Yes, I am a Scifi reader. No, this article has nothing to do with Scifi. Zombie is an internet term. It refers to software which “takes control” of a computer and sends spam e-mails. Hopefully, you will never need the information presented here. However, if you do, it could minimize the impact of a potentially very annoying (although not really destructive) phenomena.

Like most of you, I know better than to download .exe modules from the internet. However, this was a company I have done business with in the past. It seemed a little strange that they had changed their coupon program to require a .exe file be run, but, as just stated, I have done business with them for about three years. I ran the .exe file.

Within minutes my e-mail package, Mozilla Thunderbird, was notifying me about a constant stream of mail. I quickly checked. The e-mails were all about returned e-mails that I had not sent. Something was clearly amiss. Fortunately, I was well caffeinated and thinking clearly. Time to spring into action.

Action 1 – Shut off the flow

I presume you have a firewall of one sort or another. Most firewalls have a “kill switch”. Mine does. I went into the firewall and turned off all traffic in and out of my computer. Now, with the problem temporarily resolved, I could turn my attention to getting rid of the problem. As an alternative, you can simply shut your computer off. That will not eliminate the problem. When you reboot your computer, the flow will start again, but by then you should be ready to proceed with turning your firewall off and the next step (below).

Action 2 – Getting rid of the problem

I had to go out for awhile. This afforded me a perfect time to run my Virus Checker, which typically takes quite awhile to run. When I returned, the Virus Checker had found nothing amiss. Not a surprise for two reasons. First, many Zombies are missed by Virus Checkers; second, contrary to constant entreaty by my Virus Checker software, I do not run the Update facility as often as I should.
Zombie Invasion

Not to worry. I was pretty certain I knew what had set the Zombie in action. I went to the folder where I had unloaded the .exe file to. I removed the .exe file. Then, with a bit of trepidation, I opened up the firewall, but kept it on the screen so I could visually see the traffic. For a few minutes, the flow of returned e-mails continued. I expected that. Then the flow began to slow, indicating that there were no new e-mails flowing out (which I could see by watching the firewall screen). After about ten minutes, the flow had become a trickle.

Some e-mail servers are really dedicated. For three days I still received occasional e-mails from servers that were still trying to deliver the Zombie’s e-mails. Then they stopped.

Lessons learned

Even “known” sources can transmit dangerous software. By acting quickly you can minimize the effect of Zombies.

The Boston Conference

The Boston Natural Conference, after a “road trip” in 2006 to Philadelphia, is back home in Boston. Dates are August 17-20 (Sunday through Wednesday); host hotel is the Hyatt Regency Cambridge. It should be unnecessary to extol the value of this conference. It is, after all, the only real “techie” Natural conference. Rather than my writing about the conference, here is an excerpt from the website (http://www.naturalconference.com/)

After reading the description of the Tracks, and taking in to account how difficult it is to cover all the sessions yourself, go to the website and print out the entire conference website. Then present the printout to management with a request for funding for several attendees.

(1) The Natural tracks which cover Natural on various platforms, including Natural 4, Natural 6, Natural Studio and SPoD.
(2) An Adabas track which covers important issues such as performance and tuning, backup and recovery, utilities, disaster recovery, replication and DBA skills.
(3) The Natural subsystem track will include sessions in Natural Security, Predict, Construct, PAC, and Entire System Server.
(4) Connectivity and Web enablement that cover middleware and data integration issues, with an emphasis on the premier middleware software EntireX Communicator and XML technologies, decision support software and other web apps within an SOA framework.
(5) RoundTables and Best Practices seminars.
More on Dates

Last issue I endeavored, unsuccessfully, to cover any/all areas of interest with regard to dates. This issue I will try to cover all the topics I missed last issue. We will start with

Date Parameters

Last issue we discussed the parameters DTFORM and DF. This issue we will discuss DFOUT, DF-STACK, DFTITLE, and YSLW.

DFOUT

For want of a better reason, we will discuss the parameters alphabetically, hence we will start with DFOUT.

I will point out that I am rarely concerned with this parameter. Why? DFOUT controls the default display of Date Format variables for INPUT, DISPLAY, PRINT and WRITE statements. In almost all circumstances, if I am going to output a Date Format variable, I either use an edit mask, or, use a MOVE EDITED statement to create an alpha variable with the desired format.

There are actually two parameters named DFOUT, a profile parameter and a session parameter. As you might guess, the session parameter will override the profile parameter.

There are two potential values for DFOUT: S (the default) and I. The default S specifies that date variables should be displayed with a two digit year. The setting I indicates a four digit year and no delimiters. In addition, the delimiters (for S, not I) and the sequence will be controlled by the parameter DTFORM which we discussed last issue. Let’s look at some examples:

```plaintext
* THIS PROGRAM WILL DEMONSTRATE DFOUT.
* THIS WILL BE RUN WITH VARYING VALUES
* FOR DTFORM
*
DEFINE DATA LOCAL
1 #DATE (D) INIT <*DATX>
END-DEFINE
*
INCLUDE AATITLRR
INCLUDE AASETCR
*
SET GLOBALS DFOUT=S /* THE DEFAULT
*
WRITE 5T 'THE VALUE OF #DATE IS: ' #DATE
// 5T 'DTFORM IS SET TO I (INTERNATIONAL)'
// 5T 'DFOUT IS SET TO S (DEFAULT)'
END
```

And, our output:

```
------------------------ OUTPUT ------------------------
PAGE #  1                    DATE:    12-MAY-2008
PROGRAM: DFOUT01              LIBRARY: DATES
THE VALUE OF #DATE IS:  08-05-12
DTFORM IS SET TO I (INTERNATIONAL)
DFOUT IS SET TO S (DEFAULT)
```

As I mentioned last issue, I typically keep my desktop and laptop set to DTFORM = I. Hence, the output you see above, namely, YY-MM-DD.

By way of contrast, Here is the program with DFOUT set to I (DTFORM still I).

```plaintext
* THIS PROGRAM WILL DEMONSTRATE DFOUT.
* THIS WILL BE RUN WITH VARYING VALUES
* FOR DTFORM
*
DEFINE DATA LOCAL
1 #DATE (D) INIT <*DATX>
END-DEFINE
*
INCLUDE AATITLRR
INCLUDE AASETCR
*
SET GLOBALS DFOUT=I /* INTERNATIONAL
*
WRITE 5T 'THE VALUE OF #DATE IS: ' #DATE
// 5T 'DTFORM IS SET TO I (INTERNATIONAL)'
// 5T 'DFOUT IS ALSO SET TO I (INTERNATIONAL)'
*
END
```

```
------------------------ OUTPUT ------------------------
PAGE #  1                    DATE:    08-05-12
PROGRAM: DFOUT01              LIBRARY: DATES
THE VALUE OF #DATE IS:  08-05-12
DTFORM IS SET TO I (INTERNATIONAL)
DFOUT IS ALSO SET TO I (INTERNATIONAL)
```
And the rather different output.

Note that the date format is now YYYYMMDD, a format that is useful for sorting, but not, to my way of thinking, particularly useful for display. My eye/brain prefers delimiters, whether hyphens, slashes, or even dots.

I now exited Natural and went to the Configuration Utility (I was on my PC), where I changed DTFORM to U (USA). I then ran the above programs again.

Here is our output:

Note the difference between the output above and the output from DFOU01. The setting of DTFORM (U) caused the sequence to change to month-day-year, and the delimiter to change to slash from hyphen.

Next we have the same DTFORM, but a setting of I for DFOU.

Note that DFOU=I caused the year to be four positions and the delimiters to be absent. DTFORM controlled the sequence of month-day-year.

The Parameter DF (which we discussed last issue) overrides the specification for DFOU. This is shown in the program below. If you look above at the output from program DFOU01, you will see that the output includes the line:

**THE VALUE OF #DATE IS: 08-05-12**

The program below is identical to DFOU01, except, I have set DF=L in the FORMAT statement.

![Program output](image-url)

And our output:

Note that we now have a four digit year (as per DF) rather than a two digit year (as per DFOU).
DFSTACK

An appropriate parameter to discuss since the main article this issue concerns the Stack. In a very real sense, DFSTACK serves the same function regarding the Stack that DFOUT serves regarding the screen.

DFSTACK controls how date format variables are placed on the Stack. Thus, this parameter is relevant to STACK, RUN, and FETCH statements, all of which can place data on the Stack.

Two of the settings for DFSTACK are the same as for DFOUT.

A setting of S (the default) for DFSTACK indicates that the string should have a two digit year and delimiters as specified by DTFORM.

A setting of I for DFSTACK gets you a four digit year and no delimiters.

There is one more setting for DFSTACK that probably hearkens back to the year 2000. A setting of C is the same as a setting of S in terms of format. HOWEVER, if there is a change of century between the stacked data and the current century (when the Stack is "popped"), an error message results.

Here is a program that will be used to demonstrate two of the three settings:

```plaintext
* THIS PROGRAM SHOWS TWO OF THE VALUES FOR DFSTACK, *
* NAMELY I AND S. AFTER USING THE SET GLOBALS WITH *
* DFSTACK=I, COMMENT THIS OUT AND REPLACE WITH THE *
* DFSTACK=S. DO THE SAME WITH THE WRITE STATEMENT.
* DEFINE DATA LOCAL
1 #DATE (D) INIT <*DATX>
1 #DATE-IN (A8)
END-DEFINE
*
INCLUDE AATITLRR
INCLUDE AASETCR
*
SET GLOBALS DFSTACK=I
*
STACK TOP DATA #DATE
INPUT #DATE-IN
*
WRITE 5T 'WITH DFSTACK=I, #DATE-IN IS: ' #DATE-IN
END
```

And our output.

```
------------------------------- OUTPUT -----------------------------
PAGE # 1    DATE: 17-MAY-2008
PROGRAM: DFSTCK01    LIBRARY: DATES
WITH DFSTACK=I, #DATE-IN IS: 20080517
```

The century transition eight years ago resulted in all sorts of problems, especially for shops that were trying to “save” two bytes per date by only working with two digit dates. That proved to be false economy as such shops discovered a seemingly unending series of minor problems. The following option was a response to one such minor problem.

```
* THIS PROGRAM SHOWS WHAT RESULTS WITH DFSTACK=C
* WHEN THE STACKED CENTURY IS DIFFERENT THAN
* THE CURRENT CENTURY
* DEFINE DATA LOCAL
1 #DATE (D) INIT <'1898-02-14'>
1 #DATE-IN (A8)
END-DEFINE
*
INCLUDE AATITLRR
INCLUDE AASETCR
*
SET GLOBALS DFSTACK=C
*
STACK TOP DATA #DATE
INPUT #DATE-IN
*
WRITE 5T 'WITH DFSTACK=C, #DATE-IN IS: ' #DATE-IN
END
```

The resultant error message is shown below:

```
* NAT1130 Unintended century switch when stacking date string.
* TEXT: Unintended century switch when stacking date string.
* EXPL: DFSTACK=C has been specified, i.e. date strings are put on
the stack without the century information (Natural V22
mode). When recovering this information (under control of
a Sliding Window or the current century), an unintended
century switch would occur: the year is not within the
range of the sliding window or not in the current century.
ACTN: Correct error in program or specify DFSTACK=I.
```
DFTITLE

This is a parameter I never have occasion to use. As you have seen over the years, I employ a Copy-code member (AATTITLER, for Structured Mode, AA-TTITLRR, for Report Mode) to generate my own titles for reports. You will almost never see code in Inside Natural that employs a default title. DFTITLE pertains only to default titles. There are three settings for DFTITLE, S (the default), I, and L. In the following programs we will see the result of each setting. First, DFTITLE=S (default)

```
* THIS PROGRAM WILL DEMONSTRATE THE DEFAULT DATE
* USED FOR TITLES (WITHOUT A WRITE TITLE OVERRIDE)
* SET GLOBALS DFTITLE=S
  INCLUDE AASETCR
  WRITE / 5T 'ABOVE IS THE DATE WITH DFTITLE = S'
  5T 'WHICH IS THE DEFAULT'
END
```

Note the date is the default of YY-MM-DD.

```
* THIS PROGRAM WILL DEMONSTRATE THE DEFAULT DATE
* USED FOR TITLES (WITHOUT A WRITE TITLE OVERRIDE)
* SET GLOBALS DFTITLE=I
  INCLUDE AASETCR
  WRITE / 5T 'ABOVE IS THE DATE WITH DFTITLE = I'
END
```

Note that the date above is now YYYYMMDD. One last setting:

```
* THIS PROGRAM WILL DEMONSTRATE THE DEFAULT DATE
* USED FOR TITLES (WITHOUT A WRITE TITLE OVERRIDE)
* SET GLOBALS DFTITLE=L
  INCLUDE AASETCR
  WRITE / 5T 'ABOVE IS THE DATE WITH DFTITLE = L'
END
```

And our output:

```
PAGE     1                             2008-05-18  07:50:46
ABOVE IS THE DATE WITH DFTITLE = L
```

Note above that with DFTITLE=L, the date is now YYYY-MM-DD.

As noted above, DFTITLE only affects the date in a default title. In the program below, I have a WRITE TITLE override which includes the current date.

```
* THIS PROGRAM WILL DEMONSTRATE THE DEFAULT DATE
* USED FOR TITLES (WITH A WRITE TITLE OVERRIDE)
* SET GLOBALS DFTITLE=L
  INCLUDE AASETCR
  WRITE / 20T 'ABOVE IS THE DATE WITH DFTITLE = L'
  WRITE TITLE 'HERE IS THE DATE IN AN OVERRIDE TITLE==> *DATX
END
```

Here is our output:

```
HERE IS THE DATE IN AN OVERRIDE TITLE==> 08-05-18
ABOVE IS THE DATE WITH DFTITLE = L
```

Note that although DFTITLE is set to L (see above where the date is YYYY-MM-DD), the date in the override title is YY-MM-DD.
YSLW

For those of you who were not programming then, a historical (hysterical?) note. Starting in the early to mid 90’s, the Data Processing industry entered a period of utter panic. The year 2000 was approaching. For some reason, up to the onset of panic, analysts were still designing systems with two digit years. Why? I haven’t a clue. It is not as if 2000 managed to sneak up unnoticed.

Many software vendors added facilities to ease the transition to the year 2000. Software AG was no exception. YSLW is an implementation of a classic “sliding window” approach that many companies adopted to “solve” the Y2K “problem”.

An important note. YSLW has no effect on dates with four digit years. It was designed as a mechanism for systems with two digit years, to “ease” the transition to the year 2000. Unfortunately, as often happens with “temporary fixes”, there are still systems running with this “fix”.

Okay, on to the details of the mechanism. The parameter YSLW has a numeric value which ranges from 0 to 99. A value of 0 (which is the default), indicates that the sliding window is not being used.

To discuss the mechanism, we will assume a value for YSLW, namely forty five. YSLW always “works” on a sliding window which is a century in range. The low end of the century is derived by taking the current year, 2008, and subtracting the value of YSLW. Thus the low end of our range is 1963. The upper end, of course, is therefore 2063. A two digit value from 63-99 will be interpreted as beginning with 19; whereas a two digit value from 00-62 will be interpreted as beginning with 20.

The sliding window mechanism is utilized whenever an alpha variable with a two digit year component is moved to a Date Formatted variable. This occurs in scenarios such as:
1) A MOVE EDITED statement
2) Reading a value from the Stack
3) Reading data from a Map
4) With the IS (D) test
5) With the VAL function

We will take a look at several programs. First, I closed my copy of Natural and went to the Natural Configuration Utility where I changed YSLW to 45. Then I restarted Natural and ran the following program (note the YY is in the range 63-99).

```
* THIS PROGRAM WAS RUN AFTER CHANGING YSLW TO 45
* IN THE CONFIGURATION UTILITY.
* DEFINE DATA LOCAL
1 #ALPHA (A8) INIT <'75-08-22'>
1 #DATE (D)
END-DEFINE
* INCLUDE AASETCR
INCLUDE AATITLRR
*
MOVE EDITED #ALPHA TO #DATE (EM=YY-MM-DD)
WRITE 5T 'ORIGINAL #ALPHA IS: ' #ALPHA
// 5T '#DATE IS: ' #DATE (EM-DO'/DD'/'YYYY)
END
```

Here is our output:

```
********************  OUTPUT  **********************
 PAGE #   1                    DATE:    18-MAY-2008
 PROGRAM: YSLW01               LIBRARY: DATES
 ORIGINAL #ALPHA IS:  75-08-22
 #DATE IS:  08/22/1975

Note in our output, #DATE, when output as a four digit year, shows 1975, not 2075.
```

In the following program, the value for #ALPHA is in the range 00-62.

```
* THIS PROGRAM WAS RUN AFTER CHANGING YSLW TO 45
* IN THE CONFIGURATION UTILITY.
* DEFINE DATA LOCAL
1 #ALPHA (A8) INIT <'14-08-22'>
1 #DATE (D)
END-DEFINE
* INCLUDE AASETCR
INCLUDE AATITLRR
*
MOVE EDITED #ALPHA TO #DATE (EM-YY-MM-DD)
WRITE 5T 'ORIGINAL #ALPHA IS: ' #ALPHA
// 5T '#DATE IS: ' #DATE (EM-MM'/DD'/'YYYY)
END
```

Here is the expected output:

```
********************  OUTPUT  **********************
 PAGE #   1                    DATE:    18-MAY-2008
 PROGRAM: YSLW02               LIBRARY: DATES
 ORIGINAL #ALPHA IS:  14-08-22
 #DATE IS:  08/22/2014
```

November 1996 ❖ 7
Note above that #DATE is 2014, not 1914.

As a final example, we will enter a date on a Map:

* THIS PROGRAM WAS RUN AFTER CHANGING YSLW TO 45
* IN THE CONFIGURATION UTILITY.
* DEFINE DATA LOCAL
  1 #DATE (D)
END-DEFINE
* INCLUDE AASETCR
INCLUDE AATITLER
* INPUT 3/10 'ENTER THE DATE 89-10-07 ==>' #DATE
* WRITE ST '#DATE IS: ' #DATE (EM=MM'/'DD'/'YYYY)
* END

And the output.

--------------------------- OUTPUT ---------------------------
PAGE #: 1                        DATE: 18-MAY-2008
PROGRAM: YSLW03                LIBRARY: DATES
#DATE IS: 10/07/1989

Note that the 89 in #DATE has been interpreted as 1989.

Summary

There is quite a spectrum of parameters (profile and session) for Dates. Most of them can be "avoided" if you use Edit Masks. I tend to use such Edit Masks. The potential problems when relying on parameters outweigh (from my perspective) any advantages that can be attributed to their use.

Edit Mask fiasco

There is one topic that I managed to omit last issue regarding a very annoying “feature”. It is so annoying, Software AG added a COMPOPT (compiler option) to address the issue. But, we are getting ahead of ourselves. Let’s start with the basics.

I will assume you are all familiar with the MASK option of the IF statement. If not, here is a simple example which shows MASK tests for alpha values in an alpha field and for numeric values in an alpha field.

* THIS PROGRAM SHOWS BOTH ALPHA AND NUMERIC
* EDIT MASKS.
* DEFINE DATA LOCAL
  1 #ALPHA (A5) INIT <'ABCDE'>
  1 #NUMA (AS) INIT <'12345'>
END-DEFINE
* INCLUDE AASET
INCLUDE AATITLER
* WRITE ST 'WE WILL TEST #ALPHA, WHOSE VALUE IS ==> ' #ALPHA /
  IF #ALPHA = MASK (..'CD')
      WRITE ST 'YES, THERE IS CD IN POSITIONS 3 AND 4'
  ELSE
      WRITE ST 'NO, CD IS NOT IN POSITIONS 3 AND 4'
  END-IF
* IF #ALPHA = MASK (.CD')
      WRITE ST 'YES, THERE IS CD IN POSITIONS 2 AND 3'
  ELSE
      WRITE ST 'NO, CD IS NOT IN POSITIONS 2 AND 3'
  END-IF
* WRITE / '-' (50) /
* WRITE ST 'WE WILL TEST #NUMA, WHOSE VALUE IS ==> ' #NUMA /
  IF #NUMA = MASK (.30-40)
      WRITE ST 'YES, THERE IS A VALUE BETWEEN 30 AND 40'
          / 10T 'IN POSITIONS 3 AND 4'
  ELSE
      WRITE ST 'NO, THERE IS NOT A VALUE BETWEEN 30 AND 40'
          / 10T 'IN POSITIONS 3 AND 4'
  END-IF
* IF #NUMA = MASK (.30-40)
      WRITE ST 'YES, THERE IS A VALUE BETWEEN 30 AND 40'
          / 10T 'IN POSITIONS 2 AND 3'
  ELSE
      WRITE ST 'NO, THERE IS NOT A VALUE BETWEEN 30 AND 40'
          / 10T 'IN POSITIONS 2 AND 3'
  END-IF
* END
And here is our output:

```
PAGE #   1                    DATE:    MAY 15, 2008
PROGRAM: MASK01               LIBRARY: INSIDE
WE WILL TEST #ALPHA, WHOSE VALUE IS ==>  ABCDE
   YES, THERE IS CD IN POSITIONS 3 AND 4
   NO, CD IS NOT IN POSITIONS 2 AND 3
---------------------------------------------------------------------
WE WILL TEST #NUMA, WHOSE VALUE IS ==>  12345
   YES, THERE IS A VALUE BETWEEN 30 AND 40
       IN POSITIONS 3 AND 4
   NO, THERE IS NOT A VALUE BETWEEN 30 AND 40
       IN POSITIONS 2 AND 3
```

The MASK on #ALPHA is pretty straightforward. The dots indicate positions to be skipped, thus MASK (..CD') is looking for CD in positions three and four.

Numeric MASKs, which we will be playing with shortly, are a bit more interesting. A single number MASK, such as MASK (..4) is NOT looking for a four in the third position. Rather, it is looking for any integer in the range 0-4 in the third position. Indeed, I could also have coded MASK (..0-4) rather than MASK (..4). I find the former MASK to be more "informative". In case you are curious, if I did want to look for a 4 in the third position, I could either have coded MASK (..4-4), or, switched to alpha and coded MASK (..'4').

Now that we have mastered one position numeric MASKs, two position numeric MASKs should be simple to understand. In our example we had MASK (..30-40). This says we want to look at the positions three through four for a value between 30 and 40.

**On to Dates**

Suppose I put up a Map and ask the user to input a Date. Of course, if you read last issue, I would much prefer to offer the user a Calendar to enter their date, since no validation is required for a date so entered.

But, lets assume, for whatever reason, you are "stuck" with the user entering an eight position alpha field organized as MMDDYYYY, which is the "style" of date most commonly employed (albeit usually with delimiters like slashes or hyphens) in the U.S.

How should we go about “validating” our date? Realize we could do this the “hard way”. We could use a numeric MASK to test MM for the range 01-12. Suppose we pass that test. What do we do now? Basically one of three things. Based on the value of MM, we either test DD for 01-30, or 01-31, or, we go to a special routine to deal with February.

Natural, fortunately, has edit MASKs to help us. Although you will rarely see them used in the manner shown in the next program, it is quite edifying to see how they work individually.

```
DEFINE DATA LOCAL
  1 #DATE (A8)
  1 REDDEFINE #DATE
      2 #MM (A2)
      2 #DD (A2)
      2 #YYYY (A4)
END-DEFINE
*
INCLUDE AATITLER
INCLUDE AASETC
*
INPUT #DATE
*
WRITE 5T 'INPUT VALUE WAS: ' #DATE /
*
IF #MM = MASK (MM)
   WRITE 5T '#MM IS A VALID MONTH ==> ' #MM
ELSE
   WRITE 5T '#MM IS NOT A VALID MONTH ==> ' #MM
END-IF
*
IF #DD = MASK (DD)
   WRITE 5T '#DD IS A VALID DAY ==> ' #DD
ELSE
   WRITE 5T '#DD IS NOT A VALID DAY ==> ' #DD
END-IF
*
IF #YYYY = MASK (YYYY)
   WRITE 5T '#YYYY IS A VALID YEAR ==> ' #YYYY
ELSE
   WRITE 5T '#YYYY IS NOT A VALID YEAR ==> ' #YYYY
END-IF
*
END
```

And our output:

```
PAGE #   1                    DATE:    MAY 15, 2008
PROGRAM: MASK02               LIBRARY: INSIDE
INPUT VALUE WAS:  04152005
   #MM IS A VALID MONTH ==> 04
   #DD IS A VALID DAY ==> 15
   #YY IS A VALID YEAR ==> 2005
```

November 1996  ❖  9
There is actually a bit more to the discussion above. I ran the program again, however, before I did, I changed the system date to June 15 rather than May 15 (easy, since I was on my PC). Then I ran the program again.

What happened? Why is 31 not a valid day? The answer is rather interesting and will be important in what we will be discussing shortly.

If I have a MASK with DD, but not MM or YY (YYYY); Natural assumes the current month and the current year. In the example above, I had changed the current month to June (take a look at the Title, which shows a date of JUN 15, 2008). The DD test was a MASK with just DD, no Y’s or M’s. Natural assumed a month of June and a year of 2008. Since June has but thirty days, the test for DD with a value of 31 failed.

By way of contrast, here is a program with a MASK that includes both DD and MM (and YYYY). The input will be the same as for the run above.

The data for MASK03 (above) was the same as the data for the last run of MASK02 (just prior to MASK03); in particular, both had a value of 31 for DD and 03 for MM. In the MASK02 example, the MASK test for DD failed. In the absence of an MM entry for the MASK, Natural assumed a month of June, and there is no June 31st. By contrast, in MASK03, we had a MASK that contained both MM and DD. Natural checked to see that 31 was a valid day for March (MM=03), which it is. The IF test was found to be TRUE.

Tangent – MASK versus MOVE EDITED

The MOVE EDITED statement is often used to create a Date Format variable from an alpha variable. This is shown in the following program which also shows a little known fact about Date Format. If you have a Date Format variable, you can simply MOVE it to an alpha variable. The format of the resultant alpha string will be determined by DTFORM and DF.

```
DEFINE DATA LOCAL
  1 #DATE (D)
END-DEFINE

* THIS PROGRAM WILL MOVE EDITED AN ALPHA VARIABLE
* TO A DATE VARIABLE, THEN MOVE (NOT MOVE EDITED)
* THE DATE VARIABLE TO ANOTHER ALPHA VARIABLE.
*
DEFINE DATA LOCAL
  1 #ALPHA1 (A8)
  1 #ALPHA2 (A8)
END-DEFINE
*
INCLUDE AATITLER
INCLUDE AASET
*
INPUT 3/10 'ENTER A DATE IN FORMAT YYYYMMDD ==>' #ALPHA1
*
MOVE EDITED #ALPHA1 TO #DATE (EM=YYYYMMDD)
*
MOVE #DATE TO #ALPHA2
*
WRITE 5T 'RESULTANT VARIABLES' /
  5T '--------------------' /
  5T '=' #ALPHA1 /
  5T '=' #DATE /
  5T '=' #ALPHA2
*
END
```
Okay, let's discuss the output. #ALPHA1 is easy. Following the directions on my Map, I entered the date in the format YYYYMMDD. I then used the MOVE EDITED to create our date formatted variable, #DATE. In the absence of an edit mask, date variables are written out as specified by DTFORM (mine is set to I (international)), and DF (mine is set to S). The DTFORM setting is responsible for the ordering of the three components and the use of a hyphen (rather than a slash or period). The DF setting specifies a two digit year rather than a four digit year (see discussion last issue).

Finally, we have #ALPHA2. The simple MOVE (rather than a MOVE EDITED) of a Date formatted variable to an alpha variable is conducted according to the DTFORM and DF settings, and is hence the same as the display of #DATE.

A problem

Since I am letting my user (me) enter the date in alpha format, I should validate my input. Why? Suppose I enter an invalid date. I receive the following error message:

```
NAT1143: Input does not correspond to input edit mask.
```

It was a simple typo. I wanted to enter a day of 28, but accidentally entered a day of 38. Problem. The MOVE EDITED statement could not function correctly. Natural has “taken control”. My program has ended. It would be far better if I could, in some way, “test” #ALPHA1 before trying to use it with the MOVE EDITED statement.

There is such a statement in Natural. As we have seen above it is the humble IF statement. Here is a slight rewrite of our last program.

```
* THIS PROGRAM WILL MOVE EDITED AN ALPHA VARIABLE
* TO A DATE VARIABLE, THEN MOVE (NOT MOVE EDITED)
* THE DATE VARIABLE TO ANOTHER ALPHA VARIABLE.
*
DEFINE DATA LOCAL
1 #ALPHA1 (A8)
1 #ALPHA2 (A8)
1 #DATE (D)
END-DEFINE
*
INCLUDE AATITLER
INCLUDE AASETC
*
INPUT 3/10 'ENTER A DATE IN FORMAT YYYYMMDD ==>' #ALPHA1
*
IF #ALPHA1 = MASK (YYYYMMDD)
  IGNORE
ELSE
  REINPUT 'INPUT IS NOT A DATE IN PROPER FORMAT'
END-IF
*
MOVE EDITED #ALPHA1 TO #DATE (EM=YYYYMMDD)
*
MOVE #DATE TO #ALPHA2
*
WRITE 5T 'RESULTANT VARIABLES'
  / 5T '-------------------' /
  5T '=' #ALPHA1 /
  5T '=' #DATE /
  5T '=' #ALPHA2 /
*
END
```

And our eventual output (after a REINPUT).

```
#ALPHA1: 20061028
#DATE: 06-10-28
#ALPHA2: 06-10-28
```

As with the run of MASK09 that produced the NAT1143 error (above), I entered a DD value of 38. This time, however, I “intercepted” the error before trying to execute the MOVE EDITED statement. I “realized” my error upon receiving my REINPUT error message. I fixed the date, and the program proceeded as it was supposed to.
Okay, all is fine with the world. Well, not quite. I am going to run MASK10 again. HOWEVER, I will enter a date of 10661014 (Battle of Hastings). WHOOPS. I receive the NAT1143 again. What happened?

The MOVE EDITED statement is designed to work with a year range of 1582 through 2699. Any other year produces the dreaded NAT1143. HOWEVER, the default IF ...MASK YYYYMMDD accepts any year value from 0000 to 9999. Whoops. 1066 is accepted by the IF statement. It does not, however, work for the MOVE EDITED statement.

A Bad Solution

What to do? Suppose I have an employee file which contains a Date of Hire field. We know that there is no one in the file who was hired before 1977 (when our company was founded). There is a field on the input Map for new employees for Date of Hire. We keep track of time-of-service, so we want to store away the Date of Hire as a Date format field.

Someone was once “burned” by the statement IF ....MASK YYYYMMDD accepting a year of 3002 (a typo of 2002). Hence they decided to “beat the system”. As shown below, they changed the IF statement to:

```natural
IF HIRE-DATE = MASK (1950-2300MMDD)
  IGNORE
ELSE
  MOVE EDITED HIRE-DATE TO DDATE (EM=YYYYMMDD)
END-IF
```

Looks like a good workaround. And it would be. EXCEPT for the fact that without YY or YYYY in the edit mask, Natural assumes the current year. Suppose I have the following program:

```natural
* THIS PROGRAM WILL MOVE EDITED AN ALPHA VARIABLE
* TO A DATE VARIABLE, THEN MOVE (NOT MOVE EDITED)
* THE DATE VARIABLE TO ANOTHER ALPHA VARIABLE.
*
DEFINE DATA LOCAL
  1 #ALPHA1 (A8)
  1 #ALPHA2 (A8)
  1 #DATE (D)
END-DEFINE
*
* INCLUDE AATITLER
INCLUDE AASETC
*
```

Okay, time to try the program above. How about with 20070229? Whoops. The dreaded 1143 again. This is the year 2008, which is a leap year. Does #DATE1 pass the IF..MASK test? Absolutely. The year part is definitely within the range 1950-2300 (does the company plan to go out of business before 2300? ). MM is definitely within 01-12 (it is 02). Is DD (29) valid for the year 2008? Yes it is.

COMPOPT (Compiler Option)

Software AG decided to help. They introduced a Compiler Option (COMPOPT) to make the IF ..MASK YYYYMMDD compatible with the MOVE EDITED statement. The default for this option is OFF. Presumably this is because for many years there was no compiler option, and Software AG does not want existing, functioning code, to suddenly stop working.

If you set this option (MASKCME) to ON, the IF ..MASK test will only be valid for the year range 1582 through 2699, just like the MOVE EDITED statement.

Summary

I like “built in” solutions. I do not like to program my way around problems. I find the IF... MASK solution, as modified by the COMPOPT, to be perfectly acceptable. Then again, I do not have any files that store dates which fall outside the range 1582-2699. My guess is that most of you do not either.
There have been several threads recently on SAG-L, and the Developers Forum, dealing with Multi-Fetch. Of course, right in the middle of one of the more interesting ones, I had a problem with my mainframe connection. Hence, rather than trying out certain things, I was reduced to suggesting other people run code I posted. Hardly satisfactory. However, in the due course of time, the mainframe problem was resolved, and I was again able to “play” to my hearts content.

Loop Limits

The first topic we will address is the interaction of Multi-Fetch and Loop limits. I must confess, I was really surprised by what you are about to see. We are going to start off with a very simple program.

```
> test dblog *          > +  Program     MULTI01  Lib XSTRO
0010 DEFINE DATA LOCAL
0020  1 MYVIEW VIEW OF EMPLOYEES
0030   2 NAME (A5)
0040   2 LEAVE-DUE
0050 END-DEFINE
0060 *
0070 INCLUDE AATITLER
0080 *
0090 READ (5) MYVIEW
0100 DISPLAY NAME LEAVE-DUE
0110 LOOP
0120 *
0130 END
```

Some things to note about the program. First, it is in Report Mode. There is a reason for this which has nothing to do with the Multi-Fetch we will be discussing. In Report Mode, but not in Structured Mode, I can override the format of Database fields in a View. In this case, I have changed the format of NAME from its default of A20 to A5. Why did I do this? As you can see above, I am starting DBLOG (the code “b”) and requesting, via the x, that the Record Buffers (RB) be saved. Just a side note about DBLOG. If you type TEST DBLOG b from the command line, you can start DBLOG without going to the screen shown above. HOWEVER, you will not have access to the optional buffers.

Later in the article you will see examples where I specified the b option on the command line. Here, since I wanted to specify the Record Buffer, I specified an asterisk so I went to the main DBLOG menu (above). This is where I can specify the Record Buffer.

As you can see above, I am starting DBLOG (the code “b”) and requesting, via the x, that the Record Buffers (RB) be saved. Just a side note about DBLOG. If you type TEST DBLOG b from the command line, you can start DBLOG without going to the screen shown above. HOWEVER, you will not have access to the optional buffers.
Okay, I next ran the program; here is our output.

Note that most of the NAMEs have been truncated. We have but five lines of output due to our Loop Limit of five. Now we return to DBLOG.

As shown below, I requested the Record Buffer for the first Read Command (L2). By the way, note we have not yet used Multi-Fetch.

Now we are going to do something which on the surface looks pretty silly. Take a look at the slightly modified code below.

Do you see the “silliness”? We have a loop limit of five, but a Multi-Fetch of ten. Why read ten records if you are only going to process five? Why indeed.

As above, I initiated DBLOG and requested that the Record Buffers be retained. Then I ran the program and went to DBLOG with the option “e”, which indicates an end to logging and a display of the log activity.

And here is our Record Buffer. Note that without Multi-Fetch, we read but one record from Adabas with the first L2 Command.
Okay, here we are (below) in DBLOG. As you can see, I have requested the display of the Record Buffer for the first Read Command.

That was pretty smart of Natural; and saved us half of the potential I/O. Just how smart Natural is, however, can be seen in the next program.

And here is the Record Buffer.

Now you can see why I reduced the size of the NAME field from A20 to A5. If it were still A20, only three records, and part of the fourth, would fit in the Record Buffer. By contrast, as you can see above, we can see more than ten records.

HOWEVER, note that we only have data for five records in our Record Buffer. Natural was smart enough to recognize that our READ loop would read exactly five records. Therefore, there was no reason to read the ten records specified in our Multi-Fetch clause.

If you look at the DBLOG screen above, you might think only five records were read from Adabas (the five L2 commands), WRONG. Note I have requested the Record Buffer from the first L2 command. It is shown below.
If you count up the number of “records” in the Record Buffer (each record has a five character NAME and a two character LEAVE-DUE) you will see that there are ten records that were read from Adabas and placed in the Record Buffer.

While Natural is quite smart (as in knowing that for MULTI02 it only had to read five records, even though the Multi-Fetch specified ten), it will not evaluate the WHERE clause to discover that this is actually always TRUE. Hence, because of the possibility that as many as ten records might have to be read, Natural “obeys” the Multi-Fetch setting.

The L2 commands other than the first just retrieve one records worth of data from the Multi-Fetch Buffer.

This is shown below.

There was one more scenario I wanted to confirm with regard to READs. Take a look at the following program.

This is the output from our program. Note we started with a JONES record, and ended with a KAISE record.

Here is the question. We can see that if Natural were “real smart” it could do a Multi-Fetch READ of ten records, followed by a Multi-Fetch READ of five records.

Earlier, we saw Natural do something similar. If you look back at the discussion of MULTI02, you will see that for an “opening READ”, Natural was smart enough to do a Multi-Fetch of just five records when faced with READ (5) MULTI-FETCH OF 10. So, what we are looking at is the same sort of decision, but one Multi-Fetch later.

Here is the output from our program. Note we started with a JONES record, and ended with a KAISE record.
Here is our DBLOG Trace, showing that we processed fifteen records.

Now, let's look at the L3 for the eleventh record.

Even though we know what the first L3 should look like, here is the Record Buffer from that first L3.

If you look back at the output from MULTI13, you will see that the first nine records are JONES records, and the tenth record is a JOPER. Now look at the Record Buffer above; these are the ten records read by the first L3 command.

Darn, this time we read too far. As noted earlier, the fifteenth record was KAISE. We read five records further to KATZ. In other words, once past the opening Multi-Fetch, all Multi-Fetch reads will be for how ever many records were specified in the Multi-Fetch parameter. Nonetheless, it is pretty impressive that Natural does not Multi-Fetch on the opening Read, if it is not required. We will see this same capability repeated below when we discuss FIND with Multi-Fetch.

What about FIND's?

There are a lot of misconceptions with regard to FIND and Multi-Fetch. Fortunately, DBLOG makes it rather simple to reveal how they function. We will start with a rather simple program.

```
0100          WITH NAME = 'JONES'
0090 FIND MULTI-FETCH OF 10 MYVIEW
```

November 1996 ❖ 17
Prior to running the program I started DBLOG, with the Record Buffer option selected.

Above are the nine JONES records, with the FIRST-NAME’s truncated for the same reason as earlier, namely increase the number of records that fit into a single Record Buffer.

Here is the DBLOG Trace for the run of the program:

```
18:32:08      ***** NATURAL TEST UTILITIES *****              2008-05-23
User XSTRO                    - DBLOG Trace -                  Library XSTRO
M   No Cmd   DB   FNR  Rsp        ISN        ISQ  CID CID(Hex) OP Pgm      Line
_    1 S1   177     1             565          9  ??? 00900101    MULTI04  0090
_    2 L1   177     1             573             ??? 00900101 MN MULTI04  0090
_    3 L1   177     1             691             ??? 00900101 <N MULTI04  0090
_    4 L1   177     1             789             ??? 00900101 <N MULTI04  0090
_    5 L1   177     1             792             ??? 00900101 <N MULTI04  0090
_    6 L1   177     1             800             ??? 00900101 <N MULTI04  0090
_    7 L1   177     1             817             ??? 00900101 <N MULTI04  0090
_    8 L1   177     1            1060             ??? 00900101 <N MULTI04  0090
_    9 L1   177     1            1079             ??? 00900101 <N MULTI04  0090
_   10 RC   177                                       00000000 F  MULTI04  0140

Command ===>
Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
Help  Print Exit  Top   Posi  Bot    -     +                      Canc
```

Some minor discussion about how a Natural FIND “translates” into Adabas Commands (specifically, the FIND in the program above). The first command issued to Adabas is a Search command (S1) which actually performs two functions. First, it goes to the inverted table for NAME and extracts the ISN list for NAME equal to JONES. Then the same Search Command reads the first record on the list and passes it back to our Natural program in the Record Buffer. Then, the remaining eight JONES records are read, one at a time, by the L1 (READ) commands which follow the S1 command.

As shown below, I requested the Record Buffer for the S1 command.

```
18:32:08      ***** NATURAL TEST UTILITIES *****              2008-05-23
User XSTRO                    - DBLOG Trace -                  Library XSTRO
M   No Cmd   DB   FNR  Rsp        ISN        ISQ  CID CID(Hex) OP Pgm      Line
r    1 S1   177     1             565          9 ??? 00900101 MULTI04  0090
_    2 L1   177     1             573             ??? 00900101 MN MULTI04  0090
_    3 L1   177     1             691             ??? 00900101 <N MULTI04  0090
_    4 L1   177     1             789             ??? 00900101 <N MULTI04  0090
_    5 L1   177     1             792             ??? 00900101 <N MULTI04  0090
_    6 L1   177     1             800             ??? 00900101 <N MULTI04  0090
_    7 L1   177     1             817             ??? 00900101 <N MULTI04  0090
_    8 L1   177     1            1060             ??? 00900101 <N MULTI04  0090
_    9 L1   177     1            1079             ??? 00900101 <N MULTI04  0090
_   10 RC   177                                       00000000 F  MULTI04  0140

Command ===>
Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
Help  Print Exit  Top   Posi  Bot    -     +                      Canc
```

And here is the Record Buffer for the S1.
Now we will request the Record Buffer for the first L1 Command.

<table>
<thead>
<tr>
<th>18:32:30</th>
<th>***** NATURAL TEST UTILITIES *****</th>
<th>2008-05-23</th>
</tr>
</thead>
<tbody>
<tr>
<td>User XSTRO</td>
<td>- DBLOG Trace -</td>
<td>Library XSTRO</td>
</tr>
<tr>
<td>M No Ord</td>
<td>DB</td>
<td>FNR</td>
</tr>
<tr>
<td>_</td>
<td>1</td>
<td>S1</td>
</tr>
<tr>
<td>r</td>
<td>2</td>
<td>L1</td>
</tr>
<tr>
<td>_</td>
<td>3</td>
<td>L1</td>
</tr>
<tr>
<td>_</td>
<td>4</td>
<td>L1</td>
</tr>
<tr>
<td>_</td>
<td>5</td>
<td>L1</td>
</tr>
<tr>
<td>_</td>
<td>6</td>
<td>L1</td>
</tr>
<tr>
<td>_</td>
<td>7</td>
<td>L1</td>
</tr>
<tr>
<td>_</td>
<td>8</td>
<td>L1</td>
</tr>
<tr>
<td>_</td>
<td>9</td>
<td>L1</td>
</tr>
<tr>
<td>_</td>
<td>10</td>
<td>RC</td>
</tr>
</tbody>
</table>

Command ===>
Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
Help  Print Exit  Top   Posi  Bot    -     +                      Canc

And here is the Record Buffer.

<table>
<thead>
<tr>
<th>18:32:30</th>
<th>***** NATURAL TEST UTILITIES *****</th>
<th>2008-05-23</th>
</tr>
</thead>
<tbody>
<tr>
<td>User XSTRO</td>
<td>- DBLOG Trace -</td>
<td>Library XSTRO</td>
</tr>
<tr>
<td>M No Ord</td>
<td>DB</td>
<td>FNR</td>
</tr>
<tr>
<td>_</td>
<td>1</td>
<td>S1</td>
</tr>
<tr>
<td>_</td>
<td>2</td>
<td>RC</td>
</tr>
<tr>
<td>_</td>
<td>3</td>
<td>RC</td>
</tr>
<tr>
<td>_</td>
<td>4</td>
<td>RC</td>
</tr>
<tr>
<td>_</td>
<td>5</td>
<td>RC</td>
</tr>
<tr>
<td>_</td>
<td>6</td>
<td>RC</td>
</tr>
<tr>
<td>_</td>
<td>7</td>
<td>RC</td>
</tr>
<tr>
<td>_</td>
<td>8</td>
<td>RC</td>
</tr>
<tr>
<td>_</td>
<td>9</td>
<td>RC</td>
</tr>
<tr>
<td>_</td>
<td>10</td>
<td>RC</td>
</tr>
</tbody>
</table>

Note that we have a Loop Limit of just one. Also recall that the S1 command reads the first record. Following is our output, restricted to one record as per our Loop Limit.

| MORE |
| PAGE # | 1 |
| DATE: | May 23, 2008 |
| PROGRAM: MULTI05 | LIBRARY: XSTRO |
| FIRST-NAME LEAVE | DUR |

| VIRGIL | 25 |

Note above that Natural just issued one Command to Adabas (not counting the RCs). The question is whether or not Multi-Fetch had any effect on the execution.

Rather edifying. Note that the first L1 Command shows the “effect” of the MultiFetch. The Record Buffer contains the data for the second through ninth records. Also note that the MultiFetch had no effect whatsoever on the Search command (S1).

Now for some important variations of our program.

<table>
<thead>
<tr>
<th>18:35:18</th>
<th>***** NATURAL TEST UTILITIES *****</th>
<th>2008-05-23</th>
</tr>
</thead>
<tbody>
<tr>
<td>User XSTRO</td>
<td>- DBLOG Trace -</td>
<td>Library XSTRO</td>
</tr>
<tr>
<td>M No Ord</td>
<td>DB</td>
<td>FNR</td>
</tr>
<tr>
<td>_</td>
<td>1</td>
<td>S1</td>
</tr>
<tr>
<td>_</td>
<td>2</td>
<td>RC</td>
</tr>
<tr>
<td>_</td>
<td>3</td>
<td>RC</td>
</tr>
<tr>
<td>_</td>
<td>4</td>
<td>RC</td>
</tr>
<tr>
<td>_</td>
<td>5</td>
<td>RC</td>
</tr>
<tr>
<td>_</td>
<td>6</td>
<td>RC</td>
</tr>
<tr>
<td>_</td>
<td>7</td>
<td>RC</td>
</tr>
<tr>
<td>_</td>
<td>8</td>
<td>RC</td>
</tr>
<tr>
<td>_</td>
<td>9</td>
<td>RC</td>
</tr>
<tr>
<td>_</td>
<td>10</td>
<td>RC</td>
</tr>
</tbody>
</table>

Command ===>
Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
Help  Print Exit  Top   Posi  Bot    -     +                      Canc

Note above that Natural just issued one Command to Adabas (not counting the RCs). The question is whether or not Multi-Fetch had any effect on the execution.
As shown above, I requested the Record Buffer for the Search command.

As you hopefully guessed, given the Record Buffers from MULTI04, there is no Multi-Fetch effect on the S1 command.

I was curious just how smart Natural was. Take a look at the following program.

Now I am forcing Natural to do one L1 command. Will Natural Multi-Fetch or not?

Drum roll, please. As shown below, Natural is smart enough to read just one record with the L1 command. Multi-Fetch is ignored.

20 ❖ Inside Natural
Now, we will go to DBLOG.

```plaintext
> test dblog e   > + Program MULTI07 Lib XSTRO
0010 DEFINE DATA LOCAL
0020 1 MYVIEW VIEW OF EMPLOYEES
0040 2 LEAVE-DUE
0050 END-DEFINE
0060 *
0070 INCLUDE AATITLER
0080 *
0090 FIND (2) MULTI-FETCH OF 10 MYVIEW
0100 WITH NAME = 'JONES'
0110 WHERE FIRST-NAME NE 'X'
0120 DISPLAY FIRST-NAME LEAVE-DUE
0130 LOOP
0140 *
0150 END
```

Here is our DBLOG Trace. Note I will be looking at the Record Buffer for the first L1.

```
18:40:33       ***** NATURAL TEST UTILITIES *****           2008-05-23
User XSTRO                  - DBLOG Trace -                  Library XSTRO
M   No Cmd   DB   FNR  Rsp        ISN        ISQ  CID CID(Hex) OP Pgm      Line
  _    1 BT   177                                       00000000    AERROR   1870
  _    2 BT   177                                       00000000    AERROR   1870
  _                                                                             0
  _       _     Seq No ..     6   Record Buffer                                 0
      0000 * D4C1D9E2 C8F3F4D9 D6C2C5D9 F1F1D3C9 * MARSH34ROBER11LI * 0000   0
      0010 * D3D3E8F1 F8C5C4E6 C1D9F2F1 D4C1D9E3 * LLY18EDWAR21MART * 0010   0
  _                                                                             0
      0020 * C8F3F0D3 C1E4D9C5 F1F4D2C5 E5C9D5F2 * H30LAURE14KEVIN2 * 0020
      0030 * F6C7D9C5 C7D6F2F5 00000000 00000000 * 6GREGO25         * 0030
      0040 * 00000000 00000000 00000000 00000000 *                  * 0040

Command ===>  Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
Help  Print Exit  Top   Posi  Bot    -     +                      Canc
```

And here is the Record Buffer.

```
18:40:33       ***** NATURAL TEST UTILITIES *****           2008-05-23
User XSTRO                  - DBLOG Trace -                  Library XSTRO
M   No Ond DB FNR Rep ISN ISQ CID CID(Hex) OP Pgm Line
  _    1 BT   177                                        00000000    AERROR   1870
  _    2 BT   177                                        00000000    AERROR   1870
  _                                                                             0
  _                                                                 0
  _                                                                             0
  _                                                                             0
  _                                                                             0
  _                                                                             0
  _                                                                             0
  _                                                                             0
  _                                                                             0

Command ===>  Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
Help  Print Exit Top Posi Bot    -     +                      Canc
```

Note that the Record Buffer for the L1 command has all the remaining JONES records (recall that the first JONES record was read by the S1 command). The presence of the WHERE clause results in the reading of up to 10 (size of the Multi-Fetch buffer) records. However, since there are only eight records remaining on the ISN list for JONES, we only read those eight records.

**UPDATE, DELETE, and STORE**

Important observation coming up. Suppose I am using Multi-Fetch. I have the parameter set to 100. I issue my first READ statement. As you have seen above, the Record Buffer gets filled with data from the first 100 records.

Problem. The READ, from Adabas, of the 100th record in the sequence takes place at the same time as the read of the first record. Suppose (it does not actually work this way, as we will discuss shortly) I had an UPDATE statement in my READ loop. The READ of the first record would place one hundred records on HOLD. Why would this be bad? Two possible reasons. First, it might well make a “mess” of Transaction Logic. Suppose this were a Batch program and every fifty records I issued an END TRANSACTION statement. Such a statement is supposed to release all records that are on HOLD. BUT, there are fifty records that were placed in the Record Buffer that I have not yet processed. Big Problem. Also, there are two parameters that could be “in trouble”. There is an ET timeout mechanism. If I tried to synchronize the issuing of ET statements with the number specified in a Multi-Fetch, I might exceed the ET Timeout. I might also contribute to a problem with the Record Hold Queue.

Hence, when you have any read command to Adabas (generated by a READ or a FIND statement in Natural), and there is a need for Natural to hold the record (UPDATE or DELETE), Natural ignores Multi-Fetch. That’s right, Multi-Fetch is ignored. Here is an example. First, we will start with a program that does not have an UPDATE or DELETE.
Here is the DBLOG Trace for this program:

Note, as expected, the first and eleventh READ commands (L3) actually communicate with Ada-bas; the other READs work from the Multi-Fetch Buffer.

Now we will modify the program ever so slightly.

Here is the DBLOG Trace for the program:
Two things to note above. The READ commands are L6, rather than L3 (as above in the DBLOG Trace for MULTI08). L6 is just an L3 with a Hold.

Now, take a look at the OP column. In MULTI08, this column had the value MA for the first and eleventh L3 commands and <A for the other L3 commands, indicating that there was no communication with Adabas for those commands. In MULTI09, by contrast, all the OP values are RA, indicating they communicate with Adabas.

In other words, Multi-Fetch is not occurring. I wish Natural would provide an error message for this piece of code. Basically, the compiler is not doing what I asked it to do. If I knew that Multi-Fetch would not be utilized, I might have written my code differently.

Here is a similar program without MULTI-FETCH.

```
>                        > +  Program     MULTI10  Lib XSTRO
0010 DEFINE DATA LOCAL
0020 1 MYVIEW VIEW OF EMPLOYEES
0030 2 NAME
0040 2 FIRST-NAME
0050 2 LEAVE-DUE
0060 END-DEFINE
0070 *
0080 INCLUDE AATITLER
0090 *
0100 READ (15) MYVIEW
0110          BY NAME STARTING FROM 'JONES'
0120 DISPLAY NAME FIRST-NAME LEAVE-DUE
0130 IF NAME NE NAME
0140    UPDATE
0150 LOOP
0160 *
0170 END
```

And here is the DBLOG Trace.

```
15:12:43             ***** NATURAL TEST UTILITIES *****              2008-05-24
User XSTRO            +  DBLOG Trace -  Library XSTRO
N No Omd DB FNR Rep   ISR  ISQ  CID CID(Rex) OP Rgs  Line
_  1 L6 177  1       565  ? ?? 01000101 RA MULTI10  0100
_  2 L6 177  1       573  ? ?? 01000101 RA MULTI10  0100
_  3 L6 177  1       691  ? ?? 01000101 RA MULTI10  0100
_  4 L6 177  1       789  ? ?? 01000101 RA MULTI10  0100
_  5 L6 177  1       792  ? ?? 01000101 RA MULTI10  0100
_  6 L6 177  1       800  ? ?? 01000101 RA MULTI10  0100
_  7 L6 177  1       817  ? ?? 01000101 RA MULTI10  0100
_  8 L6 177  1      1060  ? ?? 01000101 RA MULTI10  0100
_  9 L6 177  1      1079  ? ?? 01000101 RA MULTI10  0100
_ 10 L6 177  1      1111  ? ?? 01000101 RA MULTI10  0100
_ 11 L6 177  1        11  ? ?? 01000101 RA MULTI10  0100
_ 12 L6 177  1         98  ? ?? 01000101 RA MULTI10  0100
_ 13 L6 177  1       237  ? ?? 01000101 RA MULTI10  0100
_ 14 L6 177  1       944  ? ?? 01000101 RA MULTI10  0100
_ 15 L6 177  1       239  ? ?? 01000101 RA MULTI10  0100
_ 16 RC 177  1     00000000 F  MULTI10  0170
_ 17 RC 177  1

Command =>>
Enter:PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
Help Print Exit Top Posi Bot - *  Canc
```

If you were to compare the Record Buffers for the program above and the earlier MULTI09, which had a MULTI-FETCH, you would discover they are identical.

Okay, so far we have made an important observation regarding UPDATE. What will happen with DELETE? Here is a simple program, and our DBLOG initiation:

```
> test dblog b        > +  Program     MULTI11  Lib XSTRO
0010 DEFINE DATA LOCAL
0020 1 MYVIEW VIEW OF EMPLOYEES
0030 2 NAME
0040 2 FIRST-NAME
0050 2 LEAVE-DUE
0060 END-DEFINE
0070 *
0080 INCLUDE AATITLER
0090 *
0100 READ (15) MYVIEW
0110    BY NAME STARTING FROM 'JONES'
0120 DISPLAY NAME FIRST-NAME LEAVE-DUE
0130 IF NAME NE NAME
0140    DELETE
0150 LOOP
0160 *
0170 END
```

And now I ran the program. Here is our output:

```
MORE
PAGE #  1          DATE: May 24, 2008
PROGRAM: MULTI11              LIBRARY: XSTRO
NAME              FIRST-NAME      LEAVE     DUE
-------------------- -------------------- ----------
JONES                VIRGINIA              25     00
JONES                MARSHA                34     00
JONES                ROBERT                11     00
JONES                LILLY                 18     00
JONES                EDWARD                21     00
JONES                MARTHA                30     00
JONES                LAUREL                14     00
JONES                KEVIN                 26     00
JONES                GREGORY               25     00
JOSPER                MANFRED               32     00
JOUSSELIN            DANIEL                45     00
JUNG                 GABRIEL                19     00
JUNKIN                ERNST                 26     00
JUNKIN                JEREMY                20     00
KAISER                REINER                25     00
```

November 1996 ❖ 23
Now let's go look at our DBLOG.

> test dblog e  > + Program  MULTI11 Lib XSTRO
0010 DEFINE DATA LOCAL
0020 1 MYVIEW VIEW OF EMPLOYEES
0030 2 NAME
0040 2 FIRST-NAME
0050 2 LEAVE-DUE
0060 END-DEFINE
0070 *
0080 INCLUDE AATITLER
0090 *
0100 READ (15) MYVIEW
0110 BY NAME STARTING FROM 'JONES'
0120 DISPLAY NAME FIRST-NAME LEAVE-DUE
0130 IF NAME NE NAME
0140    DELETE
0150 LOOP
0160 *
0170 END

Here is the DBLOG.

15:15:10 ***** NATURAL TEST UTILITIES ***** 2008-05-24
User XSTRO - DBLOG Trace - Library XSTRO
M No Cmd DB FNR Res ISN ISQ CID CID(Hex) OP Pgm Line
_ 1 L6 177 1 565 ?? 01000101 RA MULTI11 0100
_ 2 L6 177 1 789 ?? 01000101 RA MULTI11 0100
_ 3 L6 177 1 817 ?? 01000101 RA MULTI11 0100
_ 4 L6 177 1 1060 ?? 01000101 RA MULTI11 0100
_ 5 L6 177 1 1079 ?? 01000101 RA MULTI11 0100
_ 6 L6 177 1 311 ?? 01000101 RA MULTI11 0100
_ 7 L6 177 1 11 ?? 01000101 RA MULTI11 0100
_ 8 L6 177 1 98 ?? 01000101 RA MULTI11 0100
_ 9 L6 177 1 237 ?? 01000101 RA MULTI11 0100
_ 10 L6 177 1 239 ?? 01000101 RA MULTI11 0100
_ 11 L6 177 1 1 ?? 01000101 RA MULTI11 0100
_ 12 L6 177 1 444 ?? 01000101 RA MULTI11 0100
_ 13 L6 177 1 573 ?? 01000101 RA MULTI11 0100
_ 14 L6 177 1 691 ?? 01000101 RA MULTI11 0100
_ 15 L6 177 1 789 ?? 01000101 RA MULTI11 0100
_ 16 L6 177 1 792 ?? 01000101 RA MULTI11 0100
_ 17 RC 177 00000000 F MULTI11 0170

As expected (there is no MultiFetch), the READ commands are all L6's (READ LOGICAL with Hold), and each READ is a call to Adabas.

> test dblog b  > + Program  MULTI12 Lib XSTRO
0010 DEFINE DATA LOCAL
0020 1 MYVIEW VIEW OF EMPLOYEES
0030 2 NAME
0040 2 FIRST-NAME
0050 2 LEAVE-DUE
0060 END-DEFINE
0070 *
0080 INCLUDE AATITLER
0090 *
0100 READ (15) MYVIEW MULTI-FETCH OF 10
0110 BY NAME STARTING FROM 'JONES'
0120 DISPLAY NAME FIRST-NAME LEAVE-DUE
0130 IF NAME NE NAME
0140    DELETE
0150 LOOP
0160 *
0170 END

After running the program, here is the DBLOG.

15:17:13 ***** NATURAL TEST UTILITIES ***** 2008-05-24
User XSTRO - DBLOG Trace - Library XSTRO
M No Cmd DB FNR  Res ISN ISQ CID CID(Hex) OP Pgm Line
_ 1 BT 177 00000000 F  AERROR 1870
_ 2 BT 177 00000000 F  AERROR 1870
_ 3 RC 177 00000000 F  SYSFUL00 4830
_ 4 RC 177 00000000 F  SYSCHCK 4870
_ 5 ET 177 00000000 0000
_ 6 L6 177 1 565 ?? 01000101 RA MULTI12 0100
_ 7 L6 177 1 789 ?? 01000101 RA MULTI12 0100
_ 8 L6 177 1 817 ?? 01000101 RA MULTI12 0100
_ 9 L6 177 1 1060 ?? 01000101 RA MULTI12 0100
_10 L6 177 1 1079 ?? 01000101 RA MULTI12 0100
_11 L6 177 1 311 ?? 01000101 RA MULTI12 0100
_12 L6 177 1 11 ?? 01000101 RA MULTI12 0100
_13 L6 177 1 98 ?? 01000101 RA MULTI12 0100
_14 L6 177 1 237 ?? 01000101 RA MULTI12 0100
_15 L6 177 1 239 ?? 01000101 RA MULTI12 0100
_16 L6 177 1 1 ?? 01000101 RA MULTI12 0100
_17 RC 177 00000000 F MULTI11 0170

As is evident above, Natural realized that there could be a problem if it held too many records which might be DELETEd. As a consequence, MultiFetch was not implemented for this program.
And now for a “gotcha”

Actually, this is a reprise of an article that appeared in Volume 15 No 2. Why bother? Three years ago many shops were not yet looking at Multi-Fetch. I am sure that a relatively small percentage of the readership paid a lot of attention to the article. Today, however, is another matter.

When NOT to MULTIFETCH

I was teaching a class recently. The topic under discussion was the new feature in Version 4 to ESCAPE TOP REPOSITION. I was comparing the code to use this feature versus the “old” approach of placing a READ loop within a “dummy” REPEAT loop. That code typically looked something like:

```
REPEAT
  READ…
  ::::
  IF some condition
    set flag
    ESCAPE BOTTOM IMMEDIATE
    END-IF
  ::::
  END-READ
  IF FLAG set
    RESET FLAG
    ESCAPE TOP
    END-IF
END-REPEAT
```

As we discussed in a recent article, the new, much simpler code is something like:

```
READ…. WITH REPOSITION
  ::::
  IF some condition
    ESCAPE TOP REPOSITION
    END-IF
  ::::
  END-READ
```

I think you will agree, as did the class, that the new code is much simpler to write, and understand. Someone in the class wondered whether the new code would solve a problem they had that necessitated the dummy REPEAT loop code. I asked for a description of the problem. I will simplify the description a bit to maintain just the portions relevant to our discussion.

We have an order file. There are many line items per order. Every line item of every order is a separate record (it is not in our purview to suggest a redesign with a PE). Line item numbers are assigned serially. There is a super descriptor which concatenates the order number and line number.

There is an application which accepts an order number. It then reads by the superdescriptor. It is quite possible that in the process of reading the line items for an order, a new line item (record) is created for the order. The line item number will be the next one available (existing maximum plus one).

Originally they had simple code such as:

```
READ ORDER-VIEW BY ORDER-LINE-NUMBER
  STARTING FROM #ORDER-LINE-ONE
  IF ORDER-NUMBER NE #ORDER-NUMBER
    ESCAPE BOTTOM IMMEDIATE
    END-IF
  ::::
  IF some condition
    STORE new line item record
    END TRANSACTION
    END-IF
  ::::
  END-READ
```

Problem. Sometimes when they reach the end of an order, they see the new line item record (which is what they want). However, sometimes they do not see the new record.

Solution. They embedded the READ loop in a dummy REPEAT loop. Instead of the logic above they coded something similar to:
Okay, put your thinking caps on. Why did they have the original problem? Why did the code above solve the problem?

My first question was whether they were using PRE-FETCH or MULTI-FETCH. They said no. I was about to give up on this line of thought, when they said they did use Adabas FASTPATH. “Aha”, I said, almost spilling my coffee. The problem is solved. You can get rid of the weird code with the dummy REPEAT loop. What did I realize?

FASTPATH, Pre-Fetch and Multi-Fetch share a common approach to reducing overhead, namely, reading ahead. Here is what can happen. We will use the familiar Employees file for our discussion. Here is a simple program that reads the SMITH records.

As you can see from the output, there are nineteen SMITH records. Now take a look at the following program. Please note that MULTI-FETCH does not appear here. This is basically the way the company’s order processing program worked originally (before they “discovered” FASTPATH).
As you can see, the new SMITH record is indeed there.

Okay, now we will look at almost the same program. The exception is that this will have a MULTI-FETCH. In the company’s actual scenario, they decided to use FASTPATH. That was even more confusing than what you are about to see, since there was nothing “visible” in the code. At least here, we have the following change; first the old code:

```
0100 READ MYVIEW BY NAME STARTING FROM 'SMITH'
```

now the new code:

```
0100 READ MULTI-FETCH OF 10 MYVIEW
0105        BY NAME STARTING FROM 'SMITH'
```

And here is the output:

```
As you can see, the new SMITH record is indeed there.
Okay, where did the new SMITH record go?
```

November 1996 ✦ 27
Here is a minor modification of the last program:

```
> DEFINE DATA LOCAL
0010  1 MYVIEW VIEW OF EMPLOYEES
0020  2 NAME
0030  2 FIRST-NAME
0040  END-DEFINE
0050 **
0060  INCLUDE AATITLER
0070  **INCLUDE AASETIC
0080 **
0090  READ MULTI-FETCH OF 10
0100   MYVIEW BY NAME STARTING FROM 'SMITH'
0110   IF NAME NE 'SMITH'
0120      ESCAPE BOTTOM IMMEDIATE
0130      END-IF
0140   DISPLAY *COUNTER *ISN FIRST-NAME
0150 **
0160   IF *COUNTER = 15
0170      MOVE 'IS THIS THERE' TO FIRST-NAME
0180      STORE MYVIEW
0190   END-IF
0200 *
0210 END-READ
Y
0220 END-READ
0230 WRITE ST 'AFTER THE STORE'
0240 NEWPAGE
0250 ***************
0260 READ MYVIEW BY NAME STARTING FROM 'SMITH'
0270   IF NAME NE 'SMITH'
0280      ESCAPE BOTTOM IMMEDIATE
0290      END-IF
0300   DISPLAY *COUNTER *ISN FIRST-NAME
0310 **
0320 END-READ
X
0340 END-READ
0350 BACKOUT TRANSACTION
```

Note that after the original READ loop, with the STORE, I have added a READ loop that simply reads all the SMITH records.

Here are the modified output:

```
<table>
<thead>
<tr>
<th>CNT</th>
<th>ISN</th>
<th>FIRST-NAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>526</td>
<td>GERHARD</td>
</tr>
<tr>
<td>2</td>
<td>581</td>
<td>SEYMOUR</td>
</tr>
<tr>
<td>3</td>
<td>626</td>
<td>MATILDA</td>
</tr>
<tr>
<td>4</td>
<td>639</td>
<td>ANN</td>
</tr>
<tr>
<td>5</td>
<td>669</td>
<td>TONI</td>
</tr>
<tr>
<td>6</td>
<td>716</td>
<td>MARTIN</td>
</tr>
<tr>
<td>7</td>
<td>732</td>
<td>THOMAS</td>
</tr>
<tr>
<td>8</td>
<td>776</td>
<td>SUNNY</td>
</tr>
<tr>
<td>9</td>
<td>785</td>
<td>MARK</td>
</tr>
<tr>
<td>10</td>
<td>791</td>
<td>LOUISE</td>
</tr>
<tr>
<td>11</td>
<td>799</td>
<td>MAXWELL</td>
</tr>
<tr>
<td>12</td>
<td>807</td>
<td>ELSA</td>
</tr>
<tr>
<td>13</td>
<td>816</td>
<td>CHARLY</td>
</tr>
<tr>
<td>14</td>
<td>822</td>
<td>LEE</td>
</tr>
<tr>
<td>15</td>
<td>852</td>
<td>FRANK</td>
</tr>
<tr>
<td>16</td>
<td>876</td>
<td>GERALD</td>
</tr>
<tr>
<td>17</td>
<td>1052</td>
<td>FRANCIS</td>
</tr>
</tbody>
</table>
```

Now the WRITE inbetween the two READ loops:

```
<table>
<thead>
<tr>
<th>CNT</th>
<th>ISN</th>
<th>FIRST-NAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>18</td>
<td>1070</td>
<td>WINSTON</td>
</tr>
<tr>
<td>19</td>
<td>1106</td>
<td>JUNE</td>
</tr>
</tbody>
</table>
```
Notice that in the second READ loop we do see the additional SMITH record. Since there was no STORE between the two READ loops, clearly the record "existed" during the first READ loop.

Now here is the way "controversy" gets started. Someone ran the order processing program and did not see the added record (the second loop). They told someone about this, and they then ran the program again. Of course, now they saw the record. However, suppose they ran the program again, and this time the record that produced the STORE was record number three rather than record number fifteen.

> > + Program SMITH02 Lib XSTRO
0010 DEFINE DATA LOCAL
0020 1 MYVIEW VIEW OF EMPLOYEES
0030 2 NAME
0040 2 FIRST-NAME
0050 END-DEFINE
0060 **
0070 INCLUDE AATITLER
0080 INCLUDE AASETC
0090 **
0100 READ MULTI-FETCH OF 10
0110 MYVIEW BY NAME STARTING FROM 'SMITH'
0120 IF NAME NE 'SMITH'
0130 ESCAPE BOTTOM IMMEDIATE
0140 END-IF
0150 DISPLAY *COUNTER *ISN FIRST-NAME
0160 **
0170 IF *COUNTER = 3
0180 MOVE 'IS THIS THERE' TO FIRST-NAME
0190 STORE MYVIEW
0200 END-IF
0210 *
0220 END-READ
0230 BACKOUT TRANSACTION
0240 END

One can imagine the potential ensuing conversation. "Worked fine for me"; "well, not for me". Of course, both parties were correct in their observations. As long as the STORE was made prior to the last "buffer fill", the record would be there. However, if the STORE was made after the buffer was filled the last time, the record would not be there.

I can imagine a lot of experimentation with the type of record that caused the STORE, the data going in to the STORE’d record, etc. Probably very little experimentation with the "timing" of the STORE, which is the critical difference.

DBLOG

One tool which would help if it was MULTI-FETCH causing the problem (it wasn’t; it was FASTPATH) is DBLOG.

Let’s backup to the first program. We have a MULTI-FETCH of ten records. Here is what the DBLOG looks like for a run of this program.

And, the output which was expected (as opposed to the previous output).

<table>
<thead>
<tr>
<th>CNT</th>
<th>ISN</th>
<th>FIRST-NAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>526</td>
<td>GEHARD</td>
</tr>
<tr>
<td>2</td>
<td>581</td>
<td>SPENCER</td>
</tr>
<tr>
<td>3</td>
<td>626</td>
<td>MATILDA</td>
</tr>
<tr>
<td>4</td>
<td>639</td>
<td>ANN</td>
</tr>
<tr>
<td>5</td>
<td>669</td>
<td>TONI</td>
</tr>
<tr>
<td>6</td>
<td>716</td>
<td>MARTIN</td>
</tr>
<tr>
<td>7</td>
<td>732</td>
<td>THOMAS</td>
</tr>
<tr>
<td>8</td>
<td>776</td>
<td>SUNNY</td>
</tr>
<tr>
<td>9</td>
<td>785</td>
<td>MARK</td>
</tr>
<tr>
<td>10</td>
<td>791</td>
<td>LOUISE</td>
</tr>
<tr>
<td>11</td>
<td>799</td>
<td>MAXWELL</td>
</tr>
<tr>
<td>12</td>
<td>807</td>
<td>ELSA</td>
</tr>
<tr>
<td>13</td>
<td>816</td>
<td>CHARLY</td>
</tr>
<tr>
<td>14</td>
<td>822</td>
<td>LEE</td>
</tr>
<tr>
<td>15</td>
<td>852</td>
<td>FRANK</td>
</tr>
<tr>
<td>16</td>
<td>876</td>
<td>OGERALD</td>
</tr>
<tr>
<td>17</td>
<td>1052</td>
<td>FRANCES</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>CNT</th>
<th>ISN</th>
<th>FIRST-NAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>18</td>
<td>1070</td>
<td>WINSTON</td>
</tr>
<tr>
<td>19</td>
<td>1106</td>
<td>JUNE</td>
</tr>
<tr>
<td>20</td>
<td>1109</td>
<td>IS THIS THERE</td>
</tr>
</tbody>
</table>

November 1996 ♦ 29
The first ten records are read into the MULTIFETCH buffer as before (line 1). We process the first ten records. Then we read the next ten records into the MULTIFETCH buffer (line 11). During the processing of the next ten records we add a SMITH record (line 16, ISN 1115). HOWEVER, this does not change the contents of the MULTIFETCH buffer. After ISN 1106 (previous last SMITH), the next ISN in the buffer is 1029, which is a non SMITH. As a consequence, we do not “see” the new SMITH record in the READ loop.

Now realize how confusing this could be. Some of the time the program “works”, that is, the new record is there. Some of the time, the new record is not there. Unless you were already looking for something related to MULTIFETCH, it might not appear significant, indeed, you might not even notice, that the problem only occurred when you were into the last buffer before you added the new record.

If you think that is a problem, lets look at a scenario I think is even worse. We have a program that has always worked, namely, the MULTI-FETCH program above, SMITH02.

The reason this program always works is as follows. If we are going to add a new record, we always decide to do the add within the first five records (recall, the problem only occurred when we added the record during the processing of the last ten records). However, assume we always have 10-18 records per name. As you can see, with these assumptions, we will never get into trouble. We will always generate the new record (if required) while working out of the first MULTI-FETCH “block” of records. Since the new record will be in the second block, we will always “see” the new record.

Now, someone mentions that a bigger MULTI-FETCH buffer might be better. Instead of calling Adabas two, or three, or even four times, we could increase the buffer size to 20, and only call Adabas once.

Note that we called Adabas for the first record (line 3), and read a total of ten records into the MULTIFETCH buffer (the M before the A). Now, one at a time, we read the next nine records out of the MULTIFETCH buffer. While this was going on (during the third iteration of the READ loop), we created our new SMITH record (line 6, ISN1114).

Now we read the eleventh record (line 14). Natural actually read the next ten records into the MULTIFETCH buffer. This included the new SMITH record (line 23, ISN 1114) and the first non SMITH record (line 24, ISN 1029).

As is evident from DBLOG, we did indeed see the new SMITH record.

Now, here is the DBLOG from the first run of the program SMITH03 (the one without the second READ loop).

The first ten records are read into the MULTIFETCH buffer as before (line 1). We process the first ten records. Then we read the next ten records into the MULTIFETCH buffer (line 11). During the processing of the next ten records we add a SMITH record (line 16, ISN 1115). HOWEVER, this does not change the contents of the MULTIFETCH buffer. After ISN 1106 (previous last SMITH), the next ISN in the buffer is 1029, which is a non SMITH. As a consequence, we do not “see” the new SMITH record in the READ loop.

Now realize how confusing this could be. Some of the time the program “works”, that is, the new record is there. Some of the time, the new record is not there. Unless you were already looking for something related to MULTIFETCH, it might not appear significant, indeed, you might not even notice, that the problem only occurred when you were into the last buffer before you added the new record.

If you think that is a problem, lets look at a scenario I think is even worse. We have a program that has always worked, namely, the MULTI-FETCH program above, SMITH02.

The reason this program always works is as follows. If we are going to add a new record, we always decide to do the add within the first five records (recall, the problem only occurred when we added the record during the processing of the last ten records). However, assume we always have 10-18 records per name. As you can see, with these assumptions, we will never get into trouble. We will always generate the new record (if required) while working out of the first MULTI-FETCH “block” of records. Since the new record will be in the second block, we will always “see” the new record.

Now, someone mentions that a bigger MULTI-FETCH buffer might be better. Instead of calling Adabas two, or three, or even four times, we could increase the buffer size to 20, and only call Adabas once.
Here is the code:

```
>                    > +  Program  SMITH02 Lib XSTRO
 0010  DEFINE DATA LOCAL
 0020  1 MYVIEW VIEW OF EMPLOYEES
 0030    2 NAME
 0040    2 FIRST-NAME
 0050  END-DEFINE
 0060 **
 0070  INCLUDE AATITLER
 0080  INCLUDE AASETC
 0090 **
 0100  READ MULTI-FETCH OF 20
 0110       MYVIEW BY NAME STARTING FROM 'SMITH'
 0120       IF NAME NE 'SMITH'
 0130          ESCAPE BOTTOM IMMEDIATE
 0140       END-IF
 0150  DISPLAY *COUNTER *ISN FIRST-NAME
 0160 **
 0170       IF *COUNTER = 3
 0180       MOVE 'IS THIS THERE' TO FIRST-NAME
 0190       STORE MYVIEW
 0200  END-IF
 0210 *
 0220  END-READ
 0230  BACKOUT TRANSACTION
 0240  END
```

And here is our output:

<table>
<thead>
<tr>
<th>CNT</th>
<th>ISN</th>
<th>FIRST-NAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>526</td>
<td>GERHARD</td>
</tr>
<tr>
<td>2</td>
<td>581</td>
<td>SEYMOUR</td>
</tr>
<tr>
<td>3</td>
<td>626</td>
<td>MATILDA</td>
</tr>
<tr>
<td>4</td>
<td>639</td>
<td>ANN</td>
</tr>
<tr>
<td>5</td>
<td>669</td>
<td>TONI</td>
</tr>
<tr>
<td>6</td>
<td>716</td>
<td>MARTIN</td>
</tr>
<tr>
<td>7</td>
<td>732</td>
<td>THOMAS</td>
</tr>
<tr>
<td>8</td>
<td>776</td>
<td>SUNNY</td>
</tr>
<tr>
<td>9</td>
<td>785</td>
<td>MARK</td>
</tr>
<tr>
<td>10</td>
<td>791</td>
<td>LOUISE</td>
</tr>
<tr>
<td>11</td>
<td>799</td>
<td>MAXWELL</td>
</tr>
<tr>
<td>12</td>
<td>807</td>
<td>ELSA</td>
</tr>
<tr>
<td>13</td>
<td>816</td>
<td>CHARLY</td>
</tr>
<tr>
<td>14</td>
<td>822</td>
<td>LEE</td>
</tr>
<tr>
<td>15</td>
<td>852</td>
<td>FRANK</td>
</tr>
<tr>
<td>16</td>
<td>876</td>
<td>GERALD</td>
</tr>
<tr>
<td>17</td>
<td>1052</td>
<td>FRANCIS</td>
</tr>
<tr>
<td>-----</td>
<td>-----</td>
<td>------------</td>
</tr>
</tbody>
</table>

Whoops. Where did the record go? Not hard to figure out. Our buffer now spans all the SMITH records and two records beyond. We do not have to regenerate the block of records, hence, we never see the new SMITH record.

Now, for someone not familiar with MULTI-FETCH (unlike you), this is quite confusing. Here is a program that has worked forever. We increase a MULTI-FETCH buffer, and suddenly the program doesn't work. All thoughts of efficiency are ignored in an effort to restore the program to working properly. Any bets that this means that the MULTI-FETCH buffer gets set back to ten, or even five?

Okay, if MULTI-FETCH can create such problems, what do we do about it. The first thing to note is that the Natural documentation does warn about trying to use MULTI-FETCH when doing updates. Unfortunately, most people who read this warning assume the narrow interpretation of “updates”, namely the UPDATE statement. Okay, sometimes they extend this interpretation to include DELETE. Almost never does anyone consider that the STORE statement is a potential problem. WRONG.

Fortunately, the new ESCAPE TOP REPOSITION facility can sometimes “solve” the problem. Why so? Suppose I return to the order system that was discussed at the start of this article. I could recode it something like:

```
READ MULTI-FETCH OF 20 ORDER-VIEW
   BY ORDER-LINE-NUMBER
   STARTING FROM #ORDER-LINE-ONE
   IF ORDER-NUMBER NE #ORDER-NUMBER
      ESCAPE BOTTOM IMMEDIATE
      END-IF
:::
   IF some condition
      STORE new line item record
      MOVE ORDER-LINE-NUMBER
      TO #ORDER-LINE-ONE
      ADD 1 TO #ORDER-LINE-ONE
      ESCAPE TOP REPOSITION
      END-IF
:::
END-READ
```
This would “solve” the problem since the ESCAPE TOP REPOSITION basically restarts the loop, thereby regenerating the MULTI-FETCH buffer. Notice this would not really help us with our SMITH sequence, since we are not reading by a unique value (as was the case with the superdescriptor that concatenates order number and a serially assigned number). However, if we combine this logic with the STARTING WITH ISN clause, we can effectively replace the “messy” code shown previously. Here is the new program:

```
>                     > +  Program     SMITH02A Lib XSTRO
0010  DEFINE DATA LOCAL
0020  1 MYVIEW VIEW OF EMPLOYEES
0030    2 NAME
0040    2 FIRST-NAME
0050 1 #ISN (I4) INIT <1>
0060  END-DEFINE
0070 **
0080  INCLUDE AATITLER
0090  INCLUDE AASETC
0100 **
0110  READ MULTI-FETCH OF 20  MYVIEW
0120          WITH REPOSITION
0130          BY NAME STARTING FROM 'SMITH'
0140          STARTING WITH ISN = #ISN
0150       IF NAME NE 'SMITH'
0160          ESCAPE BOTTOM IMMEDIATE
0170       END-IF
0180       DISPLAY *COUNTER *ISN FIRST-NAME
0190 **
0200       IF FIRST-NAME = 'MATILDA'
0210       MOVE *ISN TO #ISN
0220       MOVE 'IS THIS THERE' TO FIRST-NAME
0230       STORE MYVIEW
0240       ESCAPE TOP REPOSITION
0250       END-IF
0260 *
0270 END-READ
0280 BACKOUT TRANSACTION
0290 END
```

And here is our output:

```
<table>
<thead>
<tr>
<th>CNT</th>
<th>ISN</th>
<th>FIRST-NAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>526</td>
<td>GERRARD</td>
</tr>
<tr>
<td>2</td>
<td>581</td>
<td>SEYMOUR</td>
</tr>
<tr>
<td>3</td>
<td>626</td>
<td>MATILDA</td>
</tr>
<tr>
<td>4</td>
<td>639</td>
<td>ANN</td>
</tr>
<tr>
<td>5</td>
<td>669</td>
<td>TONI</td>
</tr>
<tr>
<td>6</td>
<td>716</td>
<td>MARTIN</td>
</tr>
<tr>
<td>7</td>
<td>732</td>
<td>THOMAS</td>
</tr>
<tr>
<td>8</td>
<td>776</td>
<td>SUNNY</td>
</tr>
<tr>
<td>9</td>
<td>791</td>
<td>LOUISE</td>
</tr>
<tr>
<td>10</td>
<td>799</td>
<td>MAXWELL</td>
</tr>
<tr>
<td>11</td>
<td>807</td>
<td>ELSA</td>
</tr>
<tr>
<td>12</td>
<td>822</td>
<td>CHARLY</td>
</tr>
<tr>
<td>13</td>
<td>852</td>
<td>LEE</td>
</tr>
<tr>
<td>14</td>
<td>1052</td>
<td>FRANK</td>
</tr>
<tr>
<td>15</td>
<td>1070</td>
<td>GERALD</td>
</tr>
<tr>
<td>16</td>
<td>1106</td>
<td>JUNE</td>
</tr>
<tr>
<td>17</td>
<td>1114</td>
<td>IS THIS THERE</td>
</tr>
</tbody>
</table>
```

Note that *COUNTER started again. Thus, I had to change the criteria I used in the original program. Instead of testing for *COUNTER = 3 to add the new record, I tested for FIRST-NAME = ‘MATILDA’. The important point, however, is the new record is indeed at the end of our report.

The DBLOG of the run above is particularly edifying. Here it is:

```
18:56:49             ***** NATURAL TEST UTILITIES *****              2008-06-11
User XSTRO                    - DBLOG Trace -                  Library XSTRO
M   No Cmd   DB   FNR  Rsp        ISN        ISQ  CID CID(Hex) OP Pgm      Line
_    1 BT 11177                                       00000000    AERROR   0930
_    2 RC 11177                                       00000000 F  AERROR   2140
_    3 RC 11177                                       00000000 F  SYSFUL00 4700
_    4 L3 11177     1             526            ???? 01100101 MA SMITH02A 0110
_    5 L3 11177     1             581            ???? 01100101 <A SMITH02A 0110
_    6 L3 11177     1             626            ???? 01100101 <A SMITH02A 0110
_    7 L3 11177     1             669            ???? 01100101 <A SMITH02A 0110
_    8 L3 11177     1             716            ???? 01100101 <A SMITH02A 0110
_    9 L3 11177     1             732            ???? 01100101 <A SMITH02A 0110
_   10 L3 11177     1             776            ???? 01100101 <A SMITH02A 0110
_   11 L3 11177     1             785            ???? 01100101 <A SMITH02A 0110
_   12 L3 11177     1             807            ???? 01100101 <A SMITH02A 0110
_   13 L3 11177     1             816            ???? 01100101 <A SMITH02A 0110
_   14 L3 11177     1             822            ???? 01100101 <A SMITH02A 0110
_   15 L3 11177     1             852            ???? 01100101 <A SMITH02A 0110
_   16 L3 11177     1             876            ???? 01100101 <A SMITH02A 0110
_   17 L3 11177     1             1052           ???? 01100101 <A SMITH02A 0110
_   18 L3 11177     1             1070           ???? 01100101 <A SMITH02A 0110
_   19 L3 11177     1             1106           ???? 01100101 <A SMITH02A 0110
_   20 L3 11177     1             1109           ???? 01100101 <A SMITH02A 0110
_   21 L3 11177     1             1029           ???? 01100101 <A SMITH02A 0110
_   22 L3 11177     1             1106           ???? 01100101 <A SMITH02A 0110
_   23 L3 11177     1             1109           ???? 01100101 <A SMITH02A 0110
_   24 L3 11177     1             1029           ???? 01100101 <A SMITH02A 0110
_   25 L3 11177     1             1106           ???? 01100101 <A SMITH02A 0110
_   26 L3 11177     1             1109           ???? 01100101 <A SMITH02A 0110
_   27 L3 11177     1             1029           ???? 01100101 <A SMITH02A 0110
_   28 L3 11177     1             1106           ???? 01100101 <A SMITH02A 0110
```

And here is our output:
Some things to note in the above DBLOG.

The first L3, at line 3, is a MultiFetch of twenty records. Due to the ESCAPE TOP REPOSITION, we only process three of the twenty records that were read from Adabas.

On line 8 we do another MultiFetch of twenty records. This time we actually process most of them (before our ESCAPE BOTTOM at line 0160).

**WARNING - The Windows and Unix “variation”**

Do not try the examples above on your PC. They will NOT work the same way. I thought I had found a bug. Not really so, unless you want to count what I consider to be inadequate documentation as a bug.

There is the following paragraph in the documentation:
By default, Natural uses single-fetch to retrieve data from Adabas databases. This default can be configured using the profile parameter MFSET. Values "ON" (multi-fetch) and "OFF" (single-fetch) define the default behavior. If MFSET is set to "NEVER", Natural always uses single-fetch mode and ignores any settings at statement level.

And, in the writeup of READ (for example) the following:

[MULTI-FETCH OF value] is not evaluated. The default processing mode is applied.

It is the latter two sentences which I totally misinterpreted. Why would a clause, like the MULTI-FETCH clause, not be evaluated. Well, I guessed, the MFSET parameter determines whether I will MULTI-FETCH or not. So far, I was correct. Then I tried to reconcile how Natural would ignore the “value” specified. I presumed (incorrectly) that there was a way to set the size of the MULTI-FETCH buffer other than with the value clause.

It turns out that there is no way to set the MULTI-FETCH buffer size. It is FIXED at a “value” of eight. Nonetheless, with a different value, like ten (or twenty-five), a program will compile and execute. I spent quite a bit of time with the documentation looking for a reference to the size of eight. No luck. If it is there, it is really buried somewhere.

As a Change Enhancement, I suggested that using the clause produce a compiler error message. I do realize this is somewhat extreme. I think that using the clause serves as excellent documentation. As an alternative I suggested something else rather extreme, namely, the compiler alter whatever number the user has in the clause to the default of eight. Perhaps, however, plans are afoot to change the way the clause works, thus eliminating the use of a default value.

If not, this is something you must be aware of. Code that works on the mainframe, where the value clause is functional, may not work on the PC where the clause is “window dressing”.

**Summary**

You must be careful with MULTI-FETCH, PRE-FETCH, and FASTPATH. Reading ahead, as they all do, can lead to problems when you are doing any form of update, namely UPDATE, DELETE, and STORE. Restarting a loop, as we did with ESCAPE TOP REPOSITION, can often solve potential problems. You may have to use the ISN start facility to effect proper positioning.

**Performance**

Thus far we have talked a lot about how MultiFetch works, but have not discussed the magnitude of performance improvement. On the one hand, it is easy to mentally compute the savings in terms of Adabas Calls. If I wish to compare the savings of a MultiFetch OF 10 versus No MultiFetch, it is simple, the No MultiFetch will have ten times the Adabas Calls of the MultiFetch OF 10. But, what does that mean in elapsed time? In CPU time?

Here is a simple program.

```
> +  Program     MULTI15  Lib XSTRO
0010 DEFINE DATA LOCAL
0020 1 MYVIEW VIEW OF EMPLOYEES
0030   2 NAME
0040   2 FIRST-NAME
0050   2 CITY
0060   2 LEAVE-DUE
0070 1 #CPU-START (I4)
0080 1 #CPU-ELAPSED (I4)
0090 1 #LOOP (I4)
0100 END-DEFINE
0110 *
0120 INCLUDE AATITLER
```
We now have a series of programs that differ in the MultiFetch parameter. First, we have a repeat of the last program (MultiFetch of 1200), but with 100 iterations rather than 5. This will help us compare times.

Next I made one slight change to the program. Instead of no MultiFetch, I used a MultiFetch of 1200. Since there are fewer than 1200 records on the file, I will only call Adabas once.

Here is our output.

If you compare this with the output above for no MultiFetch, the CPU ratio is twenty nine to one. The elapsed time ratio is rather difficult to compute, since the MultiFetch time was 0. However, based on the output of our next program (MULTI17), we can estimate the elapsed time as being .25 (which displays as zero). This would be a ratio of eight to one for the elapsed time.

34 ♦ Inside Natural
Note, compared with our last program, the elapsed time is the same, and the CPU time is also the same. Not surprising. With a MultiFetch of 500, as opposed to 1200, we are only saving two calls to Adabas. Even multiplied by 100 (number of loop iterations), this is quite insignificant.

Time for a more dramatic change, to a MultiFetch of 100.

Here is our output.

Here is our output.

Note that the elapsed time is up 20% (5 to 6) but at least some of that difference is probably due to rounding. The CPU time, however, has gone from 14 to 19, a forty percent differential.

Now let's double the number of Adabas Calls by switching to a MultiFetch of 50.

While the elapsed time is the same, the CPU time has gone from 19 to 24. Now for another doubling of the MultiFetch parameter.

However, I was given the boot for too many Adabas Calls. Okay, I reduced the number of iterations from 100 to 50.
The numbers above must be multiplied by two (10 and 58) to compare them with our earlier runs.

Compared with the MultiFetch of 25, the elapsed time is up 25% and the CPU time is up over 40%.

In summary, you can achieve some mind blowing performance improvements by simply specifying the use of MultiFetch. All of my experiments seem to indicate you should (assuming you are not ESCAPE'ing from loop, see below) make maximum use of the MultiFetch, using a parameter as large as will be permitted given the amount of data you require from each record.

When NOT to Multi-Fetch

Okay, we have seen how the efficiency of MultiFetch is a function of the MultiFetch parameter. We have also seen that UPDATE and DELETE turn MultiFetch off, and STORE can result in what might be construed as logic problems.

Time to view one more scenario where you should NOT use MultiFetch. As should be evident, since MultiFetch results in read ahead records, if you ESCAPE out of loops a lot, you will have “wasted” the benefits of decreased Adabas Calls by reading too many records. How serious might this be? Consider the following rather silly program:

As you can see, we are reading through the employees file in logical sequence by NAME. For every record, we promptly do a READ for records with the same NAME. For example, as you have seen above, there are nineteen SMITH records. Thus, the READ for each of these records would result in a need to read an additional nineteen records. Like I said, a pretty silly program, especially given the MultiFetch parameter of 100.

How expensive would this silliness be? Here is our output:

In a word, WOW. The elapsed time is up by a factor of eleven, although the CPU time has increased under twenty percent.

The lesson is clear. If you will be escaping out of loops, do not employ MultiFetch, especially with a large parameter.