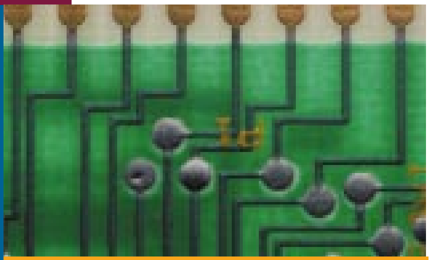

BY DUNCAN GILLINGWATER

Year 2000 Testing on an OS/390 Platform

By now most people will be moving into the testing phase of their Year 2000 projects, probably the longest and most laborious task of all. Without suitable tools and a dedicated testing platform, developers, QA and systems staff may be left stranded for days, unable to continue their work. As the deadline draws closer, it is vital that everyone look ahead to avoid the pitfalls and delays that lie in wait.

INTRODUCTION

The testing of applications for Year 2000 compliance falls into two categories: unit testing, which is the verification of individual units of program code, and integration testing, where Year 2000 compliant applications and systems software are combined to ensure that they can work in harmony. Performing the most obvious test involves restarting the system (IPL) and setting the machine clock forward to January 1, 2000. However, with so much automation and non-compliant software in place, a forward IPL could spell disaster as expiration dates are exceeded and system components fail to cope with a date beyond 1999.

Aware of the dangers, most organizations recognize the need for an isolated environment in which to safely perform the integration testing of their Year 2000 compliant software. Creating a specialized testing platform for OS/390 requires a dedicated machine or LPAR, which can be IPLed at will with whatever date and time is required.

This article presents specific issues that should be considered when establishing a Year 2000 testing platform and describes the significant dates that should be tested. It concludes that, while essential for integration testing, a dedicated system is not sufficient to enable the unit testing of application systems (which makes up the bulk of the work) to be carried out in a timely manner. Help is at hand, however, in the form of date simulation and other tools. This article concludes by explaining how this and other methods can be employed to ensure that work is finished on schedule.

CREATING A TEST PLATFORM

Establishing a Year 2000 testing platform requires, in most respects, exactly the same activities as establishing any separate testing system. Specific Year 2000 issues that should be considered are highlighted in this section.

The Year 2000 machine will not only provide a test bed for applications developed in-house but also for systems software and third-party products. Users will need to

Figure 1: Year 2000 Dates That Should Be Tested

Date	Reason
Sep. 9, 1999	Verify that expiration dates are not accidentally triggered (9/9/99)
Jan. 1, 2000	First day of the new Millennium
Feb. 29, 2000	Verify that the Leap Year is correctly processed
Mar. 1, 2000	Verify that the Leap Year is correctly processed
Dec. 31, 2000	Ensure that the full 366 days in 2000 are processed
Jan. 1, 2001	Ensure that the full 366 days in 2000 are processed
Feb. 29, 2004	Verify that the Leap Year is correctly processed
Mar. 1, 2004	Verify that the Leap Year is correctly processed

build and test their code with Year 2000 compliant products, which could mean:

- delays for new versions or fixes to support post 1999 dates
- a learning curve to understand and implement the modified features of the Year 2000-compliant product(s)
- changes to in-house code to support new interfaces

The integrity of the production system is vital; however, isolating the test system introduces further problems and delays, such as these:

- the restricted movement of data from production to test
- the inability to make suitable peripherals available to the test system during production hours

SIGNIFICANT TESTING DATES

When testing for Year 2000 compliance at both a system and functional unit level, processing on the dates shown in Figure 1 should be included. In addition to testing a static (fixed) date and time, it is important to check that applications perform correctly when certain date boundaries are crossed (rollover). See Figure 2.

Once tested against the essential Year 2000 dates applications should be regression tested to check that the existing functionality has not been affected. Suggested dates for regression testing are shown in Figure 3. There may be many other significant dates that are application-specific, for example, end of month and financial year-end. In addition to considering the processing that takes place on these days, testing may have to be performed for other dates that have some relationship with the critical dates mentioned. For example, to verify the renewal date for an insurance policy, testing should take place on a date that is effectively one year before the critical dates.

A DEDICATED LPAR IS NOT ENOUGH!

In Figure 1 a minimum of eight different dates were identified that form part of the required integration testing for each application. If rollover dates and significant application dates are included, this number is likely to reach 20.

While some applications have been specifically coded to utilize dates provided from other sources, most will use the current system date in some way, ranging from using it simply to record the date and time of processing or using it as a basis of the date-dependent processing. In an LPAR environment, the easiest way to change this date and to be sure that all applications will use the changed date is to re-IPL the system.

In theory, an IPL alone should take 15 minutes, but realistically it will take much longer because it requires the coordination and cooperation of the other users of the system. During the IPL the system is unavailable to all users and the new date chosen may not be suitable for the other tests being carried out.

In addition, while forward IPLs may be reasonably easy, a backward IPL may take much longer to arrange. Log files and checkpoint datasets will contain (unless cleared) dates that are later than the "current" date causing, for example, systems to hang while they wait for the date to catch up!

The problems are multiplied since integration testing accounts for only a fraction of the Year 2000 compliance work. The bulk of the work requires developers to test, fix and re-test individual units of code. Narrow testing windows are easily missed in a real-time environment, and it is unreasonable to expect everything to work the first time. Clearly the burden of IPLing becomes severely restrictive and will quickly jeopardize testing schedules. Fortunately a mechanism is available to assist users: date simulation.

DATE SIMULATION

Date simulation is the ability to intercept and modify the date/time calls made by an application. It offers major productivity gains by ensuring that different programmers can test different dates on the same machine, regardless of the current system date. The problems of rollover can be reduced since a simulated date can be pre-defined to start rolling at any required value.

Obtaining the Current Date and Time

An application program running in an OS/390 system can obtain the current date/time from a variety of sources:

- high-level languages (such as COBOL and PL/I) running in batch typically call a routine supplied by the programming language
- programs running under CICS will use CICS facilities (such as EXEC CICS ASKTIME and EIBDATE)
- assembler programs can use the TIME/STCKSYNC macros or issue the STCK (Store Clock) instruction

In general, regardless of the mechanism used by the application, the date/time is obtained in one of two ways:

1. using the OS/390 TIME and STCKSYNC macros, in which case the application will obtain the hardware date/time or the operating system date and time (local time)
2. using the STCK instruction, in which case the application will obtain the hardware date/time

Simulating Dates

The possible approaches available for simulating the date depend on the method used to obtain the date.

OS/390 Macros

Approaches that allow for application-specific dates when these are obtained via OS/390 macro calls include:

1. intercepting the OS/390 macro(s), which involves replacing an operating system address or value with one that invokes the date simulation routine
2. altering the compiled/assembled source to use a different method to obtain the time (for example, by using a different macro that offers the same interface) – this can provide a way to "hook" into a program without impacting the operating system
3. patching the code once it has been compiled/assembled to use a different method to obtain the time can also provide a way to "hook" into a program without impacting the operating system and is essentially the same method as above, without the need to compile/assemble the source

STCK Instruction

Two of the approaches described previously for OS/390 macros are easily adapted for application-specific dates obtained via the STCK instruction:

1. altering the compiled/assembled source to use a different method to obtain the time (for example, by

Figure 2: Date Boundaries That Should Be Tested

Date	Reason
Sep. 8, 1999 to Sep. 9, 1999	Watch for expiration
Dec. 31, 1999 to Jan. 1, 2000	Change from 1999 to 2000
Feb. 28, 2000 to Feb. 29, 2000	Ensure Leap day is recognized
Feb. 29, 2000 to Mar. 1, 2000	Ensure Leap day is recognized
Dec. 31, 2000 to Jan. 1, 2001	Check end of year
Feb. 28, 2004 to Feb. 29, 2004	Ensure Leap day is recognized

Figure 3: Sample Regression Testing Dates

Date	Reason
Dec. 31, 1998 – Jan. 1, 1999	Ensure year end correctly identified
Feb. 27, 1999 – Mar. 1, 1999	Ensure non-Leap year recognized

using a macro that offered a STCK-like interface)

2. patching the code once it has been compiled/assembled to use a different method to obtain the time

SIGNIFICANT FACTORS

Before selecting a method, it is important to understand and recognize the dangers and pitfalls.

- **System Hooks:** Components of the system should only be replaced with extreme care. Errors in a modified system component could have devastating effects.
- **Minimal Changes:** Source code changes should be avoided since this introduces a margin for error, especially when the date simulation is deactivated.
- **Consistency:** Dates should be trapped and modified consistently throughout an application or system. If multiple dates are used within an application then synchronization problems may occur.
- **Scope:** When using date simulation, you must be aware of all the components (applications/programs) that should and should not be changed. Not only does the production system need to be isolated from the Year 2000 system, but also different users of the Year 2000 testing system must be isolated from each other.

DATE SIMULATION ALONE IS NOT ENOUGH

The discussion so far has been based on the fact that an application accepts dates from an operating system call or programming language command. However, dates can be obtained from a variety of sources, for example:

- from file attributes
- from a date embedded in a file, transmission header or dataset name
- directly from another application's control block
- passed to it as a parameter in the PARM card or SYSIN, etc.
- data entered via a user interface

Clearly a forward IPL or date simulation will not directly affect many of the date sources identified above. Additionally, dates accepted from other sources generally require validation by the application (system dates are assumed to be always correct) and therefore require a different set of tests.

The tools that recreate and age data, combined with packages that automate keystrokes are beyond the scope of this article, but are no less important. Year 2000 project managers should give them proper consideration when planning their testing strategies.

THE LONG HAUL

This article has examined various problems and delays that await those responsible for Year 2000 testing on an OS/390 platform. To ensure compliance requires a dedicated and isolated platform on which integration tests can be safely performed. To set up an environment requires considerable planning and even so, will not address the bulk of the work (unit testing) that needs to be carried out. Date simulation offers major productivity gains, but even armed with a plethora of tools, IT departments still face a long journey to reach their goal. ts

U.K.-based NaSPA member Duncan Gillingwater has 10 years of mainframe experience. Having trained as a systems programmer, he joined Targetfour Ltd., Wokingham, Berkshire, UK, where he co-developed the PKZIP multi-platform data compression products for MVS and VSE and the Target2000 date simulation product. Duncan can be reached at dg@targetfour.com.