remember, this is additional work that might have to be done and will add time and cost to your project.

External sorts will most likely be affected, even though you may not be sorting on the year. By adding additional positions for century, the displacement of every field that follows an expanded date has been changed. All external sorts that sort on fields that have had their displacement changed will have to be repaired.

When date fields are expanded, few program instructions have to be changed (for example with COBOL, MOVEs with displacement and literal MOVEs [e.g., MOVE 97 to …]) are two that come to mind. Many people think that expansion requires that only Copybooks be changed and the programs recompiled. In the majority of cases, that will not be sufficient. There is a vast amount of legacy code where file definitions were hardcoded in each program. All file definitions, Copybooks, and internal storage areas that use the expanded dates will have to be expanded. For example, if FILE-DATE is moved to WS-FILE-DATE and then WS-FILE-DATE is moved to WS-JUL-DATE, both WS-FILE-DATE and WS-JUL-DATE will have to be expanded to include a four-digit year. In addition, this requires that the format of the date field be known so that the proper field (the year portion) is expanded. This typically represents more lines of code that have to be changed than in other repair methods. Using automated tools greatly facilitates this process. However, if you decide not to expand the year on the screens and reports, considerable manual effort may be required to either prevent the tool from expanding these fields or to reverse what the tool has done.

Considering all the tasks that have to be performed when date fields are expanded, what at first glance may seem to be relatively easy to implement, is not so simple, once you begin to peel back the onion.

WINDOWING

Now that we've looked at the more expensive option, let's focus on the less expensive option, windowing, be it fixed or floating.

When windowing is the chosen method of repair, many of the factors that add to the cost of date field expansion are eliminated or greatly reduced. For example, since you're not expanding the date, your DASD

storage requirements are unaffected. However, if any of your files have dates, including the year, as keys, then the file key will have to be expanded, which includes converting the data in the files. This also applies to expansion but hopefully for far fewer files.

With windowing there should be fewer programs that have to be changed than with expansion. The only program instructions that are affected are compares of greater than or less than and arithmetic instructions (adds are usually unaffected). In fact, there may be many programs not requiring changes because these instructions, pertaining to dates, may not occur in every program. I have found situations where 25 percent to 35 percent of the total programs (that would have been changed with expansion) required no changes when windowing was used. The program conversion should for the most part be automated because most tools doing automated repairs support windowing. Since files, for the most part, are unaffected, testing may go faster because few files have to be converted, bridges may not be required, and relatively few program instructions may be affected.

Windowing can be a very cost-effective solution, but it, too, has its drawbacks. In particular, windowing only works for a 100-year range and is not a permanent solution. If you're repairing an application where dates can span more than 100 years, e.g., a life insurance system, then you should choose another approach.

'TWAS THE NIGHT BEFORE 2000

By Michael A. Klanica

'Twas the night before 2000 and all through the tower,
applications were failing, more by the hour.
The programs were running on the mainframe with care,
in hope that the millennium bug was not there.

The programmers were seated in front of their PCs,
while visions of blank paychecks danced in their heads.
With Amy in her office and I at my desk,
we had just settled down for a night with no rest.

When up on my screen there arose such a ding,
I sprang from my chair screaming... "I didn't touch a thing!"
Away from my computer I ran real quick,
tore open the drawer and picked up a stick.

I glared at the PC, evil and mean, then I realized ...
it's just a machine.
What to my wondering eyes should I see,
but a miniature window, and a message for me.

With my tired eyes, I gave a glance,
only hours left ... we don't have a chance!
More rapid than eagles the languages fell,
and we whistled, and shouted, and called them with a yell;

"Now, COBOL! now, NATURAL, Batch and On-Line!
Oh, FORTRAN! Oh SAS! Now CHORE went flat-line!
From the front of my face, to the face of the wall,
now bash away! bash away! bash away all!

As the team gathered together for one last try,
the word from management came... "Fix it or die!"
So they sat in their chairs, in the up-right position,
with a desk full of work, and a nasty disposition!

And then, in a dinging, I heard the speaker mention,
"Attention, the building, Attention."
As he tried to speak the next word,
the crashing of the mainframe is all we heard.

The programs were a mess, from start to end.
My screen was tarnished with an ugly abend.
The team assembled, into one huge pack,
we looked like hungry wolves, ready to attack.

Our eyes — how they twinkled!
Our fingers typed with a clank.
Fix Payroll we said,
because our paychecks were blank!

The sweat on my face was falling like rain,
while the coding of COBOL drove me insane!
The stump of a pencil I held tight in my hand,
I chewed nervously, hoping I would not get canned!

I coded some Windows and a Bridge too,
that took a program from version one to two.
I was tired, weak, and in a delusional state,
and I laughed when I saw it, in spite of fate.

A wink of his eye and a twist of his head,
soon let me know that bug was not dead!
It spoke not a word, but went straight to work,
crashing the remaining programs, then turned with a jerk.

I placed the cursor next to the bug,
and pressed the delete key, removing the little thug.
But I heard it exclaim, as I erased the line,
"Happy Millennium for now, cause I'll return in 9999!"

Michael A. Klanica recently finished an internship with U.S. Steel, a division of USX Corporation where he worked on the Year 2000 project. He is currently pursuing a double degree in Applied Mathematics and Applied Computer Science at Indiana University of Pennsylvania.

Another drawback is that both internal and external sorts will have to be repaired manually. However, there's good news; both IBM and Syncsort have added enhancements to their sort products that allow you to perform windowing in internal and external sorts using newly developed additional control cards.

Since you are adding code to the program to support windowing, there will be a performance impact. The extent of this impact depends on how much code is added and how the code is structured. Adding windowing code to the program may also increase the complexity of the program maintenance, especially if standard windowing routines are not used. By using standard windowing routines in your Year 2000 repairs, code maintainability will become much less of an issue.

Although you are not expanding your files, bridges may still be required. If some applications use date expansion while others use windowing, bridges will be needed. The same goes for external data sources that have selected date expansion.

The most important decision is what year to choose as the base year. It is critical to know your application(s) and data before choosing your base year. If, for example, you select 70 as your base year and you have data in your files prior to 1970, you will get erroneous results. You may find that you need to select many base years, because of the applications and the data. However, there should only be one base year for a given file. If you have more than one base year for different fields on a file, you will greatly increase the complexity and maintainability of your Year 2000 project.

Once the base year(s) is selected, this decision must be made known to everyone who comes in contact with the applications/files so that they can determine the changes to be made to their applications. The last thing you want is for one application to be using 70 as the base year while another application uses 80 as the base year for the same file. Inter-group communications will be the key to reducing chances for error, which adds a level of complexity to the project and the ongoing maintenance. If a single base year is used, the communication problems are greatly reduced. If multiple base years have to be used and if multiple computing platforms are involved, the communication problems are multiplied. How can you be sure that everyone is aware of which base year(s) to use? That is a real challenge and one that should not be dismissed lightly.

I hope this article provides the needed insight so that the proper repair decision(s) can be made. As you can see, each repair method has its own set of benefits and drawbacks. The best solution for your company is the one that best matches your organization's needs, talents, and budget. ts

Paul Rosenheim is a project manager with Paragon Computer Professionals, Inc. He has more than 30 years of IT experience in a broad range of technologies and industries. His current focus is in Paragon's Odyssey 2000 – Compliant Services business practice where he is responsible for all aspects of its Year 2000 methodology.