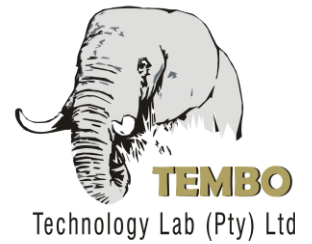


Pulling the (DB2) Trigger
A Solution Example

Author: Tommy Atkins, TEMBO Technology Labs

Abstract

This article is the follow-up to the original article on “Pulling the Trigger”. As promised, this article provides a set of code snippets and samples which have been documented in detail to provide an actual technical solution to the theoretical solution proposed in the first article.

Introduction

As it is considered as a “best practice” for each file in the database to have its own *BEFORE trigger program to manage all necessary data validations, it is necessary to customize each trigger program to match its associated file. For this reason the solution defined below is an example based on a pseudo FILEA and will need to be understood in its entirety and then customized to fit a specific file.

The snippets and samples in this document are from a “partially free” RPG IV (ILE) source, containing calculations in free-form and all other lines in fixed-form. This style is compatible with version 5 and upwards of the operating system and also with 8.5.1 and upwards of RDi.

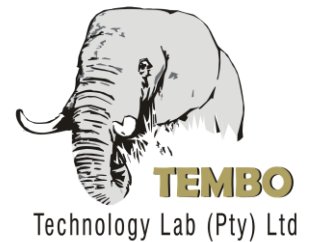
AO Open Source Project

All source components making up this complete solution “template” can be downloaded from the AO Website using the link http://www.adsero-optima.com/open-source.html. This source is provided under the standard MIT license in support of the international open source initiative.

The File (FILEA)

```
CREATE TABLE FILEA (
       STATUS CHAR(2)       NOT NULL DEFAULT,
       FIELD1 INTEGER       NOT NULL DEFAULT,
       FIELD2 CHAR(30)      NOT NULL DEFAULT,
       FIELD3 CHAR(1)       NOT NULL DEFAULT,
       FIELD4 SMALLINT      NOT NULL DEFAULT)
       RCDFMT FILEAR;
LABEL ON TABLE FILEA IS 'Dummy File for Trigger Template';
LABEL ON COLUMN FILEA (
           STATUS IS 'R/S                 '      ,
           FIELD1 IS 'Field 1             '      ,
           FIELD2 IS 'Field 2             '      ,
           FIELD3 IS 'Fld                 3'     ,
           FIELD4 IS 'Field 4             ')     ;
LABEL ON COLUMN FILEA (
           STATUS TEXT IS 'Record Status       '      ,
           FIELD1 TEXT IS 'Field 1             '      ,
           FIELD2 TEXT IS 'Field 2             '      ,
           FIELD3 TEXT IS 'Field 3             '      ,
           FIELD4 TEXT IS 'Field 4             ')     ;
```

This is the file on which the coding for this template will be based.



The Trigger Program

This trigger program should be attached to whichever *BEFORE events need some form of validation for the file.

The "H" specs are required as this is to be an ILE program and must always be run from the *CALLER activation group.

```
h dftactgrp(*no) actgrp(*caller) usrprf(*owner) aut(*use)
h bnddir('AAADIR') option(*nodebugio) debug
```

I addition a binding directory specification is required as the program makes use of a number of procedures bound into service programs (*SRVPGM), which must be defined in the nominated binding directory.

Trigger Program Parameters

The relational database management system (RDBMS) provides each and every trigger program call, with exactly the same two parameters. The parameter interface for these parameters are defined in the trigger program as follows;

```
d FILEA_B1        pr                  extpgm('FILEA_B1')
d  P                                  likeds(P1) options(*varsize)
d  PL                           10i 0 const
d FILEA_B1        pi
d  P                                  likeds(P1) options(*varsize)
d  PL                           10i 0 const
```

The first parameter (P) is a variable length parameter as it is dependent on the file to which the trigger is attached. The second parameter (PL) indicates the length of "P" and is seldom needed in my experience. To avoid defining "P" twice the LIKEDS keyword is used to point to a single definition (P1) of the parameter list. Click on the "Trigger Buffer" section of the following link to view a complete description of all the parameters provided by the RDBMS.

http://publib.boulder.ibm.com/html/as400/v4r5/ic2979/info/db2/rbafomstrzahftrb.htm

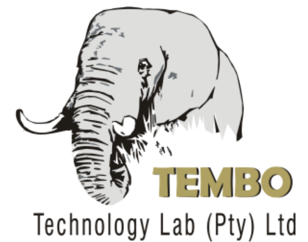
```
d P1              ds                  template qualified
d  PFName                 1     10
d  LibName               11     20
d  MbrName               21     30
d  Event                 31     31
d  Time                  32     32
d  CmtLock               33     33
d  Resvd1                34     36
d  CCSID                 37     40b 0
d  RRN                   41     44b 0
d  Resvd2                45     48
d  OldOS                 49     52b 0
d  OldLen                53     56b 0
d  OldNMOS               57     60b 0
d  OldNMLen              61     64b 0
d  NewOS                 65     68b 0
d  NewLen                69     72b 0
d  NewNMOS               73     76b 0
d  NewNMLen              77     80b 0
d  Resvd3                81     96
```

Note: Defining a parameter list using the LIKEDS keyword automatically qualifies the individual elements in the structure, therefore referring to the "PFName" would not be correct. "P.PFName" is the correct way to refer to this parameter element.

Record Definitions

Contained in the parameter structure provided by the RDBMS are either one or two record images from the file that initiated the event. The original record image, in the case of an *UPDATE or *DELETE event, and the new record image, in the case of *INSERT or *DELETE. The offsets to these images are in the fixed portion of the parameter list (first 95 bytes) and are based on the address of the first byte of the parameter list, in this case P1. The code which is used to setup these record overlays is described below. Note that the data structures for the original (O) and new (N) records are based on pointers.

```
d PP              s               *
```




```
d OP              s                *
d O               e ds                    extname(FILEA) based(OP) qualified
d NP              s                *
d N               e ds                    extname(FILEA) based(NP) qualified
```

Error Handling

The following section of code defines the prototypes for the 4 error handling procedures required by a *BEFORE trigger program. These "Error Handlers" are used by the trigger program to process and forward messages to the RDBMS as is required in order to notify the application of any problems.

These 4 "Error Handlers" are procedures defined inside various modules and then bound into a service program (*SRVPGM). Full documentation and all required source code for these handlers is available from the "Open Source" section of the AO Website at the same link as provided earlier on in this article.

```
 * Move *DIAG and Re-Send *ESCAPE Messages
d ERR03           pr
d  MT                            1    const options(*nopass)
 * Move *DIAG and *ESCAPE Messages as *DIAG
d ERR05           pr
d  MT                            1    const options(*nopass)
 * Send *DIAG Message
d ERR10           pr
d  MI                            7    const
d  MF                           10    const options(*omit:*nopass)
d  MD                          128    const options(*omit:*nopass)
d  MT                            1    const options(*nopass)
 * Send *ESCAPE Message
d ERR11           pr
d  MI                            7    const
d  MF                           10    const options(*omit:*nopass)
d  MD                          128    const options(*omit:*nopass)
d  MT                            1    const options(*nopass)
```

Check for Legacy

The most important part of this trigger program, and all the others that use the same model, is the ability to determine whether or not the application program which initiated the event is a "Legacy" program or a "New" program which is able to handle trigger program errors.

As this checking routine is likely to be used by many different trigger programs, I decided to create a procedure and imbed it, along with the I/O Server for the "Error Log File", into a Utility Service program from where it could be accessed by any trigger.

The following code defines the "CheckLegacy" procedure in the trigger program and allows it to be called as required. The procedure returns a Boolean indicator as a '1' if the program is a legacy program and a '0' if not. The actual name of the program is returned in the parameter of the call as this is required for the logging of an error in the case of a legacy program.

```
d CheckLegacy     pr              n   extproc('UTLSRV@001A')
d  Program                      10
```

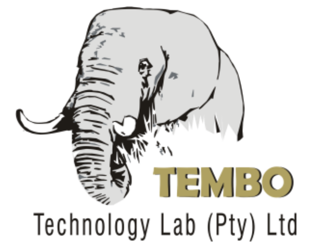
`The procedure inside the service program is named "UTLSRV@001A"`

Error Log File (ERRLOGF)

The following table is offered as a starting definition, but obviously would need to be modified for any additional requirements you may have. Changes to this file would also require changes to the ERRLOGF$ I/O Server module and the re-creation of the service program UTLSRV@@.

This is the file into which the error log records are written for all errors encountered by a trigger program when called from a legacy program.

```
CREATE TABLE ERRLOGF (
       FILENM CHAR(10)      NOT NULL DEFAULT,
       EVENTI CHAR(1)       NOT NULL DEFAULT,
       PROGNM CHAR(10)      NOT NULL DEFAULT,
```




```
       LOGDTS TIMESTAMP     NOT NULL DEFAULT CURRENT_TIMESTAMP,
       ERRMSG CHAR(7)       NOT NULL DEFAULT,
       ERRTXT CHAR(132)     NOT NULL DEFAULT,
       BEFORE VARCHAR(2048) NOT NULL DEFAULT,
       AFTER  VARCHAR(2048) NOT NULL DEFAULT)
       RCDFMT ERRLOGFR;
LABEL ON TABLE ERRLOGF IS 'Legacy App. Error Log File' ;
LABEL ON COLUMN ERRLOGF (
        FILENM IS 'File Name           ' ,
        EVENTI IS 'Event               Ind.' ,
        PROGNM IS 'Program             Name' ,
        LOGDTS IS 'Log Entry           Timestamp' ,
        ERRMSG IS 'Error               Message' ,
        ERRTXT IS 'Error Text          ' ,
        BEFORE IS 'Before Image        ' ,
        AFTER  IS 'After Image         ');
LABEL ON COLUMN ERRLOGF (
        FILENM TEXT IS 'File Name' ,
        EVENTI TEXT IS 'Event Ind.' ,
        PROGNM TEXT IS 'Program Name' ,
        LOGDTS TEXT IS 'Log Entry Timestamp' ,
        ERRMSG TEXT IS 'Error Message' ,
        ERRTXT TEXT IS 'Error Text' ,
        BEFORE TEXT IS 'Before Image' ,
        AFTER  TEXT IS 'After Image');
```

The following code is used to define the ERRLOGF structure and the I/O Server for the Error Log File.

```
     d ERRLOGFP        s                 *   inz(%ADDR(ERRLOGFR))
     d ERRLOGFR      e ds                    extname('ERRLOGF')
      * Write Log Record Service Procedure
     d ERRLOGF$        pr
     d  ParmPointer                      *   const
     d  RecPointer                       *   const
     d  MsgData                       128    const
```

Additional "D" Specs.

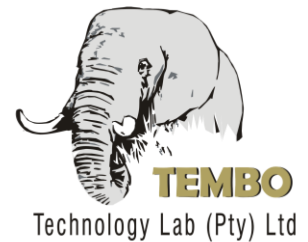
```
     d DFT             pr
     d VAL             pr
     d INS             pr
     d DLT             pr
     d UPD             pr
     d Legacy          s                 n
     d Error           s                 n   inz('0')
```

Trigger Program Mainline

This mainline portion of this trigger program controls the conditional processing of the trigger program, determining what the event was and what needs to be done. Remember that this trigger is only attached to the *BEFORE time of the file and therefore the only processing that needs doing is for the *BEFORE events.

Another thing worth remembering is that this program only processes one record during each call from RDBMS and therefore things such as repetitive (cycle) type processing are not a factor.

```
     /free
       monitor;    // Catch-All Monitor Group for Unexpected Errors
          Legacy = CheckLegacy(PROGNM); // Set Legacy Indicator & Program Name
          reset Error;                  // Reset Error Indicator
          PP = %addr(P);                // Set the Parameter Pointer to the Address of "P"
          select;
              when P.Event='1';      // Insert Event
                   NP = PP + P.NewOS;   // Set Pointer for New Record Image
                   DFT();               // Execute "Default"  Procedure
                   INS();               // Execute "Insert"   Procedure
                   VAL();               // Execute "Validate" Procedure
              when P.Event='2';      // Delete Event
```




```
               OP = PP + P.OldOS;   // Set Pointer for Original Record Image
               DLT();               // Execute "Default"  Procedure
          when P.Event='3';      // Update Event
               OP = PP + P.OldOS;   // Set Pointer for Original Record Image
               NP = PP + P.NewOS;   // Set Pointer for New Record Image
               DFT();               // Execute "Default"  Procedure
               UPD();               // Execute "Update"   Procedure
               VAL();               // Execute "Validate" Procedure
      endsl;
      if Error;                                 // At Least One Error has been found
         ERR05();                               // Percolate *DIAG and *ESCAPE Messages
         ERR11('ERR0035':'ERRMSGF':'FILEA_B0'); // Send *ESCAPE Message
      endif;                                    // Legacy Errors are Logged at Validation,
                                                // and do not generate a final *ESCAPE Error
   on-error;                        // Unexpected Error Condition.
      if Legacy;                                // Legacy Program
         ERRMSG = 'ERR0030';                    // Set Specific Error Message Number
         ERRLOGF$(PP:ERRLOGFP:'Unexpected');    // Log One Error Record for Legacy
      else;                                     // not Legacy
         ERR05();                               // Percolate *DIAG and *ESCAPE Messages
         ERR11('ERR0030':'ERRMSGF':'FILEA_B0'); // Send *ESCAPE Message
      endif;
   endmon;
   return;
  /end-free
```

"Defaults" Procedure

This procedure is included in *BEFORE trigger programs and is used as required to ensure that correct default values are inserted into fields not supplied by the application. This routine can also be used for such things as inserting the users id. into an appropriate field in the record and thereby overwriting the value supplied by the application program, as well as supplying values for fields not provided by the application. Below is an example.

```
p DFT             b
 /free
  monitor;
     if N.FIELD3 = *blank;
        N.FIELD3 = 'N';
     endif;
  on-error;
     ERR03();
  endmon;
 /end-free
p DFT             e
```

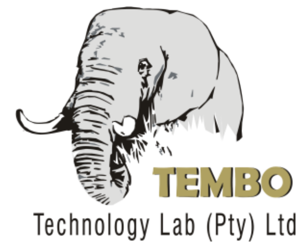
"Insert" Procedure

This procedure is specifically for the "Insert" event and contains code such as validations, the setting of defaults or anything else which is applicable to the inserting of a record into the database. The example below shows the setting of a timestamp for the creation of the record. This field is not incuded in the sample file.

```
p INS             b
 /free
  monitor;
     N.CREATE_DTS = %timestamp();
  on-error;
     ERR03();
  endmon;
 /end-free
p INS             e
```

"Delete" Procedure

The "Delete" procedure is generally not required in the *BEFORE trigger program, but is included for completeness.



It is used to provide delete specific coding which is applied before the record is deleted from the file. This procedure is more common in an *AFTER trigger program where database synchronization often follows the deletion of a record from a file.

```
p DLT            b
 /free
  monitor;
     // Deletion coding
  on-error;
     ERR03();
  endmon;
 /end-free
p DLT            e
```

"Update" Procedure

The "Update" procedure contains coding which is specific to the *UPDATE event of the trigger.

One of the most common uses of this procedure is to protect certain fields in the record from change by the application.

A good example of this would be the example field used in the "Insert" procedure above. As this CREATE_DTS field is populated on insert it should never be changed by any subsequent update.

The example code below shows how this is done.

```
p UPD            b
 /free
  monitor;
     N.CREATE_DTS = O.CREATE_DTS;
  on-error;
     ERR03();
  endmon;
 /end-free
p UPD            e
```

Validation Procedure

The "Validation" procedure is always executed by both the *INSERT and *UPDATE events in the trigger program because the validation rules need to be applied to all fields in both cases to ensure integrity.

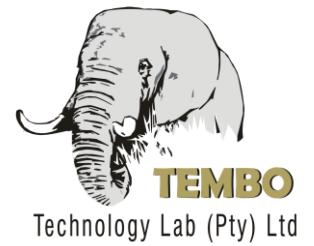
Any event specific validations are previously applied by the "Insert" and "Update" procedures. These event specific field validations are not repeated in the general "Validation" procedure.

The sample code below contains two examples of field validations, showing the different actions required depending on the setting of the "Legacy" indicator.

```
p VAL            b
 /free
  monitor;

     if %check('YN':N.FIELD3) <> 0;
        if Legacy;                            // Log Error Record for Legacy
           ERRMSG = 'ERR0014';                // Set Specific Error Message Number
           ERRLOGF$(PP:ERRLOGFP:'FIELD3');    // Log Error Record for Legacy
        else;
           Error = *on;
           ERR10('ERR0014':'ERRMSGF':'FIELD3'); // Send *DIAG Message to DBMS
        endif;
     endif;

     if N.FIELD4 < 1;
        if Legacy;                            // Log Error Record for Legacy
           ERRMSG = 'ERR0026';                // Set Specific Error Message Number
           ERRLOGF$(PP:ERRLOGFP:'FIELD4');    // Log Error Record for Legacy
        else;
           Error = *on;
           ERR10('ERR0026':'ERRMSGF':'FIELD4'); // Send *DIAG Message to DBMS
        endif;
```




```
          endif;
       on-error;
          ERR03();
       endmon;
      /end-free
     p VAL             e
```

Conclusion

This trigger program will need to be customized for each file to which it is added and will provide a small-step, no-risk method for tuning the business rules to enhance the integrity of the database.

It must be noted that the errors which are logged for legacy programs would in all probability be logged to the same ERRLOGF file for all files to which a trigger program of this style is attached. It is not recommended that each file logs its errors to a different log file as the management of such a situation would become very complicated.

Once the addition of this type of trigger to the database has been started, it is important that a process of viewing the ERRLOGF file and resolving the logged errors be put in place, to avoid a buildup of repeated errors and minimized benefits from the exercise.