TEMENOS™

VERSION ROUTINES

DateOf Issue	Version	Changes	By
2001	1.0	Initial	Alagammai Palaniappan
2002	1.1	jBASE Changes	Alagammai Palaniappan
2002	1.2	Infobasic Programming changes	Alagammai Palaniappan
July 2004	1.3	G14 changes	Alagammai Palaniappan
May 2006	1.4	Browser changes	Gerard Thomas
June 2007	2.0	Browser changes	Gerard Thomas

Table of Content

Table of Content 2
Introduction 3
Version routines 3
Auto Field Routine 3
ID Routine 6
Check Record Routine 7
After Unauth Routine 9
Before Auth Routine 12
The VERSION.CONTROL Application 14
Features Of The VERSION.CONTROL Application 14
Fields In The VERSION.CONTROL Application 14
Fields In The Version Application That Have An impact On The VERSION.CONTROL Application 15
EXC INC RTN 15
VERSION.TYPE 16

Introduction

VERSION is a standard T24 application, which allows users to create customized screens for input, authorization, display etc. It may also be used to automate the replacement of data on records. In fact T24 comes with many useful *VERSION* records for the majority of T24 applications. It is also possible to add local sub-routines, which can be used for field, input and authorization validation. These small programs will allow even greater customization of T24 by the user.

Version routines

There are three different types of routines that can be attached to a version. Each one of them is attached at different levels (i.e. - invoked at different stages during the life cycle of record) and therefore has a specific functionality. The different types of version routines available are:

Auto Field Routine

Validation Routine

Input Routine

Authorization Routine

Check Id Routine

Check Record Routine

After Unauth Routine

Before Auth Routine

Auto Field Routine

These routines are attached to specific fields in a version. They can be used to manipulate the content of the field before display They get executed after the id of the record is supplied by the user and the corresponding record is fetched from the file, but before the record can be displayed.(If the id entered is a new one then no record is fetched from the file). Therefore, using these subroutines, we could perform any special editing to the record before it is displayed to the user.

These routines are to be attached to the field ‘Auto New Content’ in the Version Application. The name of the routine should be prefixed with an @ symbol when attaching it to the field. These subroutines should have an entry in PGM.FILE with PGM.TYPE = 'S' and a corresponding entry in EB.API. These subroutines get executed for the associated field specified in ‘Autom Field No’.

Example 1

Create a local reference field in the Customer application called TOTAL.ACCOUNTS. This field is to contain the total number of accounts owned by a customer. When ever a Customer record is opened, the field TOTAL.ACCOUNTS needs to display the total number of savings accounts of that customer.

Fields that need to be part of the Customer version are

All mandatory fields of the Customer application

Local reference field TOTAL.ACCOUNTS

Solution 1

Step 1

Create a local reference field named TOTAL.ACCOUNTS and attach it to the Customer application.

Step 2

Create a Version for the Customer application with all the mandatory fields and the local reference field TOTAL.ACCOUNTS.

Step 3

Create a subroutine that will display the total number of accounts of a particular customer in the local reference field TOTAL.ACCOUNTS. If a new customer record is input then the calculation should not be performed and a value 0 needs to be populated in the field TOTAL.ACCOUNTS.

Algorithm

Obtain the id of the customer input by the user

Check to see if it is a new customer record

If it is a new customer record, then populate a value 0 in the field TOTAL.ACCOUNTS

If the customer id exists in the Customer file, then read the CUSTOMER.ACCOUNT file.

Count the total number of accounts that belong to that customer and populate the field TOTAL.ACCOUNTS.

ID.NEW

This is a Dynamic array that contains the id of the currently opened record. It is defined in I_COMMON file.

R.NEW

It is a dimensioned array that has been defined in the file I_COMMON. It holds the copy of the currently opened record in any application in T24. Always use '( )' brackets while referring to variables inside a dimensioned array as opposed to '<>" which we use for dynamic arrays. This dimensioned array has been dimensioned to contain 500 dynamic arrays. The definition for R.NEW in the I_COMMON file is as follows.

R.NEW(C$SYSDIM) where C$SYSDIM = 500

R.OLD

R.OLD is a dimensioned array that has been defined in the I_COMMON file. It holds a copy of an authorized record when it is opened for amendment. The definition for R.OLD in the I_COMMON file is as follows.

R.OLD(C$SYSDIM)

ID.OLD

This is a dynamic array that holds the id of the record held in R.OLD

R.NEW.LAST

R.NEW.LAST is a dimensioned array that will hold a copy of an unauthorized record when it is opened for amendment. The definition for R.NEW.LAST in the I_COMMON file is as follows.

R.NEW.LAST(C$SYSDIM)

ID.NEW.LAST

Is a dynamic array that holds the id of the record held in R.NEW.LAST

```
*Version auto new content routine that calculates the total number of
*accounts for a customer and displays it in the field TOTAL.ACCOUNTS
*which is a local reference field
SUBROUTINE V.TRG.AUT.CNT.RTN
$INSERT I_COMMON
$INSERT I_EQUATE
$INSERT I_F.CUSTOMER.ACCOUNT
$INSERT I_F.CUSTOMER

GOSUB INIT
GOSUB OPENFILES
GOSUB PROCESS
RETURN

INIT:
FN.CUS.ACC = 'F.CUSTOMER.ACCOUNT'
F.CUS.ACC = ''
Y.CUS.ID = ID.NEW
R.CUS.ACC = ''
Y.CUS.ACC.ERR = ''
RETURN

OPENFILES:
CALL OPF(FN.CUS.ACC,F.CUS.ACC)
RETURN

PROCESS:
IF R.OLD(1) = ‘’ THEN ; *If it is a new customer record
```

```
      R.NEW(EB.CUS.LOCAL.REF)<1,29> = 0 ; * No calculation is required
END
ELSE
      CALL F.READ(FN.CUS.ACC,Y.CUS.ID,R.CUS.ACC,F.CUS.ACC,Y.CUS.ACC.ERR)
      IF Y.CUS.ACC.ERR NE '' THEN ; * If there are no record for the
                                    customer in the CUSTOMER.ACCOUNT file
            R.NEW(EB.CUS.LOCAL.REF)<1,29> = 0
      END
      ELSE
            R.NEW(EB.CUS.LOCAL.REF)<1,29> = DCOUNT(R.CUS.ACC,FM)
      END
END
RETURN
END
```

Step 4

Compile and catalogue the subroutine and make an entry in PGM.FILE and EB.API. Attach the routine to the field Auto New Content (Prefix the routine name with an @ symbol) in the version and specify the Local Reference Field (TOTAL.ACCOUNTS) field in the field Autom Field No.

ID Routine

This routine, as the name implies is used to manipulate the id of the record. It gets executed as soon as the id of the record is entered. Any special manipulation can be done on the id using this routine.

It is attached to the field ID.RTN (prefix the routine name with an '@' while attaching it to the Version) in the Version application and needs have an entry in the PGM.FILE with the type set to 'S' and a corresponding entry in EB.API.

Example 2

Write a routine that will prefix 'TEM.TRG' to the id of any enquiry when created/opened to denote that it was created/amended as a part of the Temenos Training program. This should not happen for ids of enquiries that begin with a '%' or a Enquiry-LIST. Also ensure that the length of the id does not exceed 30 characters. If it does then an error "Temenos Trg restricts ID length to 30 Characters" should be displayed to the user

Solution 2

Step 1

Create a version for the Enquiry application with all the mandatory fields.

Step 2

Create a subroutine that will get executed as soon as the id of an enquiry is entered and prefixes the id of the enquiry with 'TEM.TRG' provided it not a % Enquiry or an Enquiry-List.

APPLICATION

T24 common variable defined in I_COMMON that holds the current application that has been opened by the user.

```
* This routine can be used to manipulate the id of the record. This sample
* routine should be attached to a version of ENQUIRY , to the field ID.RTN
* This will prefix the string 'TEM.TRG' to any id input unless it is
* the id of a % Enquiry or a Enquiry-LIST which are default enquiries by
* the name of the underlying application and should not be modified.
SUBROUTINE V.TRG.CHECK.ID.RTN
$INSERT I_COMMON
$INSERT I_EQUATE
$INSERT I_F.ENQUIRY
IF APPLICATION NE 'ENQUIRY' THEN RETURN
IF COMI THEN    ;* ID.NEW would not have been set now
      IF COMI[1,1] NE '%' AND FIELD(COMI,'-',2)[1,4] NE 'LIST' THEN
      COMI = 'TEM.TRG.':COMI
      IF LEN(COMI) GT 30 THEN
         E = "Temenos Trg restricts ID length to 30 Characters"
         CALL ERR
         MESSAGE = 'REPEAT'
         V$ERROR = 1
      END
      END
END
RETURN
END
```

Step 3

Compile and catalogue the subroutine, make entries in the PGM.FILE (with the type set to 'S') and EB.API file and attach it to the field ID RTN(prefix the routine name with an '@' when attaching it to the version) in the Version application.

Check Record Routine

This routine is called after the ID is supplied and validated. This will get invoked when the following functions I(Input), D(Delete), A(Authorise) and R(Reverse) are used. These routines could be used to alter the field attributes.

It is attached to the field CHECK.REC.RTN (prefix the routine name with an '@' while attaching it to the Version) in the Version application and needs have an entry in the PGM.FILE with the type set to 'S'.

Example 3

Write a subroutine that will restrict any user trying to open a Live Customer Record which was not input and authorized by him and will display an error message "Access Restricted"

Solution 3

Step 1

Create the necessary subroutine

OPERATOR

This is a dynamic array defined in the I_COMMON file that contains the id of the currently signed on user.

Algorithm

1. Check if the record being opened is a new Customer record. If so, processing should not be done and the user should be able to open the record.
2. Check if the record is an unauthorized record. If so, processing should not be done and the user should be able to open the record.
3. Extract the value of the field INPUTTER. There could be multiple inputters for a single record.
4. Check if the current user is one among the inputters. If not, then display the error message "Access Restricted"
5. If the current user is not one among the inputters , then check if he is the authoriser. If not, then display the error message "Access Restricted" else allow the user to continue.

```
*Version Check Record Routine - which will allow a customer record to
* be opened only if it has been input and authorized by the user who
* is trying open that record.
SUBROUTINE V.TRG.CHECK.REC.RTN
$INSERT I_COMMON
$INSERT I_EQUATE
$INSERT I_F.CUSTOMER
IF NOT(R.NEW(EB.CUS.INPUTTER)) THEN RETURN    ;* Could be a New Record
IF R.NEW(EB.CUS.RECORD.STATUS) THEN RETURN    ;* Could be an
                                               *unauthorized record
   THIS.OPERATOR.OK = 0
   INPUTTERS = R.NEW(EB.CUS.INPUTTER)
   NO.OF.INPUTTERS = DCOUNT(INPUTTERS,@VM)
   IF NO.OF.INPUTTERS = 0 THEN
        IF FIELD(INPUTTERS,'_',2) = OPERATOR THEN THIS.OPERATOR.OK = 1
```

```
    END
    ELSE
        LOOP
        REMOVE INPUTTER FROM INPUTTERS SETTING MORE.INPUTTERS
        WHILE INPUTTER : MORE.INPUTTERS DO
        INPUT.OPERATOR = FIELD(INPUTTER,'_',2)
        IF INPUT.OPERATOR EQ OPERATOR  THEN
           THIS.OPERATOR.OK = 1
           INPUTTERS = ''
        END
        REPEAT

        IF NOT(THIS.OPERATOR.OK) THEN    ;* He wasnt one of the Inputters
           AUTHORISER = R.NEW(EB.CUS.AUTHORISER)
           IF FIELD(AUTHORISER,'_',2) NE OPERATOR THEN ; * He was'nt the
                                                           authoriser
               E = 'Access Restricted'
               CALL ERR
               MESSAGE = 'REPEAT'
               V$ERROR = 1
           END
        END

RETURN
END
```

Step 2

Compile and catalogue the subroutine, make entries in PGM.FILE and EB.API , and attach the subroutine prefixed with an '@' symbol in the field CHECK.REC in the Customer Version.

After Unauth Routine

This routine is called while committing a new record or existing record after making the necessary changes. It is called after the Version Input Routine. At this stage the record would have been written into the $NAU file. Therefore it is imperative to remember that any change made to R.NEW will not reflect in the record. If any change needs to be made to the record in the $NAU file at this stage, then a separate write needs to be executed. Do not call JOURNAL.UPDATE as this is taken care by core T24. If the variable E is set then the error message is displayed and the record commit is cancelled. The difference between Input routine and this routine is that in the former we need to call the Override and if it is cancelled then the operation is cancelled, but in the later we set the value of variable E to abort the operation.

It is attached to the field AFTER.UNAU.RTN (prefix the routine name with an '@' while attaching it to the Version) in the Version application and needs have an entry in PGM.FILE and EB.API.

Example 4

There is a requirement where in, when a transaction is **committed** using a version (called Parent) of the FT application, the user should be taken to another version (called Child) in order to input the charges (if any).

Analysis

If the CHILD version were to be opened on the authorization of the record in the PARENT version, then the NEXT VERSION field in the Version application can be used. Since the CHILD version needs to be invoked on commit of a record in the PARENT version, a routine is needed for this. This routine needs to be executed after the record is written onto the FT's unauthorized file. This is similar to the functionality available in the LD application, where in when a transaction is committed using the LD application, a version of the LD.SCHEDULE.DEFINE is opened to input the schedules.

Solution 4

Step 1

Algorithm

Check if the application is FT

Check if the version used is the version to which we are attaching this subroutine

(The above2 checks are not absolutely necessary. Incase this routine is going to be made common to all versions of the FT application and if you still want to execute this routine only for the FT,PARENT version, then this check is required. We will learn how to make a routine common to all versions of a particular application later in this section.)

Check if the function used is 'I'.

If the function used is 'I', then execute the next version (FT,CHILD).

PGM.VERSION

This is a common variable defined in I_COMMON that contains the name of the currently opened version.

Example

If the version CUSTOMER,INPUT is the currently opened version, then the variable APPLICATION would contain CUSTOMER and the variable PGM.VERSION would contain ,INPUT.

INPUT.BUFFER

This is a common variable defined in I_COMMON which is capable of containing any input that a user wishes to execute from the command line.

Example

If the user wishes to open the CUSTOMER application in the input mode and input a new customer, the user would normally ensure that he is in the command line, then type 'CUSTOMER I' and then press F3 to obtain a new id. If this were to be done programmatically, then we need to store the key strokes in a common variable called INPUT.BUFFER. If any value is stored in the INPUT.BUFFER then the SYSTEM would not wait for input from the user. It would just execute the contents of INPUT.BUFFER.

INPUT.BUFFER = C.U":" CUSTOMER I ":C.F

C.U : Ctrl U Enter -> Used to get to the command line

C.F : Ctrl F Enter(F3) -> Used to obtain a new id

V$FUNCTION

This is a common variable defined in I_COMMON that contains the function that is being currently chosen by the user.

Desktop

```
*Version After Unauthorize Routine - Opens a version by name
*FUNDS.TRANSFER,CHILD when a record is committed in the
*FUNDS.TRANSFER,PARAENT version.
SUBROUTINE V.TRG.AFTER.UNAUTH.RTN
$INSERT I_COMMON
$INSERT I_EQUATE
$INSERT I_F.FUNDS.TRANSFER

IF APPLICATION = "FUNDS.TRANSFER" AND PGM.VERSION = ",PARENT" AND
                                                V$FUNCTION = "I"
       INPUT.BUFFER = C.U:" ":APPLICATION:",CHILD":" I ":C.F
END
RETURN
END
```

Browser does not support the functionality of INPUT.BUFFER. Browsers users may therefore try out the following code:

```
*manual input. Browser only
SUBROUTINE V.TRG.AFTER.UNAUTH2.RTN
$INSERT I_COMMON
$INSERT I_EQUATE
$INSERT I_F.FUNDS.TRANSFER

IF V$FUNCTION = "I" THEN
      CALL EB.SET.NEW.TASK("FUNDS.TRANSFER,CHILD I F3")
END
RETURN
END
```

Step 2

Compile and catalogue the subroutine, make an entry in the PGM.FILE with the type set to 'S' and attach it to the version in the field AFTER.UNAUTH.RTN prefixed with a '@' symbol.

Before Auth Routine

This routine is called during the authorization of an INAU record. It is called before the version's Auth routine is executed. If value of E is set then record authorization is cancelled. Though both Auth routine and Before Auth Routine are invoked at Authorization Stage, the main difference between Before Auth and Auth routine is that Before Auth is invoked before a F.WRITE is made to the live file. This means that, even at this stage we can manipulate the value of any field. However, AUTH.RTN gets invoked after F.WRITE is made to the file. This means that changes made to R.NEW will not reflect in the record unless an F.WRITE is made explicitly in the case of Auth routine. This is something that we have already discussed. Do not call JOURNAL.UPDATE, as it will be taken care of by Core T24.

These routines need to have entries in the PGM.FILE and EB.API applications and should be attached to the field Before Auth in the Version application prefixed with an '@' symbol.

Example 5

There is a requirement, where in, while authorizing a customer record, the user should be given the option to fill in any special comments that he wishes to store along with the customer record like "Special customer " etc which can be used at a later date for any special processing or could be printed along with the statements sent to that particular customer. Therefore, write a routine that will prompt the user to enter free text while authorizing a customer record. Once the text is entered by the user, the data needs to be updated in the field TEXT in the Customer application. If no text is entered by the user then the message "No text entered. Proceeding with other processing" should be displayed and the record should get authorized.

Solution 5

Step 1

Create a subroutine that will prompt the user for the text and update the TEXT field in the Customer application.

Algorithm

Display a message to the user to input free text for a customer

Once the data is entered, update the Customer file

Else display a message “No text entered. Proceeding with other processing”

Desktop

```
*Version Before Auth Routine – Accepts text from the user when a customer
*record is authorized and update the customer record with that data in the
*field ‘TEXT’.
*This example works only with desktop
SUBROUTINE V.TRG.BEFORE.AUTH.RTN
$INSERT I_COMMON
$INSERT I_EQUATE
$INSERT I_F.CUSTOMER

    Y.COLUMN = 8        ;* Not of any significance in Desktop
    Y.ROW = 22          ;* Not of any significance in Desktop
    N1 = '35.1'
    T1 = 'A'
    INP.MSG = 'Enter text for customer'
    CALL INP(INP.MSG,Y.COLUMN,Y.ROW,N1,T1)
    IF COMI = '' THEN
        CRT "No text entered. Proceeding with other procesing"
    END
    ELSE
        R.NEW(EB.CUS.TEXT)<1,-1> = COMI
    END
RETURN
END
```

Note : Please note that this is very similar to the authorization routine that we have already discussed. In the authorization routine we did an explicit write into the live file. Where as over here, we have only updated R.NEW and the contents would get reflected in the live file.

Step 2

Compile and catalogue the subroutine, make an entries in PGM.FILE (with the type set to ‘S’) and EB.API. Attach it to the field BEFORE.AUTH.RTN in the Customer version prefixed with an ‘@’ symbol.

The VERSION.CONTROL Application

As discussed earlier the routines that can be attached to a Version add functionality to the Version. Consider a scenario wherein the details of all transactions put through the FUNDS.TRANSFER application are to be sent to an external system. In which case, ideally, a subroutine would be attached at the authorization stage to do the above. We need to ensure that the subroutine is attached to every available version of FUNDS.TRANSFER application. There is a possibility that the version could be created without attaching that subroutine. To avoid such a situation, a utility by name VERSION.CONTROL is made available to us in T24 through which we could specify subroutines that need to be invoked for a set of versions and even for the original application itself.

Features Of The VERSION.CONTROL Application

VERSION.CONTROL can be used to attach subroutines at the application level. The field Non Version Run in the VERSION.CONTROL record is used to specify this.

VERSION.CONTROL can also be used to specify subroutines that need to be invoked for a set of versions. All versions for which these subroutines need to be invoked should have a value of 'YES' in the field EXC.INC.RTN at the Version level. In this case, the id of the VERSION.CONTROL application must be the name of the application to which the versions belong to. Eg- CUSTOMER

All Versions for which these subroutines need to be invoked could be be grouped together by specifying the same Version Type at the Version level. In this case the id of the VERSION.CONTROL record would be APPLICATIONNAME,VERSIONTYPE.

All fields in Version Control have corresponding fields in the Version record. Subroutines attached at the version level have a higher priority over subroutines attached at the version control level (. i.e. subroutines at the version level will be executed first)

Fields In The VERSION.CONTROL Application

Auto Field Name **Auto Old Cont** **Auto Field Rtn**	These 3 fields are similar to the associated multi value set in the Version application – 'Field No – Aut Old Content – Aut new Content'
Field Name **Validation Rtn**	They are similar to the associated multi value set in the Version application 'Validation Fld – validation Rtn'.
Input Rtn	Similar to the input routine in the Version application
Auth Rtn	Similar to the authorization routine in the Version application
Id Rtn	Similar to the id routine in the Version application
Check Rec Rtn	Similar to the check record routine in the Version application

After Unau Rtn Similar to the after unauth routine in the Version application

Before Auth Rtn Similar to the before auth routine in the Version application

Non Version Run This field can contain any one of the following values – 'Y or N or BLANK'. If this field is set to 'Y', then the routines specified in the VERSION CONTROL record for a specific application would get triggered for the application itself . If this field is set to 'N' or is left blank, then the version routines specified in VERSION.CONTROL would only get triggered for the versions of the application.

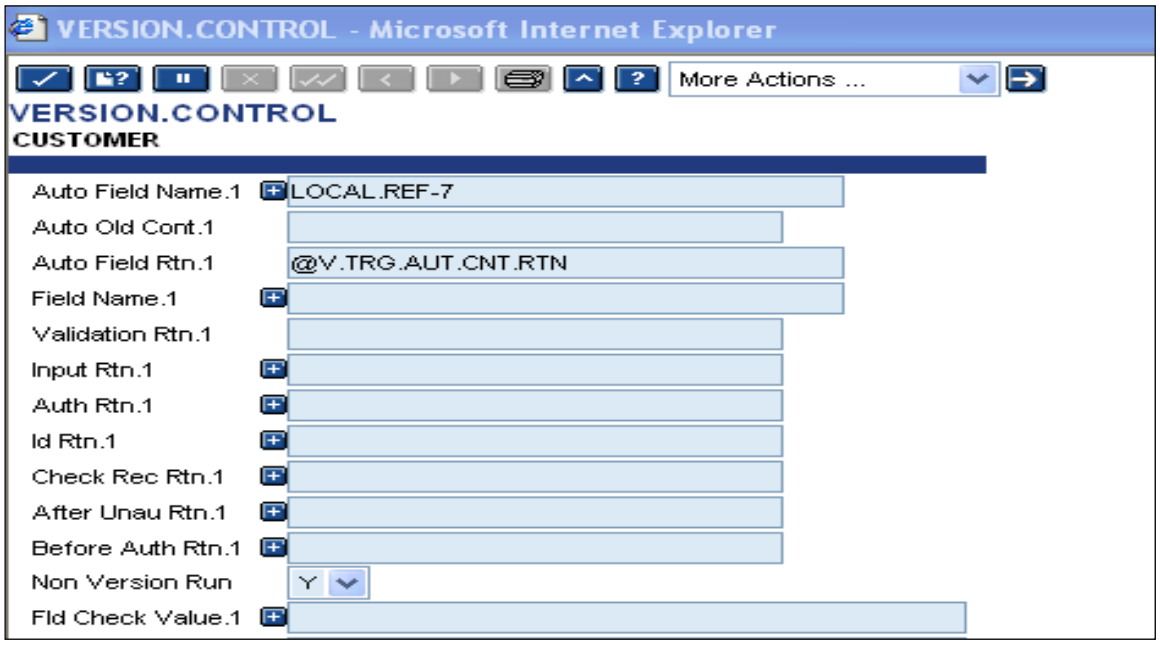
The versions where the routines may be triggered are specified using the fields EXC INC RTN & VERSION.TYPE in the versions themselves.

Example 6

Create a version control record for the customer application itself. This version control record should incorporate the functionality given in example 1, ie – the ability to populate the total accounts field with the number of accounts held by each customer

Solution 6

Create a version control record as shown below with the id as CUSTOMER AND Non Version Run field set to Y.



Fields In The Version Application That Have An impact On The VERSION.CONTROL Application

EXC INC RTN

This field can contain any one of the following values – 'YES or NO or Blank'. When this field is set to YES, the routines specified in the corresponding VERSION.CONTROL record get invoked for this version

VERSION.TYPE

This field can contain alpha numeric data of maximum 3 characters. It is used to group together a set of versions for the purpose of applying a common VERSION.CONTROL.

These set of versions would have common version routines specified in a VERSION.CONTROL record. The id of this VERSION.CONTROL record should have been given as valid application name followed by version type

Example : CUSTOMER,TRG (Where TRG is the version type).

Scenario I

If we create a record in the VERSION.CONTROL application with the id as CUSTOMER then for all versions based on the CUSTOMER application that have the field VERSION.TYPE set to 'blank', and the field EXC.INC.RTN set to 'YES', the routines specified in the VERSION.CONTROL record with the id as CUSTOMER would get invoked.

Scenario II

If we create a record in the VERSION.CONTROL application with the id as 'CUSTOMER,TRG', then for all versions based on the CUSTOMER application having the field VERSION.TYPE set to 'TRG' and the field EXC.INC.RTN set to 'Y', the routines specified in the VERSION.CONTROL application with the id 'CUSTOMER,TRG' would get executed.

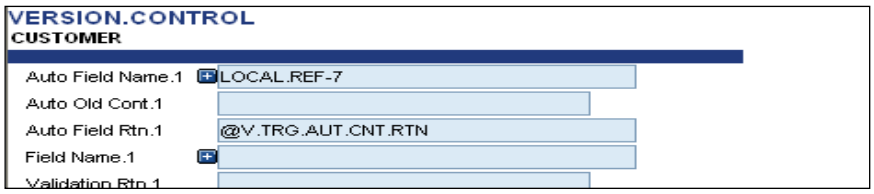
Let us illustrate this two scenarios with examples

Example 7 (illustrates Scenario 1)

Create a version control record for the customer, so that versions inheriting from this could incorporate the functionality given in example 1, ie – the ability to populate the total accounts field ,with the number of accounts held by each customer

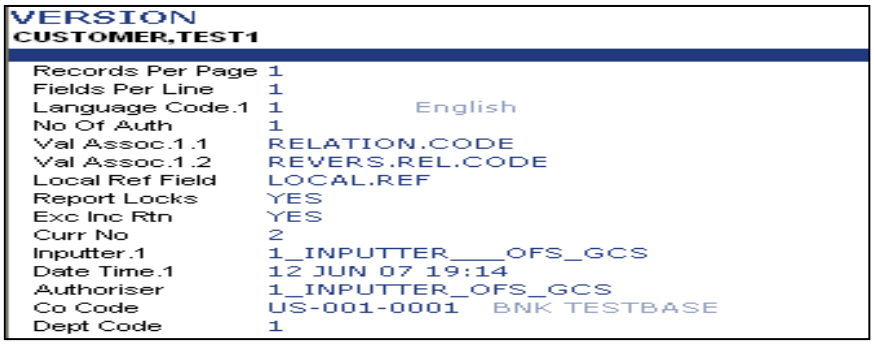
Solution

1. Create a VERSION.CONTROL record with id as CUSTOMER



Note that the id of the VERSION.CONTROL record is the application name CUSTOMER.

2. Create a version to inherit from the this VERSION.CONTROL record, by setting it's Exc Inc Rtn field to YES.

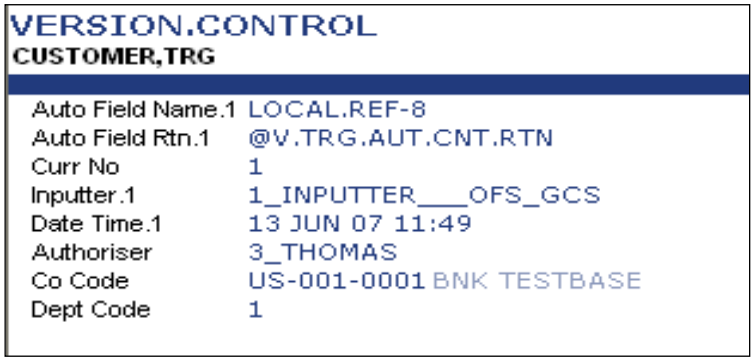


Example 8 (illustrates Scenario 2)

Create a version control group called TRG for customer. Create a version to inherit the functionality from this version group.

Solution

1.Create a VERSION.CONTROL record having an id CUSTOMER,TRG

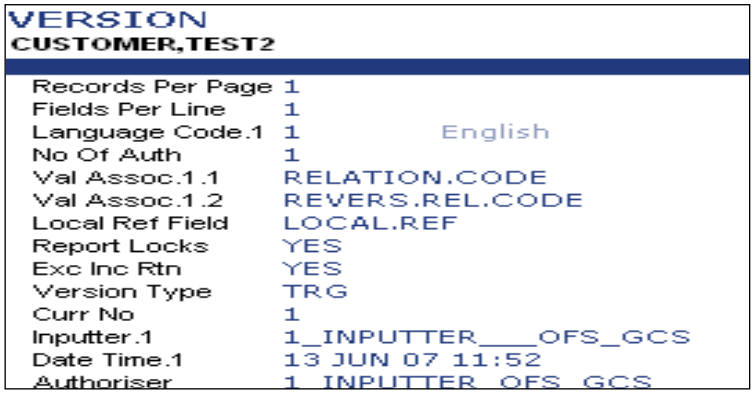


VERSION.CONTROL
CUSTOMER,TRG

Auto Field Name.1	LOCAL.REF-8
Auto Field Rtn.1	@V.TRG.AUT.CNT.RTN
Curr No	1
Inputter.1	1_INPUTTER___OFS_GCS
Date Time.1	13 JUN 07 11:49
Authoriser	3_THOMAS
Co Code	US-001-0001 BNK TESTBASE
Dept Code	1

Note that the id of the VERSION.CONTROL record follows the format of application, version-type. i.e. TRG is the version type.

2. Create a version as shown below to inherit the functionality of the VERSION.CONTROL record, ie. the Auto Field routine to populate the local reference field.



VERSION
CUSTOMER,TEST2

Records Per Page	1
Fields Per Line	1
Language Code.1	1 English
No Of Auth	1
Val Assoc.1.1	RELATION.CODE
Val Assoc.1.2	REVERS.REL.CODE
Local Ref Field	LOCAL.REF
Report Locks	YES
Exc Inc Rtn	YES
Version Type	TRG
Curr No	1
Inputter.1	1_INPUTTER___OFS_GCS
Date Time.1	13 JUN 07 11:52
Authoriser	1_INPUTTER_OFS_GCS

This version 'inherits' from VERSION.CONTROL record CUSTOMER,TRG because the Exc Inc Rtn field is set to YES and the Version type field is set to TRG.

Additional Information

R.NEW is a dimensioned array that will hold a copy of the currently opened record.

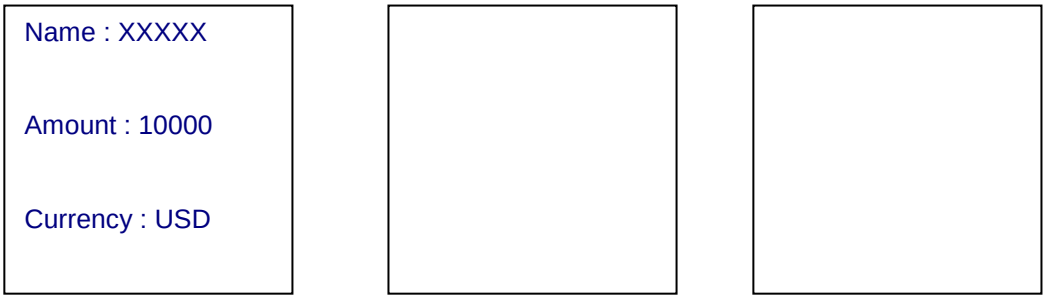
R.OLD is a dimensioned array that will hold the copy of an authorized record when it is opened for amendment.

R.NEW.LAST is a dimensioned array that will hold a copy of an unauthorized record when it is opened for amendment.

Following are the values R.NEW, R.NEW.LAST and R.OLD will contain in the following circumstances

Input a new transaction

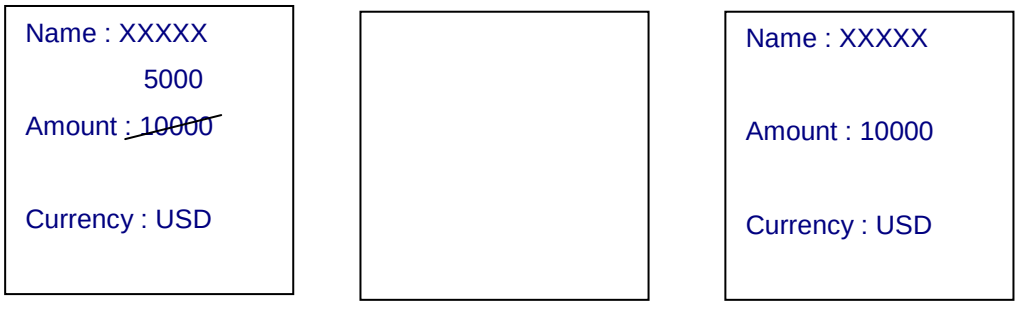
ID : 1



R.NEW (Copy of what you will see on screen) R.OLD (Will be empty) R.NEW.LAST (Will be empty)

Amend a unauthorized record

ID : 1



R.NEW R.OLD R.NEW.LAST

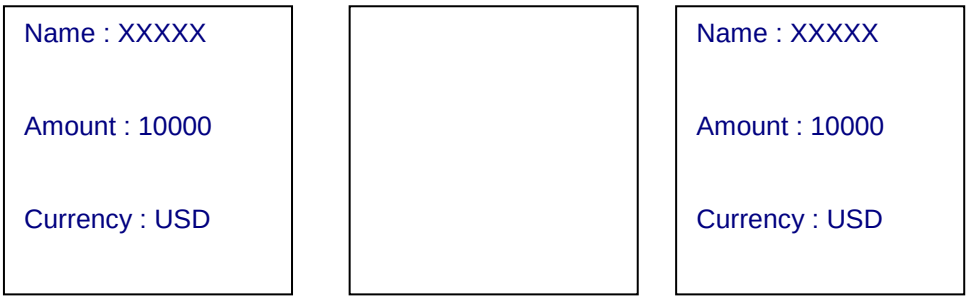
When an unauthorized record is amended, limits might be applied. But remember that if a user just changes the name or the value of any field that might not affect LIMIT we need not worry about Limit updation. It is only when fields like Amount,Date,Currency etc are changed that we need to worry about hitting limits. In the above scenario when we open an unauthorized record for amendment, a

copy of the record that we have opened gets populated in arrays R.NEW as well as R.NEW.LAST. Note that when we change a value in the unauthorized record, the value gets updated in R.NEW only and not in R.NEW.LAST. So when we amend the LIMIT we need to rollback the original amount that is there is R.NEW.LAST and update the new value that is there is R.NEW.

Assume that the original Limit amount was 10000$.When the transaction was input for the first time the limit would have got reduced to 9000$.When the transaction is amended the old value of 10000$ has to be added back to the limit amount and the new value of 5000$ ahs to be reduced from the limit amount.The process of adding back an amount to the Limit is referred to as “DEL” and the process of reducing the new amended amount to the Limit is referred to as “DEL”.To sum it up we need to do a VAL of 1000$ and a DEL of 5000$ to the limit amount.

Delete an unauthorized record

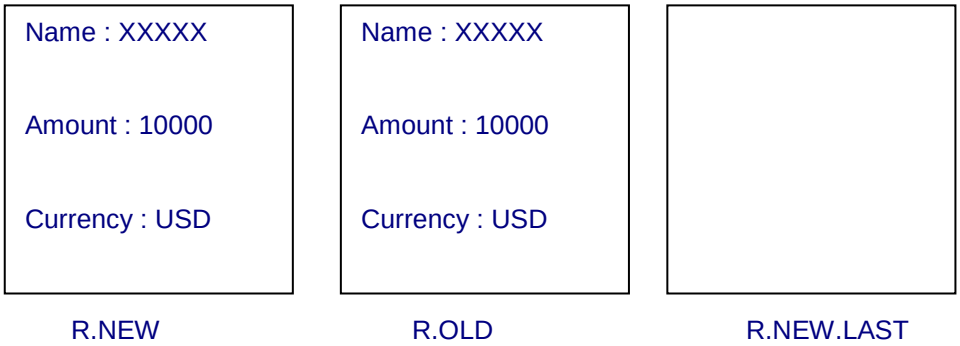
ID : 1



Since an unauthorized record has been opened, R.NEW.LAST also contains a value. When an unauthorized record is deleted, apart from reversing the accounting entries we also need to rollback the Limit updation that would have already happened. All that we need to do now is to add back the amount that we reduced from the original limit amount when we actually input the transaction .So we need to do a “DEL” of 10000$ to the original limit amount which would bring back the limit amount to the original position as it was before the transaction was input.

Reverse an authorized record

ID : 1



Now that we have opened an authorized record, apart from the values of the record getting populated in R.NEW the values would also go to R.OLD. Now when we reverse an authorized record all that we need to do is NOTHING. When an authorized record is reversed, it goes in to the INAU status and at the INAU status also the Limit has to be hit.

Amend an authorized record

ID : 1