NEWEV$ - Create New 32-bit Stacked Environment

The NEWEV$ routine is used to create a new 32-bit Stacked Environment (Universe). This allows a program to run another program(s) in a completely new Global System Manager Environment. This environment is created by saving and restoring a number of System Variables around a NEWEV$ call. Note that the new Global System Manager Environment created by NEWEV$ is not the same as the different environment provided when running a program in a different partition. Consequently, NEWEV$ should be used with great care.

All effort should be made to restructure an application to use the traditional EXEC verb to execute overlays. NEWEV$, with its inherent problems (see below), should only be used as a last resort.

1. Invocation

To create a new 32-bit Stacked Environment (Universe) code:

```
$SET tb
CALL NEWEV$ USING [par1] [par2] [par3] [par4] [par5] [par6] [par7]
```

where tb is a control block of the following format:

```
01     TB
  02   TBPROG      PIC X(8)         * NAME OF THE PROGRAM TO EXECUTE
  02   TBSAVE      PIC 9 COMP       * 1 = SAVE SCREEN IMAGE
                                    * 0 = DO NOT SAVE SCREEN IMAGE
```

and the optional par1 to par7 parameter are passed directly to the program to be executed in TBPROG.

2. STOP Codes and Exception Conditions

The following STOP codes may be generated by NEWEV$:

STOP code	Description
12501	An attempt to exceed the maximum number of stacked environments (10) has been attempted.
12502	Unable to allocate the 32-bit memory pages required for the new stacked environment.

The following EXIT codes may be returned by NEWEV$:

EXIT code	$$COND	Description
nnnaaa	aa	The exception returned by the program executed in the new stacked environment.

3. Programming Notes

The 32-bit NEWEV$ routine effectively replaces the 16-bit SBOVL$ routine which has no 32-bit equivalent.

All effort should be made to restructure an application to use the traditional EXEC verb to execute overlays. NEWEV$, with its inherent problems (see below), should only be used as a last resort.

The level of new environments is limited to 10. You may not have more than 10 NEWEV$ levels, that is the call stack level for NEWEV$ calls is 10.

Any program that is run using NEWEV$ must be a root program and not a dependent program.

The $$AREA variable is not saved by NEWEV$. If an application needs to save $$AREA it must do so itself from inside the calling program.

Any data pages allocated in a new environment (for instance by using FREEX$) will be de-allocated once execution has returned from this environment. Any shared data pages must be allocated at the root level.

The program called by NEWEV$ must not generate any STOP CODES. Generating a STOP CODE may damage the data being accessed by the underlying program (i.e. the program calling NEWEV$).

The following example **breaks** the rules:

```
PROGRAM A
DATA DIVISION
01    TB
 02   TBPROG      PIC X(8)    * PROGRAM TO EXECUTE
                  VALUE “B”
 02   TBSAVE      PIC 9 COMP  * = 1 SAVE SCREEN IMAGE
                  VALUE 1     * 0 OTHER WISE
*
77    P1          PIC X
PROCEDURE DIVISION
    $SET TB
    CALL NEWEV$ USING P1
EXIT
ENDPROG
ENDSOURCE

PROGRAM B
DATA DIVISION
LINKAGE SECTION
77    L-P1        PIC X
PROCEDURE DIVISION
ENTRY B USING L-P1
      STOP WITH 1
EXIT
ENDPROG
ENDSOURCE
```

Parameter Passing

Any parameters passed to the NEWEV$ overlay must not expect to return pointers which directly address data in the execution environment as on exit from this environment it will be de-allocated.
The following example **breaks** the rules:

```
PROGRAM A
DATA DIVISION
01    TB
 02   TBPROG      PIC X(8)    * PROGRAM TO EXECUTE
                  VALUE “B”
 02   TBSAVE      PIC 9 COMP  * = 1 SAVE SCREEN IMAGE
                  VALUE 1     * 0 OTHER WISE
*
77    P1          PIC PTR
```

```
PROCEDURE DIVISION
    $SET TB
    CALL NEWEV$ USING P1
EXIT
ENDPROG
ENDSOURCE

PROGRAM B
DATA DIVISION
77    DAYS  OCCURS 2 PIC X(3)
                  VALUE "MON"
                  VALUE "TUE"
LINKAGE SECTION
77    L-P1        PIC PTR
PROCEDURE DIVISION
ENTRY USING L-P1
      POINT L-P1 AT DAYS(2)
EXIT
ENDPROG
ENDSOURCE
```

System variables must not be passed as parameters via NEWEV$.

The following example **breaks** the rules:

```
PROGRAM A
DATA DIVISION
01    TB
 02   TBPROG      PIC X(8)     * PROGRAM TO EXECUTE
                  VALUE "B"
 02   TBSAVE      PIC 9 COMP   * = 1 SAVE SCREEN IMAGE
                  VALUE 1      * 0 OTHER WISE
*
PROCEDURE DIVISION
    $SET TB
    CALL NEWEV$ USING $$DATE
EXIT
ENDPROG
ENDSOURCE

PROGRAM B
DATA DIVISION
77    Z-LONG      PIC X(10)
LINKAGE SECTION

77    L-P1        PIC DATE

PROCEDURE DIVISION
ENTRY USING L-P1
      CALL DT-DL$ USING L-P1 Z-DATE
EXIT
ENDPROG
ENDSOURCE
```

In this example, $$DATE would need to be saved before the NEWEV$ call:

```
PROGRAM A
DATA DIVISION
01    TB
 02   TBPROG      PIC X(8)     * PROGRAM TO EXECUTE
                  VALUE "B"
 02   TBSAVE      PIC 9 COMP   * = 1 SAVE SCREEN IMAGE
                  VALUE 1      * 0 OTHER WISE
*
77    Z-D         PIC 9(6) C
PROCEDURE DIVISION
    MOVE $$DATE TO Z-D
    $SET TB
    CALL NEWEV$ USING Z-D
EXIT
ENDPROG
ENDSOURCE
```

```
PROGRAM B
DATA DIVISION
77    Z-LONG      PIC X(10)
LINKAGE SECTION

77    L-P1        PIC DATE

PROCEDURE DIVISION
ENTRY USING L-P1
      CALL DT-DL$ USING L-P1 Z-DATE
EXIT
ENDPROG
ENDSOURCE
```

Databases and Locking

In any normal application **all locks must be released** before a program calls NEWEV$, and as far as possible no locking should be done in the NEWEV$ overlay itself even though this is less serious. This is because confusion and lockouts could occur as in the examples below. There is no difference in outcome in locking behaviour, using NEWEV$, between programs accessing Global Speedbase databases and those accessing Pervasive SQL databases. It should be noted, however, that the locking behaviour of Pervasive databases records is not under our control and may differ between different versions of Pervasive SQL.

Example 1

A record may be locked in the program calling NEWEV$ as well as in the NEWEV$ overlay. This will not cause locking to fail but two locks will be issued. This is true even when the databases are in SQL format.

For example, the following would cause a problem:

```
PROGRAM A
ACCESS AA
DATA DIVISION
01    TB
 02   TBPROG      PIC X(8)     * PROGRAM TO EXECUTE
                  VALUE “B”
 02   TBSAVE      PIC 9 COMP   * = 1 SAVE SCREEN IMAGE
                  VALUE 1      * 0 OTHER WISE
*
PROCEDURE DIVISION
    CALL B$OPN USING “DB   “ “DBD” 0
    FETCH FIRST AAPRI RETRY -1
    MOVE “A” TO AAX1
    REWRITE AA
    FETCH FIRST AAPRI RETRY -1
    $SET TB
    CALL NEWEV$
    DISPLAY AAX1
    CALL B$CDB
EXIT
ENDPROG
ENDSOURCE

PROGRAM B
ACCESS AA
DATA DIVISION


PROCEDURE DIVISION
    CALL B$OPN USING “DB   “ “DBD” 0
    FETCH FIRST AAPRI   * THIS LOCK WILL NOT RETRY AND WILL SUCCEED
    ON NO EXCEPTION
       MOVE “B” TO AAX1
       REWRITE AA
    END
    CALL B$CLD
EXIT
```

```
ENDPROG
ENDSOURCE
```

Because the lock does not fail in program B, the record will be rewritten with “B” in AAX1 when the NEWEV$ B overlay is executed. Because the program A would expect the record to be locked, the record data in the record area in program A will not be modified. The display of AAX1 will therefore “A” even though the record in the database has the value of “B” in AAX1.

This is one explanation of why all locks must be released before entering a NEWEV$ call and why it is preferable not to issue locks in the program run by NEWEV$.

It is important to note that in the (hopefully never) occasion that you might ever need to leave a lock outstanding over a NEWEV$ call, and you need to leave the locks active, then when closing databases in the NEWEV$ overlay, B$CLD should be used and not B$CDB.   B$CLD is the same as B$CDB but it only releases locks held on the specific access channel and does not release all locks on the database. If you do not use the special B$CLD then when you exit the environment all locks on the database will be lost. This includes locks issued in other environments.

Example 2

A lock out for a very long time may occur for other user on the record.

For example, the following would cause a problem:

```
PROGRAM A
ACCESS AA
DATA DIVISION
01    TB
 02   TBPROG       PIC X(8)    * PROGRAM TO EXECUTE
                   VALUE “B”
 02   TBSAVE       PIC 9 COMP  * = 1 SAVE SCREEN IMAGE
                   VALUE 1     * 0 OTHER WISE
*
PROCEDURE DIVISION
    CALL B$OPN USING “DB   ” “DBD” 0
    FETCH FIRST AAPRI RETRY -1
    $SET TB
    CALL NEWEV$
    DISPLAY AAX1
    CALL B$CDB
EXIT
ENDPROG
ENDSOURCE

PROGRAM B
DATA DIVISION
77    Z-X1  PIC X

PROCEDURE DIVISION
    ACCEPT Z-X1
EXIT
ENDPROG
ENDSOURCE
```

If program B does and accept and the user leaves the screen sitting at an accept, or the NEWEV$ overlay “B” does some processing which takes a long time, the record locked in program A may remain locked for a very long time without it being visually obvious. This would cause other users to be locked out on this record for a very long time. All locks should therefore be released before NEWEV$ is called.

4. Examples
[EXAMPLE REQUIRED]

5. Copy-Books
None.

6. See Also

ENVNO$	Return 32-bit Stacked Environment (Universe) Number