

AardWolf White Papers

STRATEGIES FOR A CHANGING WORLD



Copyright 1998, Marc S.A. Glasgow, aka The CyberPoet.
All Rights Reserved Under U.S. and International Law.

The Y2038C/C++ issue. Similar to the Y2k (Year 2000 Compliance), Y2038C/C++ is a date/time handling problem that represents a possible hurdle to continued computer operations in systems performing time/date calculations beyond January 18th, 2038 A.D. under C/C++. As we have seen with the problems relating to the Y2k preparation, the problem lies within the programming and the size of the data structures, and can be overcome by proper preparation. If Y2038C/C++ is not addressed, then during the course of calculating time_t beyond January 18th, 2038, time reverts to 1901 and subsequent calculations result in meaningless data.

Time (n) - a nonspatial continuum in which events occur in apparently irreversible succession from the past through the present to the future.

Trend: Y2038C/C++ may become more prevalent a problem than Y2k. Management will recognize the problem and be forced to address the issue in a series of steps, including preventative measures for current C/C++ programming projects. Within a five year horizon, with the freeing of Y2k personnel from Y2k issues, industry will overcome most Y2038C/C++ issues through changes in existing code, recompilation of edited code, movement to alternative programming languages (Java, Visual Basic) and redefinition of the time_t data structure under C/C++ by ANSI.

Quick Answers:

Q: Which operating systems are affected by the Y2038C/C++ issue?

A: While Y2038C/C++ issues can appear on programs running on various operating systems, only 32 bit UNIX systems will be affected at the operating system level, and problems arising from this should not affect users before 2038.

Q: Which applications are affected by the Y2038C/C++ issue?

A: C or C++ programs compiled in a 32 bit environment using the standard C and C++ time handling method known as time_t. This would include programs compiled under DOS, Windows3.1/95/98/NT, various 32 bit flavors of UNIX,

Q: What will happen if the Y2038C/C++ problem is triggered?

A: The calculation of time after January 18th, 2038 will continue from a starting point in 1901. This, in turn, could mean that the attempt to calculate the yield of a 44 year investment today would return a meaningless answer.

Firms whose software was written in C or C++ face a challenge with revamping their software to support calculations of dates beyond 2038. Although the Y2k issue is addressing the creation of 4 digit years and calculations to support the extended date format where necessary, the underpinnings of the C and C++ programming environment create a whole new slew of problems relating to how time is handled internally. The problem may already be manifesting itself for organizations which calculate long-term financial data using programs written in C and C++, such as retirement fund management, corporate investments, extended depreciation and mortgage loan handling.

We are advocating that all of our clients create a plan for evaluating the scope of the problem within their organization, quickly implement a training program (1 day) for their internal and contract support C and C++ programmers, integrate that training for new hires within your organization's programming staff, add a Y2038C/C++ clause in contracts with outside custom programming vendors, and then create a phased-implementation process to convert any necessary software. Fortunately, if the organization is still in control of the original C code (uncompiled), then most of the changes can normally be made in less than a single man-day per program; less than an hour in some cases.

The primary programming language in use for application building for the past decade has been C, and its object oriented version C++. C is defined as being a standardized language, in that its wording does not normally alter between computer platforms or operating systems within the core of any program written in it. Part of that standardization is the definition of time in a memory storage container (called a variable) named `time_t`. On most systems, `time_t` is defined to be 32 bits long, or 2,147,483,647 seconds (`time_t` being a long integer). On all systems, its' value is the number of seconds which have passed since January 1st, 1970.

The problem lies in the crux of the size and method of defining `time_t`, and all subsequent routines and variable which rely on the handling of `time_t`, such as `time_b`, and the MCF `CTime` class. Since `time_t` is a long integer, it is subject to treatment as one's compliment, a process for evaluating the variable as being capable of having a range including negative numbers. Thus, when overloaded as a 32 bit long int, the value becomes a negative and the date rolls over to 1901.

Although ANSI, the body which defines the standards of C and C++, will surely address this issue in the future, any time-handling/date-handling code written in C that is being created today will suffer from this malady if the issue is not addressed. This problem can affect systems ranging from Windows and DOS, through various flavors of UNIX (HP-UX 9.x & 10, AIX, Intel/SCO, Sun), and the Mac O/S (albeit in a different way, via ticks instead of `time_t`).

To define time in a larger and more robust method is a fairly straightforward concept when a program is first written. This is where implementing training for existing programming staffs (both internal and contract) creates a fast, tangible result for a very low training cost. Conversion of existing software in cases where the original C/C++ code is available for recompilation can be performed in a straight-forward way, and the cost of conversion is directly related to the number of routines in which `time_t` and its' related structures are used. The cost of conversion of machine language code (the assembled/compiled program) without access to the original C/C++ code can become prohibitive expensive, and I urge clients in such situations to evaluate whether use of planned replacement would be more cost effective than decompilation techniques and editing.

The problem also transcends the application realm, being integral to both various filing systems of certain operating systems which call `time_t` or related calls, and embedded software in the form of controllers, ROMs and Flash Memory. Any filing system problems will most likely be addressed long before a file modification date flips over into the past; embedded controllers, ROMs and to a lesser degree Flash Memory may cause unexpected behaviors in everything from traffic light controllers to check readers.

BOTTOM LINE: While this problem has been known to some top notch programmers, it is just starting to attract attention in general, and it will be years before the problem is addressed systematically by most firms. Companies engaged in long term financial handling will find that it is critical that these issues be addressed quickly and completely in order to ensure continued operations without flawed data. Although the problem may be solved by obsolescence of software and mandatory rewrites of mission critical code, the Y2k issue has brought to the forefront that robust software often outlives its' programmers. Finally, the problem can also be considered a data security/data integrity issue, as dates beyond the range of `time_t` can cause cascading failures if inserted erroneously or maliciously.