# Stored Procedures

The following section lists the Stored Procedures we expect to need ordered based on the priorities generated by Mark Roke. Several areas of the system have yet to be analysed in details so more SP’s may be added as the project continues.

The current development status of each stored procedure is shown against it in brackets.

## Posting

### PostToHistory [Done - Catherine]

Replaces the LPost\_To\_Hist2 function in Exchequer which updates the history for all types.

*Note: Also effects Check All Stock, Check All Accounts, GL Views, Check Account Stock Analysis.*

### PostToYearToDate [Done - Catherine]

Replaces the LPost\_To\_CYTDHist2 function in Exchequer which creates the YTD History.

*Note: Also effects Check All Stock, Check All Accounts, GL Views, Check Account Stock Analysis.*

## Stock Take with Locations

### Filter [Done - ND]

Copies the Bin Location on the Stock-Location record to the Stock record for the Stock Take.

### Freeze [Done - ND]

Updates the balance and stock-level information on stock items for a specified location.

## Revalue Stock [Cancelled]

Updates the balances on the stock items for a specified location from history.

According to CS this was cancelled because the Posting stored procedure improved it enough.

## Check All Stock

### DeleteLinks [Done]

Called by Re\_CalcStockLevels to remove any FIFO Entries for a Stock Item.

MH 04/06/2008: Re-used the existing DeleteRows call.

### DeleteAuditHistory [Done]

Called from Re\_CalcStockLevels to delete or reset the posted stock levels

MH 04/06/2008: Re-used the existing DeleteRows call.

### Stk\_CheckLocHist [Done - ND]

Called from Re\_CalcStockLevels to run through the Stock-Location records resetting the balances. Can only be used on Non-Purged systems, purged systems will have to stick with the existing code.

### LCheck\_NoteNo [Done - ND]

Called from TCheckStk.ProcessFromCheck this routine runs through the Notes for the Stock Item and returns the next available note number.

## Check Account

NOTE: Check All Accounts just calls Check Account repeatedly.

### tdPostExLocal.LResetAuditHist [Done - ND]

Resets the Sales/Purchase/Cleared balances in the History to zero for a specified account. In latest profiles this consists of ~84% elapsed time for Check All Accounts and ~43% elapsed time for Check Account.

*Note: Also effects Check Account Stock Analysis.*

### LRemove\_MatchPay [Done]

Called to delete matching records, in latest profile results this consists of 23.5% elapsed time for Check Account.

MH 10/06/2008: Re-used the existing DeleteRows call.

## Telesales

### CalcStockCost [Cancelled]

The performance problem with the list in Telesales is caused by the calculation of the Cost Price column (which defaults to not being visible!).

MH 09/07/08: This was cancelled because after 5+ days investigation Chris had over 900 lines of code (with more to come) that would have to be converted into a stored procedure. It was viewed as unrealistic to try and get this done for the v6.00 SQL release and it was decided to remove the Cost column in Telesales and to have an edit field at the bottom to contain the cost.

## Add Transaction

### Stock\_AddCustAnal [Specified – see below ( PR)]

This procedure adds or removes a Customer/Stock analysis record and is called once for each transaction line added and twice for each line updated. The time taken for this is noticeable in the toolkits, so they may benefit from a stored procedure.

## GL Views

A possible Stored Procedure for GL Views is under discussion, however reorganising the routine promise greater performance improvement, although a SP for LFillBudget would be beneficial on top of that.

### Update\_NomViewActual [Cancelled – see below]

This routine creates the Posted History Balances for the GL Views.

MH: The routine itself was actually rewritten as it was calling itself recursively to update the parent headings which meant that the history rows had to be read multiple times. The new routine caches the headings in a TList and applies the history row to them all in one go reducing the number of reads on the history table. In a Demo Data test the reads dropped from 80,546 to 47,062.

### LfillBudget [Done]

Called from the revised Update\_NomViewActual routine this takes approx 40% of the elapsed time for new GL Views – not used once the GL View History records have been created so this is a low priority item.

NOTE: This routine is covered in the cover for Update\_NomViewActual.

### Reset\_ViewHistory [Done – ND]

This is called to reset the history balances when an existing GL View is being refreshed – doesn’t really take any time for new views as there are no records to update, but took approx 59% of the elapsed time when refreshing an existing view – should be relatively easy to convert to a stored procedure.

### Total Profit to Date [Done]

This is called from multiple places throughout Exchequer, either to update the financial figures for a specific range of records, or to return the current balance.

### Remove Previous Commitments [Done]

This is part of the Daybook Posting run, and deletes all the records for previous commitments, ready for them to be recreated.

### Partial Unpost History [ Specified – see below]

This is part of the Partial Unpost by Period routine, and recalculates the History balances for transactions within the period range.

# Other Procedures

These are other SQL calls which aren’t stored procedures.

## Reset Custom [Done]

Deletes the users Custom Positions/Colours

# Check All Stock – Stk\_CheckLocHist

This stored procedure can only be used if the data has not been purged, if the data has not been purged then the existing code should be run.

The existing Stk\_CheckLocHistory routine is called for each stock code, but I believe that one stored procedure applying to all stock codes can be written to minimise overhead.

The SP will work as follows:-

For each Stock Code:-

For each stock-location record for the stock code:-

Delete the history records where hiExClass = Calc\_AltStkHCode(StkType) and hiCode = CalcKeyHist(StkFolio,lsLocCode) (see DeleteAuditHist below).

Reset the Quantity fields (see Stk\_CheckLocalHist below).

The Stock-Location records for a particular Stock Code and Location can be retrieved with the following query:-

Select Cast(SubString(varCode1, 2, 16) As VarChar), Cast(SubString(varCode1, 18, 3) As VarChar), \*

From ROOT01.MLocStk

Where (RecPFix = 'C') And (SubType = 'D')

And (Cast(SubString(varCode1, 2, 16) As VarChar) = 'ALARMSYS-DOM-1')

And (Cast(SubString(varCode1, 18, 3) As VarChar) = 'AAA')

## Original Code (De-Obfuscated)

Procedure DeleteAuditHist (Code : AnyStr;

KLen : Integer;

UseReset : Boolean);

Const

Fnum = NHistF;

Keypth = NHK;

Var

KeyS : AnyStr;

Locked: Boolean;

LAddr : LongInt;

sCompany : ShortString;

sWhere : ANSIString;

Begin

// Do direct call to SQL Server to delete all rows in one go

sCompany := GetCompanyCode(SetDrive);

sWhere := '(' + GetDBColumnName('History.Dat', 'f\_ex\_class', '') + ' = ' + IntToStr(Ord(Code[1])) + ') And ' +

'(SubString(' + GetDBColumnName('History.Dat', 'f\_code', '') + ', 2, ' + IntToStr(KLen-1) + ') = ' + StringToHex(Copy(Code,2,KLen-1)) + ')';

If (Syss.AuditYr <> 0) Then

Begin

// Add clause for Purge

sWhere := sWhere + ' And (' + GetDBColumnName('History.Dat', 'f\_yr', '') + ' > ' + IntToStr(Syss.AuditYr) + ')';

End; // If (Syss.AuditYr <> 0)

DeleteRows(sCompany, 'History.Dat', sWhere);

end;

Procedure TStkChkFrm.Stk\_CheckLocHist(StkCode : Str20;

StkType : Char;

StkFolio : LongInt;

Var SetOBal : Boolean);

Const

Fnum2 = MLocF;

Keypath2 = MLK;

Var

KeyChk,KeyS,

KeyS2,KeyChk2 : Str255;

LOk,Locked : Boolean;

IdR : IDetail;

LAddr : LongInt;

Begin

SetOBal:=BOff;

KeyChk2:=PartCCKey(CostCCode,CSubCode[BOff])+StkCode;

KeyS2:=KeyChk2;

Status:=Find\_Rec(B\_GetGEq,F[Fnum2],Fnum2,RecPtr[Fnum2]^,Keypath2,KeyS2);

While (StatusOk) and (CheckKey(KeyChk2,KeyS2,Length(KeyChk2),BOff)) do

begin

With MLocCtrl^,MStkLoc do

Begin

KeyChk:=Calc\_AltStkHCode(StkType)+CalcKeyHist(StkFolio,lsLocCode);

{\* Remove posted history \*}

DeleteAuditHist(KeyChk,Length(KeyChk),BOff);

LOk:=GetMultiRecAddr(B\_GetDirect,B\_MultLock,KeyS2,KeyPath2,Fnum2,BOn,Locked,LAddr);

{\* Reset Location values \*}

If (LOk) and (Locked) then

Begin

lsQtyInStock:=0;

lsQtyAlloc:=0;

lsQtyOnOrder:=0;

lsQtyPicked:=0;

lsQtyAllocWOR:=0.0;

lsQtyIssueWOR:=0.0;

lsQtyPickWOR:=0.0;

lsQtyReturn:=0.0;

lsQtyPReturn:=0.0;

Status:=Put\_Rec(F[Fnum2],Fnum2,RecPtr[Fnum2]^,Keypath2);

Report\_BError(Fnum2,Status);

Status:=UnLockMultiSing(F[Fnum2],Fnum2,LAddr);

End;

End;

Status:=Find\_Rec(B\_GetNext,F[Fnum2],Fnum2,RecPtr[Fnum2]^,Keypath2,KeyS2);

end;

end;

# Check All Stock – LCheck\_NoteNo

This routine is called by the Check Account and Check Stock functions to determine the next available line number.

If there are no Notes for the specified Account/Stock then it returns 1 else it should return the highest LineNumber in use + 1.

For Accounts:-

Select Max(LineNumber)

from ROOT01.ExchqChk

Where (RecPFix = 'N') And (SubType = 65)

And (Cast(SubString(ExchqChkCode1, 2, 6) As VarChar) = 'ABAP01')

For Stock:-

Select Max(LineNumber)

from ROOT01.ExchqChk

Where (RecPFix = 'N') And (SubType = 83)

And (Cast(Cast(Cast(Reverse(SubString(ExchqChkCode1, 2, 4)) As Char(4)) As Binary(4)) As int) = 313) – 313 = Stock Folio Number

Note: There is also a Check\_NoteNo function in NoteSupU which duplicates the functionality with a few extras – there is an outstanding query with Eduardo over this as both routines are used by Check All Stock.

# Check Account / Check All Accounts - tdPostExLocal.LResetAuditHist

This routine is called from the Check Account and Check All Accounts processes to reset the balances on the accounts history records.

The first character of the specified Code maps onto the History.hiExClass column and the remainder of the Code maps onto the History.hiCode column.

Expected SP Parameters:-

HistClass : Byte

HistCode : String

HistCodeLen : Byte;

DelCCDepHist : Boolean;

PurgeYear : Byte // Syss.AuditYr

## Original Code

// Public variable within parent object

DelCCDepHist : Boolean;

Function tdPostExLocal.IsCCDepHist : Boolean;

Begin

Result:=BOn;

If (DelCCDepHist) then

Result:=(LNHist.Code[6]>#32) or (LNHist.Code[7]>#32) or (LNHist.Code[8]>#32);

end;

Function AfterPurge(PYr, Mode : SmallInt) : Boolean;

Begin

Result:=(PYr>Syss.AuditYr) or (Syss.AuditYr=0);

end;

// Procedure to Reset all History Records Matching a give Code and after purge year

Procedure tdPostExLocal.LResetAuditHist (Code : AnyStr;

KLen : Integer);

Const

Fnum = NHistF;

Keypth = NHK;

Var

LocalCode, KeyS : AnyStr;

LOk, Locked : Boolean;

B\_Func : Integer;

Tries,TryMax, LAddr : LongInt;

// If we are deleting CC/Dep records, check it is one by making sure pos 6-8

// does not contain spaces. v5.50

// A bug was found were G/L code 1092 is ASCI is D $04 00 00. The prefix for

// deps is D so the actual was being picked up by mistake \*}

Begin

KeyS:=Code; LocalCode:=Code;

TryMax:=1000;

LStatus := Find\_RecCId(B\_GetGEq, LocalF^[Fnum], Fnum, LRecPtr[Fnum]^,

Keypth, KeyS, ExClientId);

While (LStatus=0) and (CheckKey(Code,KeyS,KLen,BOn)) do

With LNHist do

Begin

Tries:=0; B\_Func:=B\_GetNext;

If (AfterPurge(Yr,0)) and (IsCCDepHist) then

Begin

LOk:=LGetMultiRec(B\_GetDirect,B\_MultLock,KeyS,KeyPth,Fnum,BOn,Locked);

If (LOk) and (Locked) then

Begin

LGetRecAddr(Fnum);

Sales:=0; Purchases:=0; Cleared:=0;

Repeat

LStatus:=LPut\_Rec(Fnum,KeyPth);

Inc(Tries);

Until (Not (LStatus In [84,85])) or (Tries>TryMax);

LReport\_Berror(Fnum,LStatus);

LStatus:=LUnLockMLock(Fnum);

end

else

If (Not LOk) then

LStatus:=4

else

LStatus:=84;

LReport\_BError(Fnum,LStatus);

end;

LStatus:=LFind\_Rec(B\_Func,Fnum,Keypth,KeyS);

end;

end;

# GL Views

## Source Code

  BankNHCode                                   =  'B';

  CtrlNHCode                                      =  'C';

  StkStkQCode                    =  #239;   { Qty only History Code }

  StkBillQCode                   =  #236;   {  "    "     "      "  }

  StkDLQCode                     =  #247;   { Qty only History Code }

  JobGrpCode     =  'K';

  JobPhzCode     =  'Z';

  JobJobCode     =  'J';

  ViewHedCode                    =  '9';

  ViewBalCode                    =  '8';

**YTDSet**   :  CharSet =  [BankNHCode,CtrlNHCode,StkStkQCode,StkBillQCode,

StkDLQCode, JobGrpCode,JobPhzCode,JobJobCode,

ViewHedCode,ViewBalCode];

  CustHistGPCde  =  'W';   {\* Constant Code of Customer GP History \*}

  StkGrpCode                     =  'G';    { Stock Header Line }

  StkStkCode                     =  'P';    { Stock Item Code }

  StkDescCode                    =  'D';    { Descriptive Look-up Line }

  StkBillCode                    =  'M';

  StkDListCode                   =  'X';    { Stock Item De-Listed }

  PLNHCode                                 =  'A';

{ Types with Non C/F YTD History }

**YTDNCFSet**      :  CharSet   =  [CustHistGPCde,StkGrpCode,StkStkCode,

StkDescCode,StkBillCode,StkDListCode,PLNHCode];

  YTD     =  255;    { YTD Weighting for history }

  YTDNCF  =  254;    { YTD Weighting for History, but not carry forward bal type }

{ == Function to return full uncapitalised STR Longint Equivalent code == }

Function FullNHCode(CCode : Str20) : AnyStr;

Begin

FullNHCode:=LJVar(Ccode,NHCodeLen); // NHCodeLen = 20

end;

{ ========= Function to Return Part Nominal History Key ========== }

Function PartNHistKey(Typ : Str5;

Code : Str20;

Cr : Byte) : Str30;

Begin

PartNHistKey:=Typ+FullNHCode(Code)+Chr(Cr);

end;

{ ========= Function to Return Full Nominal History Key ========== }

Function FullNHistKey(Typ : Str5;

Code : Str20;

Cr,Yr,

Pr : Byte) : Str30;

Begin

FullNHistKey:=PartNHistKey(Typ,Code,Cr)+Chr(Yr)+Chr(Pr);

end;

{ ========= Proc to Return Nhist Key from Full Key ========= }

Procedure Extract\_NHistfromNKey(NKey : Str255;

Var NHR : HistoryRec);

Var

Point : Byte;

Begin

With NHR do

Begin

Point:=1;

ExClass:=NKey[Point];

Point:=Point+{Fpos[Ntyp2No].Len}1;

Code:=Copy(NKey,Point,NHCodeLen);

Point:=Point+NHCodeLen;

Cr:=Ord(NKey[Point]);

Inc(Point);

Yr:=Ord(NKey[Point]);

Inc(Point);

Pr:=Ord(NKey[Point]);

Inc(Point);

end; {With..}

end; {Proc..}

Function tdPostExLocal.LLast\_YTD(NType : Char;

NCode : Str20;

PCr,PYr,PPr : Byte;

Fnum,NPath : Integer;

Direc : Boolean) : Boolean;

Var

KeyChk, KeyS : Str255;

B\_Func : Integer;

TmpBo : Boolean;

CEStatus: Integer;

Begin

CEStatus:=0;

TmpBo:=BOff;

If (Not Direc) then

B\_Func:=B\_GetLessEq

else

B\_Func:=B\_GetGEq;

KeyChk:=PartNHistKey(NType,NCode,PCr);

KeyS:=FullNHistKey(NType,NCode,PCr,PYr,PPr);

CEStatus:=LFind\_Rec(B\_Func,Fnum,NPath,KeyS);

TmpBo:=(

(CEStatus=0)

and

(CheckKey(KeyChk,KeyS,Length(KeyChk),BOn))

and

(

((LNHist.Yr<=PYr) and (Not Direc))

or

((LNHist.Yr>=PYr) and (Direc))

)

and

((LNHist.Pr=PPr) or (Direc))

);

{\* Check for the exact YTD as this returns next History

which will be a period \*}

If (TmpBo) and (Direc) then

TmpBo := LCheckRecExsists(FullNHistKey(NType,NCode,PCr,LNHist.Yr,PPr),

NHistF,NHK);

LLast\_YTD:=TmpBo;

end; {Func..}

(MH: I would have thought that Ladd\_NHist would be in Catherine’s SP somewhere.)

Procedure tdPostExLocal.LAdd\_NHist(NType : Char;

NCode : Str20;

PCr,PYr,PPr : Byte;

Fnum,NPath : Integer);

Var

LastPurch, LastSales, LastCleared: Real;

Tries, TryMax, N : LongInt;

Begin

LastPurch:=0; LastSales:=0; LastCleared:=0;

N:=0; Tries:=0; TryMax:=1000;

{$B-}

If (PYr>0) then

Begin

If (PPr=YTD) and (LLast\_YTD(NType,NCode,PCr, AdjYr(PYr,BOff), PPr, Fnum,

NPath, BOff)) then

With LNHist do

Begin

N:=(PYr-Yr);

If (N>1) then {\* Add YTD In between \*}

For N:=AdjYr(Yr,BOn) to AdjYr(PYr,BOff) do

LAdd\_NHist(NType,NCode,PCr,N,PPr,Fnum,NPath);

LastPurch:=Purchases;

LastSales:=Sales;

If (NType In [StkStkQCode,

StkDLQCode,

StkBillQCode,

JobGrpCode,

JobJobCode]) then

{\* This mod necessary, as otherwise qty adjustments

which transend years missed \*}

LastCleared:=Cleared;

end;

{$B+}

With LNHist do

Begin

LResetRec(Fnum);

ExClass:=NType;

Code:=FullNHCode(NCode);

Cr:=PCr; Yr:=PYr; Pr:=PPr;

Sales:=LastSales; Purchases:=LastPurch;

Cleared:=LastCleared;

Repeat

LStatus:=LAdd\_Rec(Fnum,Npath);

Inc(Tries);

Until (Not (LStatus In [84,85])) or (Tries>TryMax);

LReport\_BError(Fnum,LStatus);

end; {With..}

end

else

Begin

If (Debug) then {Break point}

MessageBeep(0);

end;

end; {Proc..}

{ ===== Proc to determine what type of YTD (if any) a heading type contains == }

Procedure tdPostExLocal.LHed\_YTDType(MoveCode : LongInt;

NTyp : Char;

Fnum,

Keypath : Integer;

Var YTDOk,

YTDNCFOk : Boolean);

Var

FoundOk : Boolean;

KeyS, KeyChk : Str255;

Begin

YTDOk:=BOff;

YTDNCFOk:=BOff;

FoundOk:=BOff;

KeyChk:=NTyp+FullNomKey(MoveCode);

KeyS:=KeyChk;

LStatus:=LFind\_Rec(B\_GetGEq,Fnum,KeyPath,KeyS);

While (LStatusOk) and

(CheckKey(KeyChk,KeyS,Length(KeyChk),BOn)) and

(Not FoundOk) do

With LNHist do

Begin

FoundOk:=(Pr In [YTD,YTDNCF]);

If (Not FoundOk) then

LStatus:=LFind\_Rec(B\_GetNext,Fnum,KeyPath,KeyS);

end; {While..}

If (FoundOk) then

With LNHist do

Begin

YTDOk:=(Pr=YTD);

YTDNCFOk:=(Pr=YTDNCF);

end;

end; {Proc..}

Procedure tdPostExLocal.LFillBudget(Const Fnum,

Keypath : Integer;

Var Mode : Byte;

NKey : Str255);

Var

n, NeedYTD : Byte;

HedYTD, HedYTDNCF, LOk, Locked: Boolean;

TSales, TPurch, TCleared, Rnum, TmpBudget, TBudget2, BV1,BV2 : Double;

Tries,TryMax, MoveCode, RecAddr: LongInt;

GStr : Str5;

LocalNHist : HistoryRec;

Begin

NeedYTD:=0; HedYTD:=BOff; HedYTDNCF:=BOff;

LOk:=BOff; Locked:=BOff;

Tries:=0; TryMax:=1000;

Begin

LResetRec(Fnum);

Extract\_NHistfromNKey(Nkey,LNHist);

With LNHist do

For n:=1 to Syss.PrinYr do

If (Not LCheckExsists(FullNHistKey(ExClass, Code, Cr, GetLocalPr(0).CYr, n),

Fnum, KeyPath)) then

LAdd\_NHist(ExClass,Code,Cr,GetLocalPr(0).CYr,n,Fnum,KeyPAth);

With LNHist do

Begin

If (ExClass=NomHedCode) then

Begin

MoveCode:=0;

GStr:=Copy(Code,1,4);

Move(GStr[1],MoveCode,4);

LocalNHist:=LNHist;

LHed\_YTDType(MoveCode,ExClass,Fnum,Keypath,HedYTD,HedYTDNCF);

LNHist:=LocalNHist;

end;

If (ExClass In YTDSet) or (HedYTD) then

NeedYTD:=YTD

else

If (ExClass In YTDNCFSet) or (HedYTDNCF) then

NeedYTD:=YTDNCF;

end;

If (NeedYTD>0) then {Set YTD if Necessary}

With LNHist do

Begin

If (Not LCheckExsists(FullNHistKey(ExClass,Code,Cr,GetLocalPr(0).CYr,

NeedYTD),

Fnum,KeyPath)) then

Begin

LocalNHist:=LNHist;

(MH: Chris tells me you have already done a cut-down version of the following function for the stock freeze SP.)

Rnum := LTotal\_Profit\_to\_Date(ExClass,

Code,

Cr,

GetLocalPr(0).CYr,

Pr,

TPurch,

TSales,

TCleared,

TmpBudget,

TBudget2,

BV1,

BV2,

BOn);

LNHist:=LocalNHist;

LAdd\_NHist(ExClass,Code,Cr,GetLocalPr(0).CYr,NeedYTD,Fnum,KeyPAth);

LOk:=LGetMultiRec(B\_GetDirect,B\_MultLock,NKey,KeyPath,Fnum,BOn,Locked);

If (LOk) and (Locked) then

Begin

LGetRecAddr(Fnum);

Sales:=TSales; Purchases:=TPurch; Cleared:=TCleared;

Repeat

LStatus:=LPut\_Rec(Fnum,Keypath);

Inc(Tries);

Until (Not (LStatus In [84,85])) or (Tries>TryMax);

LReport\_Berror(Fnum,LStatus);

LStatus:=LUnLockMLock(Fnum);

end;

end;

end;

Mode:=4

end;

end; {Proc..}

Function CalcCCKeyHistP(NomFolio : LongInt;

lm : Boolean;

lc : Str10) : Str255;

Begin

CalcCCKeyHistp:=CalcCCKeyHist(NomFolio,PostCCKey(lm,lc));

end;

Function CalcCCDepKey(IsCC : Boolean;

CCDep : CCDepType) : Str10;

Begin

If (Not EmptyKeyS(CCDep[BOff],ccKeyLen,BOff)) and

(Not EmptyKeyS(CCDep[BOn],ccKeyLen,BOff)) and

(Syss.PostCCDCombo) then

Begin

Result := LJVar(CCDep[IsCC],ccKeyLen) +

Chr(1+Ord(IsCC))+

LJVar(CCDep[Not IsCC],ccKeyLen);

end

else

Result:=CCDep[IsCC];

end;

{ ====== Function to set full NHist Key for Nom View ====== }

Function TCheckNom.Set\_NVHist( NRec : NomViewRec;

Mode, AutoPBF : Boolean) : Str255;

Var

IsCC : Boolean;

NCode : Char;

HYr : Byte;

ComKey, CCKey : Str255;

Begin

IsCC:=BOff; HYr:=0;

HYr:=GetLocalPr(0).CYr;

Case Mode of

BOff : With NRec.NomViewLine do

Begin

NCode:=ViewType;

Result:=FullNHistKey(NCode,PostNVIdx(NomViewNo,ABSViewIdx),0,HYr,1);

end;

BOn : With MTEXLocal^,NRec.NomViewLine do

Begin

If (LinkGL<>0) and LGetMainRecPos(NomF,FullNomKey(LinkGL)) then

Begin

If (

(Trim(LinkCCDep[BOn])<>'')

and

(Trim(LinkCCDep[BOff])<>'')

)

or

(Trim(LinkCCDep[BOn])<>'') then

Begin

CCKey:=CalcCCDepKey(BOn,LinkCCDep);

IsCC:=BOn;

end

else

CCKey:=LinkCCDep[BOff];

NCode:=LNom.NomType;

{$IFDEF SOP}

If (IncCommit) then

ComKey:=CommitKey

else

{$ENDIF}

ComKey:='';

Result := FullNHistKey(Ncode, ComKey + CalcCCKeyHistP(LNom.NomCode,

IsCC,CCKey),

0, HYr,1);

end;

end;

end; {case..}

end; {Func..}

{ ==== Procedure to Update all Parent with lower budget records ======== }

Procedure TCheckNom.Update\_NomViewActual(LowKey,

HiKey : Str255;

PostView : NomViewRec;

ProMode,

AutoPBF : Boolean);

Const

Fnum = NHistF;

Keypath = NHK;

Var

n, YTDType, Loop : Byte;

CDpCode: Str20;

KeyChk, KeyHChk, KeyS, KeyHS : Str255;

FoundOk,FLoop, BeenWarned, LOk, Locked : Boolean;

LowHist, GetHist, HedHist, HiHist : HistoryRec;

LastStatus: Integer;

TmpKPath, TmpStat: Integer;

TmpRecAddr: LongInt;

PBalBF : Double;

LUP : tPassDefType;

Begin

LastStatus:=0; Loop:=0; n:=0; PBalBF:=0;

Locked:=BOff;

Extract\_NHistfromNKey(Lowkey,LowHist);

Extract\_NHistfromNKey(Hikey,HiHist);

LUP:=UserProfile^;

With MTEXLocal^ do

Begin

With LowHist do

KeyChk:=ExClass+Code;

KeyS:=KeyChk;

LStatus:=LFind\_Rec(B\_GetGEq,Fnum,Keypath,KeyS);

While (LStatusOk) and

(CheckKey(KeyChk,KeyS,Length(KeyChk),BOn)) and

(Not THreadRec^.THAbort) do

Begin

Begin

If (THreadRec^.THAbort) and (Not BeenWarned) then

Begin

ShowStatus(3,'Please Wait, finishing current actual.');

BeenWarned:=BOn;

end;

GetHist:=LNHist;

{$IFDEF DBD}

If ((GetHist.Purchases+GetHist.Sales)<>0.0) and (AutoPBF) then

MessageBeep(0);

{$ENDIF}

FLoop:=BOff; FoundOk:=BOff;

TmpKPath:=KeyPath;

TmpStat:=LPresrv\_BTPos(Fnum,TmpKpath,LocalF^[Fnum],TmpRecAddr,BOff,BOff);

If ((GetHist.Pr<>YTD) and (GetHist.Pr<>YTDNCF)) then

Begin

{ Only process non carry forward codes, as YTD will be updated

automatically. Purge year not destroyed}

Repeat

{Attempt to find immediate parent equivalent, if not create via fill}

With HiHist do

KeyHChk:=FullNHistKey(ExClass,Code,GetHist.Cr,GetHist.Yr,GetHist.Pr);

KeyHS:=KeyHChk;

LStatus:=LFind\_Rec(B\_GetEq,Fnum,Keypath,KeyHS);

If (Not LStatusOk) or

(Not CheckKey(KeyHChk,KeyHS,Length(KeyHChk),BOn)) then

Begin

With UserProfile^ do

Begin

UCYr:=GetHist.Yr;

UCPr:=GetHist.Pr;

end;

LFillBudget(FNum,KeyPath,n,KeyHChk);

end

else

FoundOk:=BOn;

fLoop:=Not fLoop;

Until (Not fLoop) or (FoundOk);

end

else

Begin

FoundOk:=AutoPBF;

LNHist:=HiHist;

LNHist.Yr:=GetHist.Yr;

LNHist.Pr:=GetHist.Pr;

LNHist.Cr:=GetHist.Cr;

end;

If (FoundOk) then

Begin

HedHist:=LNHist;

(MH I believe the following function is one of Catherine SP functions )

LPost\_To\_Hist2(HedHist.ExClass,HedHist.Code,

GetHist.Purchases,

GetHist.Sales,

GetHist.Cleared,

GetHist.Value1,

GetHist.Value2,

HedHist.Cr,HedHist.Yr,HedHist.Pr,

PBalBF);

If (GetHist.ExClass In YTDSet+YTDNCFSet) and

(Not AutoPBF) then {Process carry forward codes YTD, and future YTD}

Begin

If (GetHist.ExClass In YTDSet) then

YTDType:=YTD

else

YTDType:=YTDNCF;

(MH I believe the following function is one of Catherine SP functions )

LPost\_To\_CYTDHist2(HedHist.ExClass,HedHist.Code,

GetHist.Purchases,

GetHist.Sales,

GetHist.Cleared,

GetHist.Value1,

GetHist.Value2,

HedHist.Cr,HedHist.Yr,YTDType);

end;

If (PostView.NomVieWLine.IncBudget) then

Begin

With HedHist do

KeyHChk:=FullNHistKey(ExClass,Code,Cr,Yr,Pr);

KeyHS:=KeyHChk;

LStatus:=LFind\_Rec(B\_GetEq,Fnum,Keypath,KeyHS);

If (StatusOk) then

With LNHist do

Begin

Budget:=Budget+GetHist.Budget;

Budget2:=Budget2+GetHist.Budget2;

LStatus:=LPut\_Rec(Fnum,Keypath);

LReport\_BError(Fnum,LStatus);

end;

end;

end; {If FoundOk..}

TmpStat:=LPresrv\_BTPos(Fnum,TmpKpath,LocalF^[Fnum],TmpRecAddr,BOn,BOff);

end; {If wrong type}

LStatus:=LFind\_Rec(B\_GetNext,Fnum,keypath,KeyS);

end; {While..}

LReport\_BError(Fnum,LastStatus);

end; {With..}

UserProfile^:=LUP;

With PostView.NomViewLine do

Begin

If (ViewCat<>0) then

With MTExLocal^ do

Begin

KeyS:=FullNVIdx(NVRCode,NVVSCode,NomViewNo,ViewCat,BOn);

LStatus:=LFind\_Rec(B\_GetEq,NomViewF,NVViewIDxK,KeyS);

If (LStatusOk) then

Begin

KeyS:=Set\_NVHist(LNomView^,BOff,AutoPBF);

Update\_NomViewActual(LowKey,KeyS,LNomView^,ProMode,AutoPBF);

end;

end;

end;

end; {Proc..}

# Stock\_AddCustAnal

Procedure Stock\_AddCustAnal in CustKA3U.pas is used to add, update or delete a Customer/Stock Analysis record when adding or deleting a transaction line.

**Overview**

We look for a Customer/Stock Analysis record in MLocStk which matches the Customer Code and Stock Code passed in.

Depending on the Mode passed in and whether we find a record there are 4 possiblities:

A. Mode = 1 and we find a record

B. Mode = 1 and we don't find a record

C. Mode <> 1 and we find a record

D. Mode <> 1 and we don't find a record

**Actions:**

A.

Check if any other transaction lines refer to this record

if yes then

do nothing

else

delete the record

B.

Do nothing

C.

Is IdDocHed in [0,1,3,4,6,8,9]?

If yes then

Update the record with set values

Else

do nothing

D.

Add a record

Set values

Is IdDocHed in [0,1,3,4,6,8,9]?

If yes then set more values

==============================================================

Procedure Stock\_AddCustAnal

{Parameters passed in – these parameters (apart from Mode) come from the Detail record passed into the existing procedure.}

CustCode, StockCode, PDate : string

FolioRef, AbsLineNo : Integer

Currency : Integer

IdDocHed : Integer

LineType : Char

LineTotal : Double

Mode : Integer //(0 or 1 only)

BEGIN

//Try to find an existing record for this Customer/Stock combination eg

Select @Count = Count(\*) from [Company].MLOCSTK

Where Prefix = 'T' AND SubType = 'P' AND csCustCode = CustCode AND csStockCode = StockCode

If Mode = 1 then

begin

If @Count > 0 then

begin

//A

//Check whether we have any other transaction lines which refer to this record. eg

Select @Count = Count(tlFolioNum) from [Company].Details where

tlLineType = LineType AND tlAcCode = CustCode AND tlStockCode = StockCode

AND (tlFolioNum <> FolioRef OR tlAbsLineNo <> AbsLineNo)

if @Count = 0 then

Delete the record

end

else

//B Do Nothing

end

else

begin

Select @StockFolio = stFolio,

@CalcPack = stCalcPack

From [Company].Stock

where stCode = StockCode

If @Count > 0 then

Begin

//C

if IdDocHed in [0,1,3,4,6,8,9] then

begin

Update MLocStk record with the following values

csLastDate = PDate

csLPCurr = Currency

csLastPrice = LineTotal

end

end

else

begin

//D

//Find the next line number for this customer

Select @LineNo = MAX(csLineNo) from [Company].MLocStk

where Prefix = 'T' AND SubType = 'P' AND csCustCode = CustCode

If @LineNo IS NULL

@LineNo = 1

Else

@LineNo = @LineNo + 1

Add MLocStk record with the following values

RecPFix= 'T'

SubType= 'P'

csLineNo=@LineNo;

csCode1=Full\_CuStkLKey(CustCode,csLineNo);

csCode2=Full\_CuStkKey(CustCode,StockCode);

csCode3=Full\_CuStkKey2(CustCode,StockCode);

csCustCode=CustCode;

csStockCode=StockCode;

csStkFolio= @StockFolio;

if IdDocHed in [0,1,3,4,6,8,9] then

begin

csLastDate = PDate

csLPCurr = Currency

csLastPrice = LineTotal

end

end

end;

END

**Called functions**

function Full\_CuStkKey(cc : Str10;

sc : Str20) : Str30;

Begin

//FullCustCode pads cc to 6 chars with spaces

//ConstStr returns a string of 4 char 0s

//FullStockCode pads sc to 16 chars with spaces

Full\_CuStkKey:=FullCustCode(cc)+ConstStr(#0,4)+FullStockCode(sc);

end;

function Full\_CuStkKey2(cc : Str10;

sc : Str20) : Str30;

begin

Full\_CuStkKey2:=FullStockCode(sc)+ConstStr(#0,4)+FullCustCode(cc);

end;

Function Full\_CuStkLKey(cc : Str10;

LineNo : LongInt) : Str30;

begin

//Dec2Hex returns a 4 character hex string for LineNo

Full\_CuStkLKey:=FullCustCode(cc)+ConstStr(#0,4)+Dec2Hex(LineNo);

end;

==================================================================================

**Calling the stored procedure from the Stock\_AddCustAnal procedure**

I would envisage calling the procedure as follows

Begin

Locked:=BOff;

OldId:=Id;

OStat:=Status;

RunOk:=BOn;

If (Not EmptyKey(IdR.CustCode,CustKeyLen)) and (Is\_FullStkCode(IdR.StockCode)) and (AnalCuStk)

and (Not (IdR.IdDocHed In QuotesSet)) and (IdR.LineNo>0) and (RunOk) then

Begin

If SQLUtils.UsingSQLAlternateFuncs then

call stored procedure

else

continue with standard procedure

Original code

Procedure Stock\_AddCustAnal(IdR : IDetail;

GetSRec : Boolean;

Mode : Byte);

Const

Fnum = MLocF;

Keypath = MLSecK;

Fnum2 = IdetailF;

Keypath2 = IdCAnalK;

Var

KeyS,

KeyChk : Str255;

GenStr : Str20;

n : Byte;

OStat,

OStat2 : Integer;

TmpKPath,

TmpStat

: Integer;

UseNext,

TmpRecAddr,

LAddr

: LongInt;

KeepRec,

NewRec,

RunOk,

Locked : Boolean;

OldId : Idetail;

Begin

Locked:=BOff;

OldId:=Id;

OStat:=Status;

RunOk:=BOn;

If (Not EmptyKey(IdR.CustCode,CustKeyLen)) and (Is\_FullStkCode(IdR.StockCode)) and (AnalCuStk)

and (Not (IdR.IdDocHed In QuotesSet)) and (IdR.LineNo>0) and (RunOk) then

Begin

KeepRec:=BOff;

With IdR do

KeyChk:=PartCCKey(MatchTCode,MatchSCode)+Full\_CuStkKey(CustCode,StockCode);

Status:=Find\_Rec(B\_GetEq,F[Fnum],Fnum,RecPtr[Fnum]^,KeyPath,KeyChk);

NewRec:=(Status=4) and (Mode<>1);

If ((StatusOk) or (NewRec)) then

With IdR do

Begin

If (NewRec) then

With MLocCtrl^,CuStkRec do

Begin

RunOk:=BOn;

Locked:=BOn;

UseNext:=Get\_LastLineNo(CustCode);

ResetRec(Fnum);

RecPFix:=MatchTCode;

SubType:=MatchSCode;

csLineNo:=UseNext;

csCode1:=Full\_CuStkLKey(CustCode,csLineNo);

csCode2:=Full\_CuStkKey(CustCode,StockCode);

csCode3:=Full\_CuStkKey2(CustCode,StockCode);

csCustCode:=CustCode;

csStockCode:=StockCode;

If (GetSRec) and (Stock.StockCode<>StockCode) then

Begin

Global\_GetMainRec(StockF,StockCode);

end;

csStkFolio:=Stock.StockFolio;

end

else

RunOk:=GetMultiRecAddr(B\_GetDirect,B\_MultLock,KeyChk,KeyPath,Fnum,BOn,Locked,LAddr);

If (RunOk) and (Locked) then

With MLocCtrl^,CuStkRec do

Begin

If (Mode=1) then {\* its a deduct check if any other lines have it, else delete it \*}

Begin

TmpKPath:=GetPosKey;

TmpStat:=Presrv\_BTPos(Fnum2,TmpKPath,F[Fnum2],TmpRecAddr,BOff,BOff);

KeyChk:=LineType+Full\_CuStkKey(CustCode,StockCode);

KeyS:=KeyChk;

Status:=Find\_Rec(B\_GetGEq,F[Fnum2],Fnum2,RecPtr[Fnum2]^,KeyPath2,KeyS);

KeepRec:=(StatusOk) and (CheckKey(KeyS,KeyChk,Length(KeyChk),BOff)) and

(FullRunNoKey(IdR.FolioRef,IdR.AbsLineNo)<>FullRunNoKey(Id.FolioRef,Id.AbSLineNo));

TmpStat:=Presrv\_BTPos(Fnum2,TmpKPath,F[Fnum2],TmpRecAddr,BOn,BOff);

Id:=OldId;

end

else

KeepRec:=BOn;

If (KeepRec) then

Begin

If (Mode<>1) and (IdDocHed In SalesSplit-CreditSet) then

Begin

csLastDate:=PDate;

csLPCurr:=Currency;

If (Stock.CalcPack) then

QtyMul:=1;

Qty:=1;

csLastPrice:=DetLTotal(IdR,BOn,BOff,0.0)\*DocNotCnst;

end;

If (NewRec) then

Status:=Add\_Rec(F[Fnum],Fnum,RecPtr[Fnum]^,KeyPath)

else

Begin

Status:=Put\_Rec(F[Fnum],Fnum,RecPtr[Fnum]^,KeyPath);

OStat2:=UnLockMultiSing(F[Fnum],Fnum,LAddr);

end;

end

else

Begin

Status:=Delete\_Rec(F[Fnum],Fnum,KeyPath);

end;

Report\_BError(Fnum,Status);

end;

end; {With..}

end; {If no cust, or stock code}

Id:=OldId;

Status:=OStat;

end; {Proc..}

# GL Views – Reset\_ViewHistory

Reset\_ViewHistory is the initial 50% of the progress bar when Refreshing a GL View.

It basically runs through the Nom View Lines and resets the History records for each one. It can be run in two modes, firstly it can be run against a specific GL View and secondly it will run against all GL Views.

To test you can simply run the stored procedure against the GL View and any numbers against it should be zeroed.

## Code

{ ======= Function Dec2Hex ======= }

Function Dec2Hex(D : LongInt) : Str255;

Var

Err : Byte;

Lnum : LongInt;

TmpStr : Str255;

Begin

Str(D:0,TmpStr);

ConvNumBase(TmpStr,10,Err,Lnum);

TmpStr:='';

If (Err=0) then

PutNum(Lnum,16,TmpStr)

else

TmpStr:='';

While (Length(TmpStr)<4) do

TmpStr:='0'+TmpStr;

Dec2Hex:=TmpStr;

end;

Function PostNVIdx(VN,VIdx : LongInt) : Str255;

Begin

Result:=#1+#1+FullNomKey(VN)+Dec2Hex(VIdx);

end;

{ ========== Zero NomView Balance ========== }

Procedure TCheckNom.Reset\_NomViewActual;

Const

Fnum = NHistF;

Keypath = NHK;

Var

KeyS, KeyChk : Str255;

BalCFS, BalCFP, BalCFC,

Begin

With MTExLocal^, LNomView^.NomViewLine do

Begin

KeyChk:=ViewType+PostNVIdx(NomViewNo,ABSViewIdx);

KeyS:=KeyChk;

LStatus:=LFind\_Rec(B\_GetGEq,Fnum,KeyPath,KeyS); // History Table

While (LStatusOk) and (CheckKey(KeyChk,KeyS,Pred(Length(KeyChk)),BOn)) do

With LNHist do

Begin

Sales:=0.0;

Purchases:=0.0;

Cleared:=0.0;

Value1:=0.0;

Value2:=0.0;

Value3:=0.0;

If (IncBudget) then

Begin

Budget:=0.0;

Budget2:=0.0;

end;

LStatus:=LPut\_Rec(Fnum,Keypath); // History Table

LReport\_BError(Fnum,LStatus);

LStatus:=LFind\_Rec(B\_GetNext,Fnum,KeyPath,KeyS); // History Table

end; {With..}

end; {With..}

end; {Proc..}

MH: Basically loops through the NomView Table where RecPFix = NVRCode (‘N’) and SubType = NVVSCode (‘V’) calling Reset\_NomViewActual (above).

{ ========== Procedure to reset all Headings prior to an update ======= }

Procedure TCheckNom.Reset\_ViewHistory;

Const

Fnum = NomViewF;

Keypath = NVCodeK;

Var

KeyChk, KeyS : Str255;

LOk, Locked : Boolean;

ItemCount : LongInt;

Begin

KeyChk:=PartCCKey(NVRCode,NVVSCode); // ‘N’ + ‘V’

If (tViewNo<>0) then {\* One view only \*}

KeyChk:=KeyChk+FullNomKey(tViewNo); // tViewNo = ViewCtrlNo from GL View Ctrl

KeyS:=KeyChk;

ItemCount:=0;

Locked:=BOff;

With MTEXLocal^ do

Begin

LStatus:=LFind\_Rec(B\_GetGEq,Fnum,keypath,KeyS); // NomView table

While (LStatusOk) and (Not ThreadRec^.THAbort) and (CheckKey(KeyChk,KeyS,Length(KeyChk),BOn)) do

With LNomView^.NomViewLine do

Begin

Inc(ItemCount);

UpdateProgress(ItemCount);

Reset\_NomViewActual;

LStatus:=LFind\_Rec(B\_GetNext,Fnum,Keypath,KeyS); // NomView table

end; {While..}

end; {With.}

end; {Proc..}

# Total Profit To Date

Recalculates the financial figures for the specified account, and returns the balance. The figures are returned through the Purch, PSales, PCleared, PBudget, PRBudget, Bvalue1, and BValue2 parameters.

## Code

function Total\_Profit\_To\_DateRange(NType : Char;

NCode : Str20;

PCr : Byte;

PYr : Byte;

PPr : Byte;

PPr2 : Byte;

Var Purch : Double;

PSales : Double;

PCleared : Double;

PBudget : Double;

PRBudget : Double;

Var BValue1 : Double;

BValue2 : Double;

Range : Boolean;

Set\_ACHist: Boolean): Double;

var

NHKey : Str255;

NHChk : Str255;

NHKey2 : Str255;

Bal : Double;

begin

Purch := 0.0;

PSales := 0.0;

PCleared := 0.0;

PBudget := 0.0;

Bal := 0.0;

PRBudget := 0.0;

BValue1 := 0.0;

BValue2 := 0.0;

NHChK := FullNHistKey(NType, NCode, PCr, PYr, PPr);

if (Range) then

// Set the range from the Year-To-Date period of the previous year (if any)

NHKey := FullNHistKey(NType, NCode, PCr, AdjYr(PYr, BOff), YTD)

else

// No range -- just use the specified year and period

NHKey := NHChk;

// If we are processing a range, find the last valid Year-To-Date record

// which is previous to the current year

if (NType in ['B', 'C', #239, #236, #247, 'K', 'Z', 'J', '9', '8', 'H', 'U', 'V', '[']) and

(Range) then

begin

NHKey2 := NHKey;

Status := Find\_Rec(B\_GetLessEq, F[Fnum], Fnum, RecPtr[Fnum]^, NPath, NHKey2);

if (StatusOk) and

(CheckKey(NHChk, NHKey2, Length(NHChk) - 2, BOn)) and

(NHist.Pr = YTD) then

begin

NHKey := NHKey2;

end;

end;

// If a period range has been specified, calculate the key for the last valid

// record in the range

if (PPr2 > PPr) and (PPr2 <> 0) and (not Range) then

begin

NHChK := FullNHistKey(NType, NCode, PCr, PYr, PPr2);

end;

// Find the first record

Status := Find\_Rec(B\_GetGEq, F[Fnum], Fnum, RecPtr[Fnum]^, NPath, NHKey);

// Process the records until we reach the last valid record (if we are not

// doing a range, the start and end keys will be the same, and only one

// record (at most) will be processed

while (StatusOK) and (NHKey <= NHChk) do

begin

if ((NType <> 'U') or (not (History.Pr in [255, 254]))) or

(Set\_ACHist) then

begin

Purch := Purch + History.Purchases;

PSales := PSales + History.Sales;

end;

Bal := Bal + (History.Purchases - History.Sales);

PCleared := PCleared + History.Cleared;

if (not (History.Pr in [255, 254])) then

begin

PBudget := PBudget + History.Budget;

PRBudget := PRBudget + History.Budget2;

end;

BValue1 := BValue1 + History.Value1;

BValue2 := BValue2 + History.Value2;

Status := Find\_Rec(B\_GetNext, F[Fnum], Fnum, RecPtr[Fnum]^, NPath, NHKey);

end;

Result := Bal;

end;

// Returns the History search key for the supplied details

function FullNHistKey(Typ: Str5; Code: Str20; Cr, Yr, Pr: Byte): Str30;

begin

Result := Typ + LJVar(Code, NHCodeLen) + Chr(Cr) + Chr(Yr) + Chr(Pr);

end;

// Right-pads the supplied string to the specified length

function LJVar(Str2Pad : AnyStr; PadLen : Integer) : AnyStr;

begin

if (Length(Str2Pad) < PadLen) then

LJVar := Str2Pad + ConstStr(' ', (PadLen - Length(Str2Pad)))

else

LJVar := Copy(Str2Pad, 1, PadLen);

end;

// Adjusts the supplied year by one and returns the result

function AdjYr(AYr: Integer; Advance: Boolean): Integer;

begin

if (Advance) then

Inc(AYr)

else

Dec(AYr);

if (Ayr < 0) then

Ayr := 99;

Result := AYr;

end;

# Remove Previous Commitments

This routine is part of the Daybook Posting, and deletes the commitment records from the History table and the Details table.

## Code

procedure Remove\_LastCommit;

const

// GLNHCodes: array[0..4] of Char = (StkBillCode, NomHedCode, PLNHCode, BankNHCode, CtrlNHCode);

GLNHCodes: array[0..4] of Char = ('M', 'H', 'A', 'B', 'C');

CommitOrdRunNo = -53; // Run time lines generated by commitment module

var

n : byte;

KeyS : Str255;

DelKey : Str255;

begin

KeyS := CommitKey;

// Delete all the records for each General Ledger History code

for n := Low(GLNHCodes) to High(GLNHCodes) do

begin

DelKey := GLNHCodes[n] + KeyS;

KLen := Length(DelKey);

Status := Find\_RecCId(B\_GetGEq, LocalF^[NHistF], NHistF, LRecPtr[NHistF]^, NHK, DelKey);

while (Status = 0) and (CheckKey(Code, DelKey, KLen, BOn)) do

begin

Delete\_Rec(NHistF, KeyPth);

Status := Find\_Rec(B\_GetNext, NHistF, NHK, KeyS);

end;

end;

// Delete all the transaction lines with commitment run numbers

KeyS := FullNomKey(CommitOrdRunNo);

KLen := Length(KeyS);

Status := Find\_Rec(B\_GetGEq, LocalF^[IdetailF], IdetailF, LRecPtr[IdetailF]^, IdRunK, KeyS);

while (Status = 0) and (CheckKey(Code, KeyS, KLen, BOn)) do

begin

Delete\_Rec(IdetailF, IdRunK);

Status := Find\_Rec(B\_GetNext, IdetailF, IdRunK, KeyS);

end;

end;

function CommitKey: Str255;

begin

Result := 'CMT' + ConstStr(#2, 2) + '!';

end;

# Partial Unpost History

This function is central to the Partial Unpost routines. It updates the History records for the specified class and key, for all currency types, also recreating the Year-To-Date records where required.

## Code

procedure TPartialUnpost.Part\_UPHist(BackPr,BackYr : Byte;

Fnum,Keypath : Integer;

UPNomKey : Str255;

UPNomT : Char;

var PALBFRec : PALBFType);

var

CurrLoop : Byte;

KeyChk,

KeyS : Str255;

UStatus,

B\_Func : Integer;

HighYear,

HYrLoop : LongInt;

LOk,

Locked,

HadYTD : Boolean;

PSales,

PPurch,

PCleared,

PBudget,

PBudget2,

BVal1,

BVal2,

PBalance : Double;

CorrYTD : Byte;

KeepYTD : HistoryRec;

begin

CurrLoop := 0;

CorrYTD := YTD;

HighYear := BackYr;

HYrLoop := 0;

with MTExLocal^ do

// Repeat for every currency type (0 to 90)

repeat

Blank(KeepYTD, Sizeof(KeepYTD));

B\_Func := 0;

HadYTD := BOff;

// Set up search keys to find all the History records for the Type and

// Code, from the 'Unpost To' year and period onwards.

KeyChk := PartNHistKey(UpNomT, UpNomKey, CurrLoop);

KeyS := FullNHistKey(UpNomT, UpNomKey, CurrLoop, BackYr, Succ(BackPr));

LStatus := LFind\_Rec(B\_GetGEq, Fnum, KeyPath, KeyS);

while (LStatusOk) and (CheckKey(KeyChk, KeyS, Length(KeyChk), BOn)) do

with LNHist do

begin

// Only process the record if it is for a year and period after the

// year and period to unpost to.

if (Pr2Fig(Yr, Pr) > Pr2Fig(BackYr, BackPr)) then

begin

// Year-To-Date records will need to be recalculated, so delete any

// existing ones

if (Pr In [YTD, YTDNCF]) then

begin

// Take a copy of the current History details, and delete the

// record from the table

KeepYTD := LNHist;

LStatus := LDelete\_Rec(Fnum, KeyPath);

B\_Func := B\_GetGEq;

HadYTD := BOn;

// Keep a record of the most recent year for which we found a

// Year-To-Date record

if (Yr > HighYear) and (Pr = YTD) then

HighYear := Yr;

end

else

begin

// Reset the figures for the current record

Sales := 0;

Purchases := 0;

Cleared := 0;

LStatus := LPut\_Rec(Fnum, Keypath);

B\_Func := B\_GetNext;

end;

LReport\_BError(Fnum,LStatus);

end

else

B\_Func := B\_GetNext;

LStatus := LFind\_Rec(B\_Func, Fnum, KeyPath, KeyS);

end;

// If the current record was a Year-To-Date record, we need to recreate

// it with updated values

if (HadYTD) then

begin

PBalance := Total\_Profit\_To\_Date(UpNomT, UpNomKey, CurrLoop, BackYr, BackPr, PPurch, PSales, PCleared, PBudget, PBudget2, BVal1, BVal2, BOn);

// Use the details stored from the deleted Year-To-Date record

LNHist := KeepYTD;

with LNHist do

begin

Yr := BackYr;

Purchases := PPurch;

Sales := PSales;

Cleared := PCleared;

Budget := PBudget;

Budget2 := PBudget2;

// Determine whether this should be YTD or YTDNCF

if (ExClass in ProfitBFSet + [CtrlNHCode, BankNHCode, NomHedCode]) then

begin

CorrYTD := Calc\_PropYTD(UpNomKey, UpNomT, Pr);

if (Pr <> CorrYTD) then

Pr := CorrYTD;

end;

LStatus := LAdd\_Rec(Fnum, Keypath);

LReport\_BError(Fnum, LStatus);

// Fill in the intervening years for carry forwards

if (HighYear > BackYr) then

for HYrLoop := Succ(BackYr) to HighYear do

begin

Yr := HYrLoop;

LStatus := LAdd\_Rec(Fnum, Keypath);

LReport\_BError(Fnum, LStatus);

end;

// Profit & Loss figures need to be returned from this function

if (ExClass = PLNHCode) then

with PALBFRec[CurrLoop] do

begin

PALPurch := PALPurch + PPurch;

PALSales := PALSales + PSales;

PALBudget := PALBudget + PBudget;

PALCleared := PALCleared + PCLeared;

end;

end;

end;

Inc(CurrLoop);

until (CurrLoop > CurrencyType);

end;

// -----------------------------------------------------------------------------

function PartNHistKey(Typ: Str5; Code: Str20; Cr: Byte): Str30;

begin

Result := Typ + LJVar(Code, NHCodeLen) + Chr(Cr);

end;

// -----------------------------------------------------------------------------

function FullNHistKey(Typ: Str5; Code: Str20; Cr, Yr, Pr: Byte): Str30;

begin

Result := PartNHistKey(Typ, Code, Cr) + Chr(Yr) + Chr(Pr);

end;

// -----------------------------------------------------------------------------

function LJVar(Str2Pad: AnyStr; PadLen: Integer): AnyStr;

begin

if (Length(Str2Pad) < PadLen) then

Result := Str2Pad + ConstStr(' ', (PadLen - Length(Str2Pad)))

else

Result := Copy(Str2Pad, 1, PadLen);

end;

// -----------------------------------------------------------------------------

// ConstStr returns a Str255 with N characters of value C

function ConstStr(C: Char; N: Integer): AnyStr;

var

S: AnyStr;

begin

if N < 0 then

N := 0;

S[0] := Chr(N);

FillChar(S[1], N, C);

ConstStr := S;

end;

// -----------------------------------------------------------------------------

function Pr2Fig(FYr, FPr: Byte): LongInt;

begin

Result := (FYr \* 100) + FPr;

end;

// -----------------------------------------------------------------------------

function TPartialUnpost.Calc\_PropYTD(UpNomKey : Str255;

UpNomCat : Char;

CCode : Byte) : Byte;

var

TmpRes : Byte;

begin

TmpRes := CCode;

Result := CCode;

If (UPNomCat In ProfitBFSet) then

Result := YTDNCF

else if (UPNomCat In YTDSet) then

Result := YTD

else if (UpNomCat=NomHedCode) then

begin

if (Get\_NomHedType(UpNomKey, TmpRes)) and (TmpRes In [YTD,YTDNCF]) then

Result := TmpRes;

end;

end;

// -----------------------------------------------------------------------------

function TPartialUnpost.Get\_NomHedType(KeyNom: Str255; var YTDRes: Byte): Boolean;

{ This routine attempts to determine whether the nominal code in KeyNom is for

YTD figures or for YTDNFC figures, by searching down through the G/L tree }

const

Fnum = NomF;

Keypath = NomCatK;

{

YTDSet: CharSet = [BankNHCode, CtrlNHCode, StkStkQCode, StkBillQCode, StkDLQCode,

JobGrpCode, JobPhzCode, JobJobCode, ViewHedCode, ViewBalCode];

ProfitBFSet: CharSet = [PLNHCode]; // Nom Types affecting Profit b/f

}

YTDSet: CharSet = ['B', 'C', #239, #236, #247, 'K', 'Z', 'J', '9', '8'];

ProfitBFSet: CharSet = ['A'];

NomHedCode = 'H';

var

FoundOk : Boolean;

GResult : Byte;

KeyChk,

KeyS : Str255;

begin

GResult := 0;

KeyChk := KeyNom;

TmpKPath := GetPosKey;

FoundOk := BOff;

KeyS := KeyChk;

LStatus := LFind\_Rec(B\_GetGEq, Fnum, KeyPath, KeyS);

{ Try to locate a Nominal record where the parent code matches the

current Nominal code (passed through KeyNom) }

while (LStatus = 0) and

(CheckKey(KeyChk, KeyS, Length(KeyChk), BOn)) and

(not FoundOk) do

begin

FoundOk := (MTExLocal^.LNom.NomType in YTDSet + ProfitBFSet);

if (FoundOk) then

begin

if (MTExLocal^.LNom.NomType In YTDSet) then

GResult := YTD

else

GResult := YTDNCF;

end

else if (MTExLocal^.LNom.NomType = NomHedCode) then

{ The Nominal record is not for a code included in YTD or YTDNCF

figures, take the Nominal Code, and recursively search for a record

against this as the parent code. }

FoundOk := Get\_NomHedType(FullNomKey(MTExLocal^.LNom.NomCode), GResult);

if (not FoundOk) then

LStatus := LFind\_Rec(B\_GetNext, Fnum, KeyPath, KeyS);

end;

if (FoundOk) then

YTDRes := GResult;

Result := FoundOk;

end;