# Order Processing – Deliver This Order

## (SOR/POR to SDN/PDN, and SDN/PDN to SIN/PIN)

**[*DelvRunU: TSOPRunFrm.SOP\_GenDel()*] Deliver This Order** – creates a Delivery Note from an Order, moving the Order into Order History if it is fully delivered or written-off.

1. Prepare custom hook points.
2. If the current transaction is not one of PIN,PPY,PCR,PJI,PJC,PRF,PPI,PQU,POR,PDN,PBT, ask the user for confirmation that the transaction should be delivered.
3. Retrieve the Trader record matching SOR.DOCUMENT.thAcCode.
4. Check Trader balance against their credit limit (if used).
   1. [*Warn1U: Check\_AccForCredit()*] If this Trader has a head-office record (acInvoiceTo), locate the Trader whose acCustCode matches this value, and do the credit check against this record instead.
   2. [*SysU1: Get\_CurCustBal()*] Get the balance for this Trader, for this year up to period 99, and against a History code of ‘U’, and including the committed balance (i.e. add the Cleared column to the returned balance), using the Profit to Date Range stored procedure.
   3. [*Warn1U: Warn\_ODCredit()*] Get the balance for this Trader, for this year up period 99, and against a History code of ‘U’, but excluding the committed balance, using the Profit to Date stored procedure.
   4. If the Use Credit Limit Check system flag is set, determine whether or not to continue with the transaction.
      1. If the Balance calculated by 1.4.2 exceeds the trader’s credit limit (CUSTSUPP.acCreditLimit) (note that for Customers the sign of the Balance needs to be inverted before making this check), and the credit limit is not zero, set the error message to 'Bad Debtor! - Credit Status Exceeded'.
      2. [ TBD ]
5. [*DelvRunU: SOP\_Check4Pick()*] Check that there are Picked Lines against the Transaction.
   1. For each line against the transaction:
      1. Add QtyPWOff to QtyPick. If the result is not zero, at least one line is picked and the transaction can be processed.
   2. If no picked lines were found, display a message advising the user of this, and exit.
6. [*DelvRunU: SOP\_ProcessDel()*] For each SOR Transaction Line, add a new line to the Delivery Transaction (creating a Transaction Header when we find the first valid line to be added), using the following rules:
   1. Check Order Payments to see if this Line requires a refund. Do not process the line if it does (see 1.21 below).
      1. [ TBD ]
   2. [*SOPCT3U: Get\_InvHed()*] If we don’t have a Transaction Header yet, create one now, copying most of the details from the Order Transaction.
      1. Locate the Trader record (in theory we should already have this).
      2. Store the Delivery Address, Post Code, and Country Code.
      3. If there is a code for a Head Office record (in CUSTSUPP.acInvoiceTo), locate this record and use it instead.
      4. Copy SOR.DOCUMENT to SDN.DOCUMENT.
      5. Change SDN.DOCUMENT.thDocType to SDN/PDN.
      6. Clear SDN.DOCUMENT.thHoldFlag, unless the Notes flag is included, in which case set it to Notes (only).
      7. Set SDN.DOCUMENT.thNextLineNumber to 1.
      8. Set SDN.DOCUMENT.thNextNotesLineNumber to 1.
      9. Set SDN.DOCUMENT.thBatchLink to the binary (FullNomKey) of the SOP Run Number (this value is passed in to the Deliver This Order routine) plus the string of the document type (“SDN”/“PDN”).
      10. Copy SOR.DOCUMENT.thOurRef to SDN.DOCUMENT
          1. If EXCHQSS.ProtectYRef is true, then if SDN.DOCUMENT.thYourRef is empty, copy SOR.DOCUMENT.thOurRef to SDN.DOCUMENT.thYourRef.
          2. If EXCHQSS.ProtectYRef is true, but SDN.DOCUMENT.thYourRef already holds a value, then check SDN.DOCUMENT.thLongYourRef instead. If this is empty, copy SOR.DOCUMENT.thOurRef to SDN.DOCUMENT.thLongYourRef.
          3. If EXCHQSS.ProtectYRef is true, but both SDN.DOCUMENT.thYourRef and SDN.DOCUMENT.thLongYourRef already hold values, do not copy SOR.DOCUMENT.thOurRef at all.
          4. If EXCHQSS.ProtectYRef is false, copy SOR.DOCUMENT.thOurRef to SDN.DOCUMENT.thYourRef, and copy SOR.DOCUMENT.thYourRef to SDN.DOCUMENT.thLongYourRef.
      11. Copy SOR.DOCUMENT.thOurRef to SDN.DOCUMENT.thRemitNo.
      12. Set SDN.DOCUMENT.thOrdMatch to 0 (False).
      13. For Sales Orders, set SDN.DOCUMENT.thRunNo to -43, for Purchase Orders set SDN.DOCUMENT.thRunNo to -45.
          1. ***Note****: I believe that these values are deliberately invalid, and are set to ensure that the newly-created document header is not visible to other users (e.g. in the daybook or in reports) until the Run Number is set to valid values later in this routine.*
      14. Set SDN.DOCUMENT.thTransDate to today’s date.
      15. Calculate and store the value for SDN.DOCUMENT.thDueDate.
          1. Add the number of days stored in CUSTSUPP.acPayTerms to SDN.DOCUMENT.thTransDate to get the Due Date.
          2. However, if the number of days is 500 or greater, the 900-Codes are being used, and the Due Date should instead be calculated from SDN.DOCUMENT.thTransDate using the rules for these codes. See Appendix 1 for an explanation.
      16. Set SDN.DOCUMENT.thPrinted to 0 (False).
      17. If the Keep Tag Number hook-point was not active, set SDN.DOCUMENT.thTagged to 0.
      18. Copy Cust.acCustCode to SDN.DOCUMENT.thCustCode (note that because of 1.6.2.3 above, this might be the head-office code).
      19. Set SDN.DOCUMENT.thOperator to the current user’s log-in name.
      20. Set the Originator fields:
          1. Set SDN.DOCUMENT.thCreationDate to today’s date.
          2. Set SDN.DOCUMENT.thCreationTime to now, as ‘hhmmss’
          3. Set SDN.DOCUMENT.thOriginator to the current user’s log-in name.
      21. If SDN.DOCUMENT.thDeliveryAddr1 and SDN.DOCUMENT.thDeliveryAddr2 are both empty:
          1. Copy the Delivery Address lines from the actual Trader record (*not* the Head Office record) to the Delivery Address lines in SDN.DOCUMENT.
          2. Similarly copy the Post Code and Country Code from the actual Trader record to SDN.DOCUMENT.thPostCode and SDN.DOCUMENT.thCountryCode.
      22. [*SysU1: SetNextDocNos()*] Set SDN.DOCUMENT.thOurRef and update the document numbers table.
          1. [*SysU1: FullDocNum()*] Calculate the complete Our Ref, based on the document type and the current SDN/PDN entry in the Document Numbers table (EXCHQNUM)
             1. Get the next number:

Look up the record for the document type (the table is indexed on the document type string, in this case ‘SDN’ or ‘PDN’).

Use the existing value of EXCHQNUM.ssNextCount as the basis for the Our Ref, and increment the number in the table.

* + - * 1. If the number is greater than 999999, replace the first digit (most significant digit) with an alpha character, from A (=1000000) to Z (=3200000), omitting I, O, and S.
        2. If the number has less than 6 digits, left-pad it with zeroes to 6 digits.
        3. Prefix the number with the document type to form the final Our Ref, and assign this to SDN.DOCUMENT.thOurRef.
      1. Calculate the next folio number.
         1. Look up the ‘AFL’ record in EXCHQNUM.
         2. Store the existing value of EXCHQNUM.ssNextCount as SDN.DOCUMENT.thFolioNum, and increment the number in the EXCHQNUM table.
    1. Set SDN.DOCUMENT.thCustSupp to ‘?’
       1. ***Note****: I believe that this value is deliberately invalid, and is set to ensure that the newly-created document header is not visible to other users until the Cust Supp code is set to a valid value later in this routine.*
    2. [*SysU2: Re\_SetDocTots()*] Reset the document totals.
       1. If SDN.DOCUMENT.thManualVAT is False, set all the VAT Analysis columns to zero.
       2. Set the total fields to zero:
          1. SDN.DOCUMENT.thNetValue
          2. SDN.DOCUMENT.thTotalLineDiscount
          3. SDN.DOCUMENT.thSettleDiscAmount
          4. SDN.DOCUMENT.thPPDGoodsValue
          5. SDN.DOCUMENT.thPPDVATValue
          6. SDN.DOCUMENT.thSettleDiscTaken
          7. SDN.DOCUMENT.thSettleDiscPerc
          8. SDN.DOCUMENT.PostDiscAm
          9. SDN.DOCUMENT.thPostDiscTaken
          10. SDN.DOCUMENT.thAmountSettled
          11. SDN.DOCUMENT.thVariance
          12. SDN.DOCUMENT.thTotalReserved
          13. SDN.DOCUMENT.thTotalInvoiced
          14. SDN.DOCUMENT.thTotalOrdered
          15. SDN.DOCUMENT.thTotalCost
          16. SDN.DOCUMENT.thCurrSettled
          17. SDN.DOCUMENT.thReValueAdj
    3. Set SDN.DOCUMENT.thPickingRunNo to the SOP Run Number (this run number is generated before the order processing begins and is passed in to the routine).
    4. Clear SDN.DOCUMENT.thVATPostDate to a string of spaces.
    5. Clear SDN.DOCUMENT.thPostedDate to a string of spaces.
    6. Set SDN.DOCUMENT.thOldCompanyRate and SDN.DOCUMENT.thOldDailyRate to zero.
    7. If SDN.DOCUMENT.thFixedRate is 0 (false), set SDN.DOCUMENT.thDailyRate to the system daily rate for the currency.
    8. [*CurrncyU: SetTriRec()*] If SDN.DOCUMENT.thUseOriginalRates is zero, read the triangulation rates from the system currency records:
       1. Set SDN.DOCUMENT.thCurrencyTriRate to CURRENCY.TriRate for the document currency.
       2. Set SDN.DOCUMENT.thCurrencyTriEuro to CURRENCY.TriCurrency for the document currency.
       3. Set SDN.DOCUMENT.thCurrencyTriInvert to CURRENCY.TriInvert for the document currency.
       4. Set SDN.DOCUMENT.thCurrencyTriFloat to CURRENCY.IsFloating for the document currency.
       5. ***Note****: some rows in the CURRENCY table have NULL values in these columns – this would appear to be an error in the trigger which populates this table.*
    9. Set SDN.DOCUMENT.thCompanyRate to zero.
    10. Copy SDN.DOCUMENT.thVATCompanyRate and SDN.DOCUMENT.thVATDailyRate from the System VAT Currency rates (EXCHQSS.VATCurr holds the System VAT Currency).
    11. Copy SDN.DOCUMENT.thOriginalCompanyRate and SDN.DOCUMENT.thOriginalDailyRate from the System Currency rates for the transaction currency.
    12. Set SDN.DOCUMENT.thUseOriginalRates to zero.
    13. Read the triangulation rates for the VAT Currency from the system currency records:
        1. Set SDN.DOCUMENT.thVATTriRate to CURRENCY.TriRate.
        2. Set SDN.DOCUMENT.thVATTriEuro to CURRENCY.TriCurrency.
        3. Set SDN.DOCUMENT.thVATTriInvert to CURRENCY.TriInvert.
        4. Set SDN.DOCUMENT.thVATTriFloat to CURRENCY.IsFloating.
    14. Store the new Transaction Header.
    15. [*AuditNotes: TAuditNote.WriteAuditNote()*] Add an Audit Note.
        1. Add a new record to the TransactionNote table
           1. Set TRANSACTIONNOTE.NoteFolio to SDN.DOCUMENT.thFolioNum
           2. Set TRANSACTIONNOTE.NoteDate to the current date, formatted as YYYYMMDD.
           3. Set TRANSACTIONNOTE.NoteType to 2.
           4. Searching the TransactionNote table in NoteFolio and NoteLine order, find the last record against the current Transaction, to get the last-used LineNumber value. Copy this value + 1 to the TRANSACTIONNOTE.LineNumber on the newly added TransactionNote record.
           5. Set TRANSACTIONNOTE.NoteUser to the current user’s login name.
           6. Set TRANSACTIONNOTE.NoteLine to ‘CREATED BY ‘ + loginname + ‘:’ + current date and time formatted as ‘dd/mm/yyyy hh:nn:ss’

E.g.: ‘CREATED BY MANAGER: 02/11/2015 09:33:19’

* 1. [*InvCtSuU: Stock\_Deduct()*] Put the item back into Stock, as the line on the original Order should no longer affect Stock unless there are still items outstanding (this is dealt with by 1.6.13 below). When the new Transaction Line is added to the Delivery, the Stock will be updated to match the new Stock position.
     1. Locate the Stock record, using SOR.DETAILS.tlStockCode.
     2. Close any open Stock dialog.
     3. If STOCK.StockType is ‘P’, ‘M’, or ‘X’, update the Stock record with the new Allocated and Picked values.
        1. STOCK.stQtyAllocated = STOCK.stQtyAllocated – (SOR.DETAILS.tlQty \* TransactionTypeSignMultiplier). Real48
        2. STOCK.stQtyPicked = STOCK.stQtyPicked – ((SOR.DETAILS.tlQtyPicked \* SOR.DETAILS.tlQtyMul) \* TransactionTypeSignMultiplier). Real48
        3. **Store for sending to Stored Procedure:**
           1. **STOCK.stQtyAllocated**
           2. **STOCK.stQtyPicked**
     4. Update the FIFO valuation (these calculations make extensive use of Real48 values, although the fields in the database are Doubles), provided that SOR.DETAILS.tlYear is greater than EXCHQSS.AuditYr or EXCHQSS.AuditYr is zero.
        1. ***Note****: As far as I can work out, this procedure actually results in no actions being taken, as the values and settings at this point result in every active code path being by-passed.*
     5. If Stock Locations are in use, update the Stock Location record with the new Allocated and Picked values.
        1. STOCKLOCATION.lsQtyAlloc = STOCKLOCATION.lsQtyAlloc – SOR.DETAILS.tlQty. Real48
        2. STOCKLOCATION.lsQtyPicked = STOCKLOCATION.lsQtyPicked – SOR.DETAILS.tlQtyPicked \* SOR.DETAILS.tlQtyMul. Real48
        3. Save STOCKLOCATION.
        4. **Store for sending to Stored Procedure:**
           1. **STOCKLOCATION.stQtyAlloc**
           2. **STOCKLOCATION.stQtyPicked**
     6. Set SOR.DETAILS.tlStockDeductQty to zero.
     7. **Store for sending to Stored Procedure:**
        1. **DETAILS.tlStockDeductQty**
     8. Save STOCK.
  2. **OSValue**: Calculate the outstanding value on the line, taking any discounts into account, but excluding VAT and excluding any items that have already been delivered (tlQtyDel) or have been written-off (tlQtyWOff). (See 1.6.9 below for the use of this value.)
  3. **OSValueWithVAT**: Calculate the outstanding value on the line, taking any discounts into account, but excluding any items that have already been delivered (tlQtyDel) or have been written-off (tlQtyWOff). If Syss.IncludeVATInCommittedBalance is True then include VAT. If Syss.IncludeVATInCommittedBalance is False the result will be the same as 1.6.4 (see 1.6.10 below for the use of this value). Real48
  4. **OSValueExcludingWriteOffs**: Calculate the outstanding value on the line, taking any discounts into account, but excluding VAT and treating tlQtyDel as zero – this result will be the value excluding any items that have been written-off. (See 1.6.11 below for the use of this value.)
  5. **OSValueWithVATExcludingWriteOffs**: Calculate the outstanding value on the line, taking any discounts into account. If Syss.IncludeVATInCommittedBalance is True then include VAT. Treating tlQtyDel as zero – this result will be the value excluding any items that have been written-off. If Syss.IncludeVATInCommittedBalance is False, the result will be same as 1.6.6. (See 1.6.12 below for the use of this value.)
  6. If SOR.DETAILS.tlLocation matches the Stock Location filter, or if no Stock Location filter has been supplied, update the picked, delivered, and written-off values.
     1. SOR.DETAILS.tlQtyDel = SOR.DETAILS.tlQtyDel + SOR.DETAILS.tlQtyPicked. Real48
     2. SOR.DETAILS.tlQtyWOff = SOR.DETAILS.tlQtyWOff + SOR.DETAILS.tlQtyPickedWO. Real48
     3. SOR.DETAILS.tlQtyPicked = 0.
     4. SOR.DETAILS.tlQtyPickedWO = 0.
     5. **Store for sending to Stored Procedure:**
        1. **SOR.DETAILS.tlQtyDel**
        2. **SOR.DETAILS.tlQtyWOff**
        3. **SOR.DETAILS.tlQtyPicked**
        4. **SOR.DETAILS.tlQtyPickedWO**
  7. **TotalOSValue**: Accumulate the transaction amount delivered and/or written-off for this run, excluding VAT – calculate the new outstanding value using the same basis as 1.6.4 (but with the new QtyDel and QtyWOff values), and subtract the previous outstanding value (**OSValue**) originally calculated by 1.6.4.
  8. **TotalOSValueWithVAT**: Accumulate the transaction amount delivered and/or written-off for this run, including VAT if Syss.IncludeVATInCommittedBalance is True, using the same basis as 1.6.5 (but with the new QtyDel and QtyWOff values), and subtract the previous outstanding value (**OSValueWithVAT**) originally calculated by 1.6.5. Real48
  9. **WrittenOffValue**: Accumulate the transaction amount written off for this run, excluding VAT – calculate the new outstanding value using the same basis as 1.6.6 (but with the new QtyWOff value), and subtract the previous outstanding value (**OSValueExcludingWriteOffs**) originally calculated by 1.6.6.
  10. **WrittenOffValueWithVAT**: Accumulate the transaction amount written off for this run, including VAT if Syss.IncludeVATInCommittedBalance is True, using the same basis as 1.6.7 (but with the new QtyWOff value), and subtract the previous outstanding value originally calculated by 1.6.7.
  11. Calculate the amount still outstanding on the line: SOR.DETAILS.tlQty – (SOR.DETAILS.tlQtyDel + SOR.DETAILS.tlQtyWOff), rounded to EXCHQSS.NoQtyDec
      1. If this is zero (i.e. there is nothing outstanding), set SOR.DETAILS.tlLineType to ‘S’ (SOR) or ‘P’ (POR).
      2. Otherwise
         1. Record the fact that at least one line is still outstanding.
         2. Store SOR.DETAILS.tlLineDate as the potential DueDate, unless the potential DueDate (as read from other lines) is earlier than this date.
      3. **Store for sending to Stored Procedure:**
         1. **SOR.DETAILS.tlQty**
         2. **//SOR.DETAILS.tlLineType**
  12. [*InvCtSuU: Stock\_Deduct()*] Amend Stock allocation to hold the number of items that will still be allocated by the Order line after the current delivery and any write-off has been applied.
      1. Close any open Stock dialog.
      2. For Sales Orders only (not Purchase Orders) recalculate the Cost Price, and update the Stock quantity values. Ignore this for Bill of Material items (STOCK.stType = ‘M’) where STOCK. stShowKitOnSales is True.
         1. Calculate the unit price in Stock currency. Real48
            1. If SOR.DETAILS.tlUsePack is True, use STOCK.stCostPrice, otherwise use STOCK.stCostPrice divided by STOCK.stPurchaseUnits.
            2. [*CurrncyU: Currency\_ConvFT()*] Convert the Unit Price from the Stock currency to the base currency. Use the System Currency rates, selecting Daily Rate if EXCHQSS.TotalConv is ‘V’, otherwise using Company Rate.
            3. Convert the Unit Price from the base currency to the transaction currency. Use the System Currency rates, selecting Daily Rate if EXCHQSS.TotalConv is ‘V’, otherwise using Company Rate.
         2. If SOR.DETAILS.tlCostPrice is zero and this is not a Serial/Batch item (these are identified by STOCK.stValuationMethod of ‘R’ and a non-zero SOR.DETAILS.tlSerialQty), calculate SOR.DETAILS.tlCostPrice from the Unit Price multiplied by SOR.DETAILS.tlQtyMul.
            1. **Store for sending to Stored Procedure:**

**SOR.DETAILS.tlCostPrice**

* + 1. [*SysU2: FreeStock()*] Calculate the Free Stock
       1. If EXCHQSS.FreeExAll is True, then the free stock is STOCK.stQtyInStock
       2. Otherwise calculate the Free Stock by subtracting either the STOCK.stQtyPicked value (if Syss.UsePick4All is True) or the STOCK.stQtyAllocated value from the STOCK.QtyInStock value. Also subtract the STOCK. stQtyPickedWOR (if EXCHQSS.UseWIss4All is True) or STOCK.stQtyAllocWOR (if EXCHQSS.UseWIss4All is False). Real48
    2. Calculate how many items will still be outstanding on the line after taking the current delivery and write-offs into account – subtract SOR.DETAILS.tlQtyDel and SOR.DETAILS.tlQtyWOff from SOR.DETAILS.tlQty. Round the result to EXCHQSS.NoQtyDec decimal places. Multiply the result by SOR.DETAILS.tlQtyMul to get the Stock Required [Real48].
    3. For Sales (SOR) Bill of Material items only (STOCK.stType = 'M'), where either EXCHQSS.DeadBOM or STOCK.stShowKitOnSales is True, and Free Stock is less than the Stock Required, or the Stock Required is less than zero, adjust the SOR.DETAILS.tlStockDeductQty [Real48].
       1. If Free Stock is greater than zero, and Stock Required is greater than or equal to zero, set SOR.DETAILS.tlStockDeductQty to Free Stock, otherwise set SOR.DETAILS.tlStockDeductQty to zero.
    4. For Purchases (POR) Bill of Materials items only (STOCK.stType = ‘M’), where STOCK.stShowKitOnPurchases it True, set SOR.DETAILS.tlStockDeductQty to zero.
    5. For Sales Bill of Materials items with sufficient Free Stock, or for items which are not Bill of Materials items, set SOR.DETAILS.tlStockDeductQty to the Stock Required (i.e. the items still outstanding on the line).
    6. If STOCK.StockType is ‘P’, ‘M’, or ‘X’, update the Stock record with the new Allocated and Picked values.
       1. STOCK.stQtyAllocated = STOCK.stQtyAllocated + (SOR.DETAILS.tlQty \* TransactionTypeSignMultiplier). Real48
       2. STOCK.stQtyPicked = STOCK.stQtyPicked + ((SOR.DETAILS.tlQtyPicked \* SOR.DETAILS.tlQtyMul) \* TransactionTypeSignMultiplier). Real48
       3. **Store for sending to Stored Procedure:**
          1. **STOCK.stQtyAllocated**
          2. **STOCK.stQtyPicked**
    7. Update the FIFO valuation (these calculations make extensive use of Real48 values, although the fields in the database are Doubles), provided that SOR.DETAILS.tlYear is greater than EXCHQSS.AuditYr or EXCHQSS.AuditYr is zero.
       1. ***Note****: As far as I can work out, this procedure actually results in no actions being taken, as the values and settings at this point result in every active code path being by-passed.*
    8. If Stock Locations are in use, update the Stock Location record with the new Allocated and Picked values.
       1. STOCKLOCATION.lsQtyAlloc = STOCKLOCATION.lsQtyAlloc + SOR.DETAILS.tlQty. Real48
       2. STOCKLOCATION.lsQtyPicked = STOCKLOCATION.lsQtyPicked + SOR.DETAILS.tlQtyPicked \* SOR.DETAILS.tlQtyMul. Real48
       3. Save STOCKLOCATION.
       4. **Store for sending to Stored Procedure:**
          1. **STOCKLOCATION.stQtyAlloc**
          2. **STOCKLOCATION.stQtyPicked**
    9. Save STOCK.
  1. If Job Costing is active, and SOR.DETAILS.tlBOMKitLink is zero, and SOR.DETAILs.tlLocation matches the filter Stock Location (if any – this is set by the user when starting the process, and will be passed in to the routine), then update Job Actuals.
     1. [*InvLst2U: Update\_JobAct()*] Search for a Job Actual record (a JOBDET record with RecType of ‘J’ and SubType of ‘E’) matching the DETAILS.tlFolioRef and DETAILS.tlABSLineNo.
     2. If a record is found, but DETAILS.tlJobCode is empty, delete the Job Actual record and do not continue with updating Job Costing.
     3. If a record was not found, create a new JOBDET record, with a RecType of ‘J’ and a SubType of ‘E’.
     4. Copy the details from SOR.DETAILS to the new or existing JOBDET record.
        1. Copy JOBDET.JobCode from SOR.DETAILS.tlJobCode
        2. Copy JOBDET.var\_code10 from SOR.DETAILS.tlCurrency + SOR.DETAILS.tlYear + SOR.DETAILS.tlPeriod (i.e. 3 byte values)
        3. Copy JOBDET.LineFolio from SOR.DETAILS.tlFolioNum
        4. Copy JOBDET.LineNo from SOR.DETAILS.tlABSLineNo
        5. Copy JOBDET.LineORef from SOR.DETAILS.tlOurRef
        6. Copy JOBDET.StockCode from SOR.DETAILS.tlStockCode
        7. Copy JOBDET.JDate from SOR.DOCUMENT.thTransDate
        8. Copy JOBDET.OrigNCode from SOR.DETAILS.tlGLCode
        9. Copy JOBDET.JUseORate from SOR.DETAILS.tlUseOriginalRates
        10. Copy JOBDET.JPriceMulX from SOR.DETAILS.tlPriceMultiplier
        11. Copy JOBDET.Qty from SOR.DETAILS.tlQty, subtracting SOR.DETAILS.tlQtyDel and SOR.DETAILS.tlQtyWOff, rounding to EXCHQSS.NoQtyDec.
        12. Calculate JOBDET.Cost
            1. Calculate the quantity outstanding (JOBDET.Qty) on the line by adding SOR.DETAILS.tlQtyDel to SOR.DETAILS.QtyWOff, and then subtracting the result from SOR.DETAILS.tlQty. If SOR.DETAILS.tlUsePack is True then multiply the quantity outstanding by SOR.DETAILS.tlQtyMul. Real48
            2. **Net Value**: If SOR.DETAILS.tlPriceMultiplier is not zero, then the Net Value is SOR.DETAILS.tlNetValue multiplied by SOR.DETAILS.tlPriceMultiplier, otherwise it is simply SOR.DETAILS.tlNetValue. Real48
            3. Calculate the Price Per Unit:

If SOR.DETAILS.tlPrxPack is True, and SOR.DETAILS.tlQtyPack is not zero, and SOR.DETAILS.tlQtyMul is not zero:

If SOR.DETAILS.tlShowCase is True, then the Price Per Unit is the Net Value (as calculated by step 1.6.15.4.12.1 above), and the Quantity Outstanding (as calculated by step 1 above) should be recalculated by adding SOR.DETAILS.tlQtyDel to SOR.DETAILS.QtyWOff, subtracting the result from SOR.DETAILS.tlQty, then dividing the result by SOR.DETAILS.tlQtyPack.

If SOR.DETAILS.tlShowCase is False, then the Price Per Unit is SOR.DETAILS.tlQtyMul divided by SOR.DETAILS.tlQtyPack, multiplied by the Net Value (as calculated by step 1.6.15.4.12.1 above).

If SOR.DETAILS.tlPrxPack is False, or SOR.DETAILS.tlQtyPack is zero, or SOR.DETAILS.tlQtyMul is zero, then the Price Per Unit is the same as the Net Value (as calculated by step 1.6.15.4.12.1 above).

* + - * 1. Determine the Discount Basis. If SOR.DETAILS.tlVATIncValue is not zero and SOR.DETAILS.tlVATCode is ‘M’, then use this as the Discount Basis, otherwise use the Price Per Unit.
        2. [*ComnU2: Calc\_PAmountAD()*] Calculate the total Line Discount. There are three discounts on each line, held in SOR.DETAILS.tlDiscount, DETAILS.tlDiscount2, and SOR.DETAILS.tlDiscount3. Each has an accompanying flag/character field which either holds ‘%’ or blank: SOR.DETAILS.tlDiscFlg, SOR.DETAILS.tlDiscount2Chr, and SOR.DETAILS.tlDiscount3Chr. For each discount:

Subtract the current Line Discount (this will be zero for the first discount) from the Discount Basis to obtain a Revised Discount Basis.

If the discount flag is ‘%’ then multiply the Revised Discount Basis by the Discount, and add the result to the total Line Discount.

If the discount flag is blank, add the Discount to the total Line Discount.

Note that these value are all Real48, and that the returned total is also Real48, although it is then converted to Double.

* + - * 1. Calculate the final JOBDET.Cost by rounding the Quantity Outstanding to the 12 decimal places, rounding the Price Per Unit to the 12 decimal places for prices, multiplying these together, then subtracting the Line Discount.
      1. Copy JOBDET.JDDT from SOR.DETAILS.tlDocType.
      2. Copy JOBDET.ActCCode from SOR.DETAILS.tlAcCode
      3. Copy JOBDET.var\_code5 from SOR.DETAILS.tlAnalysisCode
      4. [*InvListU: GetJobMisc()*] Locate the Job Analysis record.
         1. Search JOBMISC for JOBMISC.RefPFix of ‘J’, JOBMISC.SubType of ‘A’, where JOBMISC.var\_code1 matches SOR.DETAILS.tlAnalysisCode.
         2. Copy JOBMISC.AnalHed to JOBDET.JType.
      5. Search JOBHEAD for the record matching JOBDET.JobCode.
      6. If JOBHEAD.ChargeType is 3, copy JOBDET.Charge from JOBDET.Cost multiplied by JOBDET.Qty, rounding to 2 decimal places.
      7. If JOBHEAD.ChargeType is 1, and JOBDET.StockCode is not empty (which should be guaranteed to be the case at this point), calculate the charge, based on the Stock Price.
         1. [*DiscU3U: Calc\_StockPrice()*] Scan for Stock discounts (see Appendix 2).
         2. [*DiscU3U: Calc\_AccDMatch()*] If discounts were found, check for In-Currency discounts.

For Suppliers, search EXSTKCHK for the in-currency discounts, where EXSTKCHK.RecMFix is ‘C’, EXSTKCHK.SubType is ‘S’, and EXSTKCHK.exstchkvar1 is CUSTSUPP.acCode (padded with spaces to 6 characters) + STOCK.stCode (padded with spaces to 16 characters) + SOR.DETAILS.tlCurrency converted to a Char (i.e. Currency 1 = CHAR(1), etc).

For Customers, search CUSTOMERDISCOUNT for the in-currency discounts where CUSTOMERDISCOUNT.CustCode is CUSTSUPP.acCode (padded with spaces to 6 characters), CUSTOMERDISCOUNT.StockCode is STOCK.stCode (padded with spaces to 16 characters), and CUSTOMERDISCOUNT.Currency is SOR.DETAILS.tlCurrency.

If a record is found where DOCUMENT.thTransDate is between the StartDate and EndDate values in CUSTOMERDISCOUNT (or the CStartDate and CEndDate values in EXSTKCHK), use this record.

If no matching record was found, but a record was found where CUSTOMERDISCOUNT.UseDates (or EXSTKCHK.CUseDates) was False, use this record instead (this is the default discount record).

* + - * 1. [*DiscU3U: Calc\_AccDMatch()*] If no In-Currency discounts were found, search for Consolidated (currency = 0) discounts instead, using the same rules as for In-Currency discounts above, except for the Currency.
        2. If no discounts were found, search for in-currency Quantity Breaks, but only if the Trader is a Supplier.

[*DiscU3U: Calc\_QtyBreak()*] Search the QTYBREAK table for a record where QTYBREAK.qbStockFolio matches STOCK.stFolioNum, and QTYBREAK.qbCurrency matches with SOR.DETAILS.tlCurrency.

If a record is found, compare QTYBREAK.qbQtyFrom and QTYBREAK.qbQtyTo with the Stock quantity, calculated either as JOBDET.Qty if STOCK. stShowQtyAsPacks is False, or as JOBDET.Qty multipled by SOR.DETAILS.tlQtyMul if STOCK.stShowQtyAsPacks is True.

If the quantity falls between the QtyFrom and QtyTo values (inclusive), and SOR.DETAILS.thTransDate falls between QTYBREAK.qbStartDate and QTYBREAK.qbEndDate (inclusive), then this Quantity Break should be used.

If no matching record could be found, but a Quantity Break was found where the quantity was valid but QTY.qbUseDates was False, use this record instead (this is the default Quantity Break for this range).

If no In-Currency Quantity Breaks were found, search for Consolidated (currency = 0) Quantity Breaks instead, using the same rules as for In-Currency discounts above, except for the Currency.

* + - * 1. For Customers, if CUSTOMERDISCOUNT.DiscountType is ‘Q’, or for Suppliers, if EXSTKCHK.QBType is ‘Q’ (I believe this Supplier check will always be False), the discount is a Quantity Break header, so check for Quantity Break lines in the QuantityBreak table.

[*DiscU3U: Calc\_QtyBreak()*] Search the QTYBREAK table for a record where QTYBREAK.qbFolio matches CUSTOMERDISCOUNT.QtyBreakFolio, and QTYBREAK.qbCurrency matches with SOR.DETAILS.tlCurrency.

If a record is found, compare QTYBREAK.qbQtyFrom and QTYBREAK.qbQtyTo with the Stock quantity, calculated either as JOBDET.Qty if STOCK. stShowQtyAsPacks is False, or as JOBDET.Qty multipled by SOR.DETAILS.tlQtyMul if STOCK.stShowQtyAsPacks is True.

If the quantity falls between the QtyFrom and QtyTo values (inclusive), and SOR.DETAILS.thTransDate falls between QTYBREAK.qbStartDate and QTYBREAK.qbEndDate (inclusive), then this Quantity Break should be used.

If no matching record could be found, but a Quantity Break was found where the quantity was valid but QTY.qbUseDates was False, use this record instead (this is the default Quantity Break for this range).

* + - * 1. If no In-Currency Quantity Breaks were found, search for Consolidated (currency = 0) Quantity Breaks instead, using the same rules as for In-Currency discounts above, except for the Currency.
        2. [*DiscU3U: Calc\_UPriceDisc()*] Calculate the unit price and discount. This will return the Unit Price, the Discount Amount, and either a ‘%’ character, if a percentage discount is being used, or else the Sales Band indicator. This character can be 0 if neither option is relevant. If a percentage discount is being used then the Discount Amount will be the percentage discount to apply, otherwise it will be the specific discount to apply.

If CUSTOMERDISCOUNT.DiscountType is ‘P’ (Special Price), the Unit Price is CUSTOMERDISCOUNT.Price converted from CUSTOMERDISCOUNT.Currency to JOBDET.Currency (as read from byte 1 of JOBDET.var\_code10. The Discount is zero.

If CUSTOMERDISCOUNT.DiscountType is ‘B’ (Band Price), calculate the Unit Price and Discount for the Sales Band.

[*InvLst3U: Stock\_LocFullSubst()*] If EXCHQSS.UseMLoc is True, and SOR.DETAILS.tlLocation is not empty, then look up the Stock Location record and use Sales Bands (Real48) from here instead of from the Stock record.

[*InvCt2SU: Get\_StkPrice()*] Read the Sales Price and Currency details from the relevant Sales Band (based on the CUSTOMERDISCOUNT.Band).

Convert the Sales Band Price from the Sales Band currency to JOBDET.tlCurrency.

If we found a Discount then take the Amount (CUSTOMERDISCOUNT.DiscountA), Percentage (CUSTOMERDISCOUNT.DiscountP), and Band (CUSTOMERDISCOUNT.Band) from the Customer Discount table.

If we found a Quantity Break then take the Amount (QUANTITYBREAK.qbDiscountAmount, Percentage (QUANTITYBREAK.qbDiscountPercent), and Band (QUANTITYBREAK.qbPriceBand) from the Quantity Break table.

If the Percentage value is not zero, return a ‘%’ character and the Percentage value as an actual percentage (i.e. divide it by 100).

If the Discount Amount is not zero, return it.

If Percentage and Discount Amount are both zero, return the Discount Band character, and leave the Percentage value and Discount Amount at zero.

If CUSTOMERDISCOUNT.DiscountType is ‘U’ (Mark-up), calculate the Unit Price with the Mark-up applied.

[*InvLst3U: Stock\_LocFullSubst()*] If EXCHQSS.UseMLoc is True, and SOR.DETAILS.tlLocation is not empty, then look up the Stock Location record and use Sales Bands (Real48) from here instead of from the Stock record.

[*FIFOL2U: FIFO\_GetCost()*] Calculate the Cost Price.

If a Stock Location Filter is set, locate the Location record. If LOCATION.loUseCPrice is set, take the values from the Location record instead of the Stock record.

If STOCK.stCalcPack is set, divide STOCK.stCostPrice by STOCK.stPurchaseUnits, otherwise simply use STOCK.stCostPrice.

Calculate the Unit Price by applying the Markup percentage to STOCK.stCodePrice.

If CUSTOMERDISCOUNT.DiscountType is ‘M’ (Margin), calculate the Unit Price with the Margin applied.

[*InvLst3U: Stock\_LocFullSubst()*] If EXCHQSS.UseMLoc is True, and SOR.DETAILS.tlLocation is not empty, then look up the Stock Location record and use Sales Bands (Real48) from here instead of from the Stock record.

[*FIFOL2U: FIFO\_GetCost()*] Calculate the Cost Price.

If a Stock Location Filter is set, locate the Location record. If LOCATION.loUseCPrice is set, take the values from the Location record instead of the Stock record.

If STOCK.stCalcPack is set, divide STOCK.stCostPrice by STOCK.stPurchaseUnits, otherwise simply use STOCK.stCostPrice.

Calculate the Unit Price by applying the Margin percentage to STOCK.stCodePrice.

* + - 1. Set JOBDET.Charge to the calculated Unit Price, adjusted by the Discount.
    1. If the transaction line no longer has an outstanding quantity, delete any existing Job Actuals record.
       1. **For the Stored Procedure, store the fact that the existing record must be deleted.**
    2. Otherwise, save the Job Actuals record.
    3. **Store for sending to the Stored Procedure:**
       1. **Full details of the JOBDET record.**
  1. Save DETAILS.
  2. Create the DETAILS record for the new SDN/PDN.
     1. Create a new, empty SDN.DETAILS record.
     2. [*SOPCT3U: Gen\_InvLine()*] Set the columns:
        1. SDN.DETAILS.tlCustCode = SDN.DOCUMENT.thCustCode
        2. SDN.DETAILS.tlCompanyRate = SDN.DOCUMENT.thCompanyRate
        3. SDN.DETAILS.tlDailyRate = SDN.DOCUMENT.thDailyRate
        4. SDN.DETAILS.tlTriRates = SDN.DOCUMENT.thCurrencyTriRate
        5. SDN.DETAILS.tlTriEuro = SDN.DOCUMENT.thCurrencyTriEuro
        6. SDN.DETAILS.tlTriInvert = SDN.DOCUMENT.thCurrencyTriInvert
        7. SDN.DETAILS.tlTriFloat = SDN.DOCUMENT.thCurrencyTriFloat
        8. SDN.DETAILS.tlUseOriginalRates = SDN.DOCUMENT.thUseOriginalRates
        9. Copy SDN.DETAILS.COSDailyRate from the consolidated (currency 0) System Currency Company Rate
        10. SDN.DETAILS.tlFolioNum = SDN.DOCUMENT.thFolioNum
        11. SDN.DETAILS.tlOurRef = SDN.DOCUMENT.thOurRef
        12. SDN.DETAILS.tlDocType = SDN.DOCUMENT.thDocType
        13. SDN.DETAILS.tlLineType = ‘O’ for SDN or ‘R’ for PDN
        14. SDN.DETAILS.tlLineNo = SDN.DOCUMENT.thNextLineNumber
        15. SDN.DETAILS.tlABSLineNo = SDN.DOCUMENT.thNextLineNumber
        16. SDN.DETAILS.tlYear = SDN.DOCUMENT.thYear
        17. SDN.DETAILS.tlPeriod = SDN.DOCUMENT.thPeriod
        18. SDN.DETAILS.tlLineDate = SDN.DOCUMENT.thTransDate
        19. SDN.DETAILS.tlCurrency = SDN.DOCUMENT.thCurrency
        20. SDN.DETAILS.tlDescription = SDN.DOCUMENT.thOurRef
        21. SDN.DETAILS.tlQty = SOR.DETAILS.tlQtyPicked
        22. SDN.DETAILS.tlQtyDel = 0
        23. SDN.DETAILS.tlQtyWOff = 0
        24. SDN.DETAILS.tlSOPFolioNum = SOR.DOCUMENT.thFolioNum
        25. Accumulate the Matching Value using the line value of the new line (the Matching Value is eventually used (in Rule 15) to create the Matching record for the Delivery Note. Real48
        26. SDN.DETAILS.tlSerialQty = SDN.DETAILS.tlQty \* SDN.DETAILS.tlQtyMul. Real48
     3. [*InvCtSuU: Stock\_Deduct()*] Take the item out of Stock.
        1. Close any open Stock dialog.
        2. Calculate the Cost Price, and update the Stock quantity values.
           1. Calculate the unit price in Stock currency. Real48
           2. If DETAILS.tlCostPrice is zero, calculate and store it from the unit price multiplied by DETAILS.tlQtyMul.
           3. Read STOCK.stQtyInStock, and get the Quantity Outstanding on the new line – this will actually be the delivered quantity.
           4. If STOCK.stQtyInStock is less than the Quantity Outstanding on the new line, set DETAILS.stStockDeductQty to STOCK.stQtyInStock, otherwise set DETAILS.stStockDeductQty to the Quantity Outstanding.
           5. STOCK.stQtyInStock = STOCK.stQtyInStock – DETAILS.tlStockDeductQty.
           6. **Store for sending to the Stored Procedure:**

**SDN.DETAILS.tlCostPrice**

**STOCK.stQtyInStock**

**SDN.DETAILS.tlStockDeductQty**

* + - 1. Update the FIFO valuation (these calculations make extensive use of Real48 values, although the fields in the database are Doubles).
         1. Determine the FIFO Mode, based on STOCK.stValuationMethod and update the FIFO records.
      2. If Stock Locations are in use, update the Stock Location record.
         1. STOCKLOCATION.lsQtyInStock = STOCKLOCATION.lsQtyInStock – DETAILS.tlStockDeductQty.
         2. Save STOCKLOCATION.
         3. **Store for sending to the Stored Procedure:**

**STOCKLOCATION.lsQtyInStock**

* + - 1. Save STOCK.
    1. [*MiscU: CalcVAT()*] Calculate the VAT for the line.
       1. Call the customisation hook point 190001, 1, if enabled, to allow plug-ins to provide the VAT calculation.
       2. If there was no hook point call, calculate the VAT normally.
       3. **Store for sending to the Stored Procedure:**
          1. **SDN.DETAILS.VAT**
          2. **SDN.DETAILS.IncNetValue**
          3. **SDN.DETAILS.NetValue**
          4. **SDN.DETAILS.VATCode**
    2. Save the new line.
    3. Update Job Actuals if this Transaction Line has a Job Code. Add a new Job Actuals record if one does not exist yet.
       1. Search for a Job Actual record matching the DETAILS.tlFolioRef and DETAILS.tlABSLineNo. If a record is found, but DETAILS.tlJobCode is empty, delete the Job Actual record.
       2. If a record was not found, create a new one.
       3. Copy the details from DETAILS to the new or existing record.
       4. Save the Job Actuals record.
    4. If this is a Purchase Order and tlPORPickSOR is True (1), and tlB2BLink is not zero, and tlB2BLineNo is not zero, then this is a back-to-back Purchase Order, so we need to locate the matching Sales Order and pick the equivalent quantity on it.
       1. [*DelvRunU: TSOPRunFrm.PickSORvPOR()*] Search for a Transaction Line matching with SDN.DETAILS.tlSOPFolioNum and SDN.DETAILS.tlSOPABSLineNo.
       2. Make sure that the Stock code on the Sales Order line matches the Stock Code on SDN.DETAILS. If Stock Locations are in use, also make sure that the Locations match.
       3. Locate the Sales Order transaction header.
       4. Calculate the new cost as Qty + QtyPWOff, multiplied by the Cost Price, on the original Sales Order line, and invert the sign.
       5. [*InvCtSuU: Stock\_Deduct()*] Subtract the original Sales Order tlQtyPick from STOCK.stQtyPicked.
       6. Set the tlQtyPick on the original Sales Order. If the amount received (tlQtyPick \* tlQtyMul on the Purchase Order line) plus the amount picked on the original Sales Order line is less than or equal to the outstanding amount on the Sales Order Line then set the Sales Order Line tlQtyPick value to the amount received plus the quantity picked. Real48
       7. Otherwise, set tlQtyPick on the original Sales Order line to the quantity outstanding on that line.
       8. [*InvCtSuU: Stock\_Deduct()*] Add the original Sales Order tlQtyPick to STOCK.stQtyPicked.
       9. If STOCK.stMultiBinMode is set, calculate the difference between the previous tlQtyPick value on the original Sales Order line to the new tlQtyPick value, and add this to the tlBinQty.
       10. Otherwise, calculate the difference between the previous tlQtyPick value on the original Sales Order line to the new tlQtyPick value, and add this to the tlSerialQty.
       11. If STOCK.stValuationMethod is anything other that ‘S’, recalculate the Cost Price:
           1. If the STOCK.stValuationMethod is ‘A’ or ‘E’:

If Stock Locations are in use, and a Stock Location is being used on the line, use the Cost Price and Currency from the Stock Location record.

If tlUsePack on the original Sales Order line is set, then calculate the Cost Price as the Cost Price multiplied by STOCK.stPurchaseUnits.

Otherwise use the existing Cost Price on the Stock item.

Convert the Cost Price from the Stock item currency to the currency on the original Sales Order line. Use the System Currency rates, selecting Daily Rate if EXCHQSS.TotalConv is ‘V’, otherwise using Company Rate.

* + - * 1. If the STOCK.stValuationMethod is not ‘A’ or ‘E’:

Calculate the discount on the line.

If tlShowCase is set on both the current Purchase Order line and the original Sales Order line:

If tlUsePack on the original Sales Order line is set, then calculate the Cost Price as the Purchase Order line tlNetValue less the discount, multiplied by the Purchase Order line tlQtyPack.

Otherwise use the Purchase Order line tlNetValue less the discount.

If tlShowCase is not set on one or other the current Purchase Order line of the original Sales Order line:

If tlUsePack on the original Sales Order line is set, then calculate the Cost Price as the Purchase Order line tlNetValue plus the Purchase Order line tlCostPrice, less the discount, multiplied by the Purchase Order line tlQtyPack.

Otherwise use the Purchase Order line tlNetValue plus the Purchase Order line tlCostPrice, less the discount.

* + - * 1. Round the Cost Price to EXCHQSS.NoCosDec decimal places.
        2. If the currency of the original Sales Order is not the same as the currency of the Purchase Order, set tlCOSDailyRate on the original Sales Order line to the system currency rate of the Sales Order line currency.
        3. Otherwise, copy the currency daily rate from the Purchase Order line to tlCOSDailyRate on the original Sales Order line. Use the System Currency rates, selecting Daily Rate if EXCHQSS.TotalConv is ‘V’ and the system Daily Rate is not zero, otherwise using Company Rate.
        4. Save the original Sales Order line.
        5. Calculate the new cost as Qty + QtyPWOff, multiplied by the Cost Price, on the original Sales Order line, and add this to the previously calculated new cost (from 1.6.18.1.4).
        6. [*DelvRunU: TSOPRunFrm.B2BSNos()*] If stValuationMethod is ‘R’, then this Stock item has Serial Numbers. If the difference between the previous tlQtyPick value on the original Sales Order line and the new tlQtyPick value is not zero, the Serial numbers need to be updated. The difference between the two QtyPick values is the number of Serial Numbers or Batch Numbers that are required.

Find all the SERIALBATCH entries against STOCK.stFolioNum. Scan through these to find an entry where the Purchase Order tlOurRef matches with the Serial Number InDoc and the tlABSLineNo matches with the Serial Number BuyLine, or the Purchase Order tlOurRef matches with the Serial number InOrdDoc and the tlABSLineNo matches with the Serial Number InOrdLine, and where the Serial Number Sold field is zero.

If the BatchRec field on the Serial record is True (non-zero), then this is actually a Batch item rather than a Serial item, so adjust the Batch numbers:

[*StkSerNU: Make\_BatchSetUse()*] Set the QtyUsed value on the Serial Record to the QtyUsed value plus the Numbers Required value calculated above.

If BuyQty minus QtyUsed is less than or equal to zero, set Sold to True (1), otherwise set it to False (0).

Save the Serial record.

Create a new Serial record, copying the details from the existing one. Set the other details:

Set Sold to True (1)

Copy SERIALBATCH.OutDoc from SOR.DOCUMENT.thOurRef

Copy SERIALBATCH.SoldLine from SOR.DETAILS.tlABSLineNo

If EXCHQSS.UseMLoc is set, copy SERIALBATCH.OutMLoc from SOR.DETAILS.tlLocation

Copy SERIALBATCH.DateOut from SOR.DETAILS.tlLineDate

Copy SERIALBATCH.SaleCurrency from SOR.DETAILS.tlCurrency

Copy SERIALBATCH.SalePrice from SOR.DETAILS.

If tlUsePack on the original Sales Order line is set, then calculate the SERIALBATCH.SalePrice as SOR.DETAILS.tlNetValue multiplied by SOR.DETAILS.tlQtyPack.

Otherwise use SOR.DETAILS.tlNetValue.

Copy SERIALBATCH.QtyUsed from the Numbers Required value calculated above.

Copy SERIALBATCH.ChildNFolio from SERIALBATCH.NoteFolio.

Set SERIALBATCH.BatchChild to True (1).

Save the SERIALBATCH record.

If the BatchRec field on the Serial record is False (zero), then this is a Serial item, so adjust the Serial numbers:

[*StkSerNU: SERN\_SetUse()*] Set the SERIALBATCH.Sold flag to True (1).

Copy SERIALBATCH.OutDoc from SOR.DOCUMENT.thOurRef.

Copy SERIALBATCH.SoldLine from SOR.DETAILS.tlABSLineNo

If EXCHQSS.UseMLoc is set, copy SERIALBATCH.OutMLoc from SOR.DETAILS.tlLocation

Copy SERIALBATCH.DateOut from SOR.DETAILS.tlLineDate

Copy SERIALBATCH.SaleCurrency from SOR.DETAILS.tlCurrency

Copy SERIALBATCH.SalePrice from SOR.DETAILS.

Save the SERIALBATCH record.

Find the next SERIALBATCH record, unless we have now found the required number of Serial Numbers or Batch Number, or the next SERIALBATCH record has the Sold flag set to True (in which case we have run out of available records).

* + - * 1. If stValuationMethod is not ‘R’, check to see if this Stock item has Bin Numbers.
        2. [*DelvRunU: TSOPRunFrm.B2BBNos()*] Locate any Bin records against the Stock item, based on the Stock Folio, the Location, and whether or not the Sold flag is set
        3. Scan through these to find records where MLOCSTK.brInDoc matches with thOurRef on the Purchase Order and MLOCSTK.brBuyLine matches with the tlABSLineNo on the Purchase Order Line, or where MLOCSTK.brInOrdDoc matches with thOurRef on the Purchase Order and MLOCSTK.brInOrdLine matches with tlABSLineNo on the Purchase Order line, and where the MLOCSTK.brSold flag is zero.
        4. For each record, if the difference between MLOCSTK.brBuyQty and MLOCSTK.brQtyUsed is greater than or equal to the number of items required less any items that we have taken from other Bin records, set the number of items to the number of remaining items required.
        5. Otherwise, set the number of items to the difference between MLOCSTK.brBuyQty and MLOCSTK.brQtyUsed (i.e. the total number of items taken from this Bin).
        6. Accumulate the number of items taken so far.
        7. [*StkBinU: Make\_BinSetUse()*] Update the Bin record:

Add the number of items taken from this Bin to MLOCSTK.brQtyUsed.

If MLOCSTK.brBuyQty - MLOCSTK.brQtyUsed it less than or equal to zero (i.e. we have sold all the items from this Bin), set the MLOCSTK.brSold flag to True (1), otherwise set it to False (0).

Set MLOCSTK.VarCode2 to FullNomKey(MLOCSTK.brStockFolio) + CHAR(MLOCSTK.brSold) + MLOCSTK.brInMLoc, MLOCSTK.brPriority, MLOCSTK.brDateIn, MLOCSTK.varCode1.

If MLOCSTK.varCode1 (the Bin Location Code) does not match with the STOCK.stBinLocation then this is one of the ‘child’ Bin records. If EXCHQSS.KeepBinHist is False (0), delete this record, otherwise store this record.

Copy the details of the MLOCSTK record to a new record (regardless of whether or not the record was deleted), and update the record with new details:

Set MLOCSTK.brSold to True (1).

Copy MLOCSTK.brOutDoc from the original Sales Order thOurRef.

Copy MLOCSTK.brSoldLine from the original Sales Order line tlABSLineNo.

Copy MLOCSTK.brOutMLoc from the original Sales Order line tlLocation.

Copy MLOCSTK.brDateOut from the original Sales Order line tlLineDate.

Copy MLOCSTK.brCurrSell from the original Sales Order line tlCurrency.

If tlUsePack on the original Sales Order line is set, then calculate the MLOCSTK.brBinSell as SOR.DETAILS.tlNetValue multiplied by SOR.DETAILS.tlQtyPack.

Set MLOCSTK.brQtyUsed to the number of items taken from this Bin.

Set MLOCSTK.brBatchChild to True (1).

Set MLOCSTK.VarCode2 to FullNomKey(MLOCSTK.brStockFolio) + CHAR(MLOCSTK.brSold) + MLOCSTK.brInMLoc, MLOCSTK.brPriority, MLOCSTK.brDateIn, MLOCSTK.varCode1.

Add the new MLOCSTK record.

Find the next MLOCSTK, unless we have now taken the required number of Bin items, or the next MLOCSTK record has the brSold flag set to True (in which case we have run out of available records).

* + - 1. [*SOPCT3U: SOP\_SeekDescLines()*] On the original Sales Order, search for any description lines which follow the current order line and make sure that they are also picked:
         1. Searching by tlFolioRef in the original Sales Order, ordered by tlLineNo, find all the lines that follow the current line, until we reach the end of the lines for the transaction, or we find a line which has a Stock Code (and hence we have reached the end of any description lines.
         2. If the line has a non-zero tlBOMKitLink value, update the line:

Copy tlQtyPicked value from the original Sales Order line to this line.

Set tlQtyPickedWO to 0.

Set tlSOPABSLineNo to 0.

Save the line.

* + - 1. Add the new cost (from 1.6.18.1.11.7) to thTotalCost on the original Sales Order transaction header.
      2. If tlBOMKitLink on the original Sales Order lines is not empty, and does not match the thFolioNum on the transaction header, this is an exploded BoM line, and we need to find the parent line and update it:
         1. [*DelvRunU: TSOPRunFrm.Auto\_PickKit()*] Starting from the current original Sales Order line, work backwards up the lines, retrieving the Stock item for each line.
         2. If the STOCK.stValuationMethod is not ‘R’, and the line number is greater than 1 and STOCK.stType = ‘M’ (Bill of Materials), and STOCK.stFolioNum matches tlBOMKitLink on the original Sales Order Line, we have found the parent line for this BoM component.
         3. If this line is not the parent line, and the tlBOMKitLink on this line is zero, we have failed to find a matching BoM header line, so do not continue.
         4. If we found a BoM header line, work through the following transaction lines until we reach a line where tlBOMKitLink does not match STOCK.stFolioNum, or we reach the end of the transaction.
         5. For each line found, work out how many of the BoM items could be built using the picked items on the line. The tlQty is the total quantity of this item that would be required to fulfil the number of items required by the BoM. The Exchequer version divides the tlQty by the tlQty on the BoM, to give the number of items required for a single BoM. It then divides the tlQtyPicked by this number to find out how many items could be built.
         6. Keep a record of the lowest number found – this is the maximum quantity of the BoM that could be built using the items picked on the components.
         7. If this is greater than the tlQtyPicked on the BoM line, we have enough component items to fulfil the current requirements, so update the BoM line:

[*InvCtSuU: Stock\_Deduct()*] Subtract the original Sales Order tlQtyPicked from STOCK.stQtyPicked.

[*StkBinU: Auto\_PickBin(), mode = 1*] Internally keep track of the Quantity Required From Bins, initialising this to tlQtyPicked from the original Sales Order line.

Search MLOCSTK for the Bin records (RecPFix = ‘I’ and SubType = ‘R’) where varCode2 matches with the Stock Folio number. If EXCHQSS. FiltSNoBinLoc is True (1), then restrict the search to records where varCode2 bytes 6 to 8 match with tlLocation in the original Sales Order line.

For each record, if brSold is True, brBatchChild is True, brOutOrdDoc matches thOurRef from the original Sales Order, and brSoldLine matches tlABSLineNo on the original Sales Order line, then take a note of brQtyUsed (this is needed in a subsequent step), and delete the record.

[*StkBinU: Un\_UseBin()*] Search for an MLOCSTK record which matches with STOCK.stFolioNum, where brBatchRec is True (1), brBatchChild is False (0), brInDoc matches brInDoc of the record we have just deleted, and brBuyLine matches brBuyLine of the record we have just deleted.

If a matching record is found, this is the header Bin record, so update it:

Subtract the brQtyUsed of the record we’ve just deleted from the brQtyUsed on this record.

If the brQtyUsed on this record is less than or equal to zero, set brSold to True (1), otherwise set it to False (0).

Set VarCode2 on this record to FullNomKey(MLOCSTK.brStockFolio) + CHAR(MLOCSTK.brSold) + MLOCSTK.brInMLoc, MLOCSTK.brPriority, MLOCSTK.brDateIn, MLOCSTK.varCode1.

Save the record

If a matching record was not found, create a new one:

Copy the basic details from the record we’ve just deleted, then update the other details:

Set brBuyQty to brQtyUsed from the record we’ve just deleted.

Set brQtyUsed to zero.

Set brBatchChild to False (0).

Set brSold to False.

Set VarCode2 to FullNomKey(brStockFolio) + CHAR(brSold) + brInMLoc, brPriority, brDateIn, varCode1.

Set brOutDoc to a string of spaces.

Set btOurOrdDoc to a string of spaces.

Set brSoldLine to zero.

Set brOutOrdLine to zero.

Set brBinSell to zero.

Set brDateOut to a string of spaces.

Save the new record.

Reduce the Quantity Required From Bins by the brQtyUsed value from the record we deleted.

Get the next Bin record, if any, unless the Quantity Required From Bins value is less than or equal to zero, in which we have found all the items we need and can stop.

* + - * 1. Set tlQtyPick to the number of items that could be built (see 1.6.17.7.14.6).
        2. [*InvCtSuU: Stock\_Deduct()*] Add the original Sales Order tlQtyPick to STOCK.stQtyPicked.
        3. [*StkBinU: Auto\_PickBin(), mode = 0*] Internally keep track of the Quantity Required From Bins, initialising this to tlQtyPicked from the original Sales Order line.
        4. Search MLOCSTK for the Bin records (RecPFix = ‘I’ and SubType = ‘R’) where varCode2 matches with the Stock Folio number. If EXCHQSS. FiltSNoBinLoc is True (1), then restrict the search to records where varCode2 bytes 6 to 8 match with tlLocation in the original Sales Order line.
        5. For each record, if brSold is False, brBatchChild is False, and brHoldFlag is not 1 or 2, we can use this Bin:

Calculate the quantity available by subtracting btQtyUsed from brBuyQty. If the result is less than the quantity required, use the quantity available, otherwise use the quantity required.

[*StkBinU: Make\_BinSetUse ()*] Add the number of items taken from this Bin to MLOCSTK.brQtyUsed.

If MLOCSTK.brBuyQty - MLOCSTK.brQtyUsed it less than or equal to zero (i.e. we have sold all the items from this Bin), set the MLOCSTK.brSold flag to True (1), otherwise set it to False (0).

Set MLOCSTK.VarCode2 to FullNomKey(MLOCSTK.brStockFolio) + CHAR(MLOCSTK.brSold) + MLOCSTK.brInMLoc, MLOCSTK.brPriority, MLOCSTK.brDateIn, MLOCSTK.varCode1.

If MLOCSTK.varCode1 (the Bin Location Code) does not match with the STOCK.stBinLocation then this is one of the ‘child’ Bin records. If EXCHQSS.KeepBinHist is False (0), delete this record, otherwise store this record.

Copy the details of the MLOCSTK record to a new record (regardless of whether or not the record was deleted), and update the record with new details:

Set MLOCSTK.brSold to True (1).

Copy MLOCSTK.brOutDoc from the original Sales Order thOurRef.

Copy MLOCSTK.brSoldLine from the original Sales Order line tlABSLineNo.

Copy MLOCSTK.brOutMLoc from the original Sales Order line tlLocation.

Copy MLOCSTK.brDateOut from the original Sales Order line tlLineDate.

Copy MLOCSTK.brCurrSell from the original Sales Order line tlCurrency.

If tlUsePack on the original Sales Order line is set, then calculate the MLOCSTK.brBinSell as SOR.DETAILS.tlNetValue multiplied by SOR.DETAILS.tlQtyPack.

Set MLOCSTK.brQtyUsed to the number of items taken from this Bin.

Set MLOCSTK.brBatchChild to True (1).

Set MLOCSTK.VarCode2 to FullNomKey(MLOCSTK.brStockFolio) + CHAR(MLOCSTK.brSold) + MLOCSTK.brInMLoc, MLOCSTK.brPriority, MLOCSTK.brDateIn, MLOCSTK.varCode1.

Add the new MLOCSTK record.

Reduce the Quantity Required From Bins by the brQtyUsed value from the record we deleted.

* + - * 1. Get the next Bin record, if any, unless the Quantity Required From Bins value is less than or equal to zero, in which case we have found all the items we need and can stop.
        2. If the Quantity Required From Bins value is greater than zero, we need to adjust the quantity picked on the line:

[*InvCtSuU: Stock\_Deduct()*] Subtract the original Sales Order tlQtyPick from STOCK.stQtyPicked.

Subtract the remaining Quantity Required From Bins value from tlQtyPicked on the original Sales Order line.

If the result is less than zero, set it to zero.

[*InvCtSuU: Stock\_Deduct()*] Add the original Sales Order tlQtyPick to STOCK.stQtyPicked.

* + - * 1. Save the transaction line.
  1. [*DelvRunU: TxFrSNos()*] Update Serial Numbers
     1. Only continue if SDN.DETAILS.tlSerialQty is not zero.
     2. Locate the Stock record.
     3. Only continue if STOCK.stValuationMethod is ‘R’.
     4. For each SerialBatch record against STOCK.stStockFolio:
        1. For Sales, only include Serial records where SDN.DETAILS.tlABSLineNo matches with SERIALBATCH.SoldLine. For Purchases, only include Serial records where SDN.DETAILS.tlABSLineNo matches with SERIALBATCH.BuyLine.
        2. For sales:
           1. Copy SERIALBATCH.OutDoc to SERIALBATCH.OutOrdDoc
           2. Copy SERIALBATCH.SoldLine to SERIALBATCH.OutOrdLine
           3. Copy SDN.DOCUMENT.thOurRef to SERIALBATCH.OutDoc
        3. For Purchases:
           1. Copy SERIALBATCH.InDoc to SERIALBATCH.InOrdDoc
           2. Copy SERIALBATCH.BuyLine to SERIALBATCH.InOrdLine
           3. Calculate the discount amount for the line. Real48
           4. Calculate SERIALBATCH.CostPrice
           5. Copy SDN.DOCUMENT.thCurrency to SERIALBATCH.CurCost
           6. Copy SDN.DETAILS.tlCompanyRate to SERIALBATCH.SerCompanyRate
           7. Copy SDN.DETAILS.tlDailyRate to SERIALBATCH.SerDailyRate
           8. Copy SDN.DETAILS.tlTriRates to SERIALBATCH.TriRates
           9. Copy SDN.DETAILS.tlTriEuro to SERIALBATCH.TriEuro
           10. Copy SDN.DETAILS.tlTriInvert to SERIALBATCH.TriInvert
           11. Copy SDN.DETAILS.tlTriFloat to SERIALBATCH.TriFloat
           12. If SERIALBATCH.SerCompanyRate is zero, copy it from the System Currency rate.
           13. Copy SDN.DOCUMENT.thOurRef to SERIALBATCH.InDoc
        4. Save the SERIALBATCH record.
        5. If this is a Sales Transaction, or is a Purchase Transaction and SERIALBATCH.BatchChild is False, update the count of Serial records found.
           1. If SERIALBATCH.BatchRec is True, for Sales add SERIALBATCH.QtyUsed, otherwise add SERIALBATCH.BuyQty.
           2. If SERIALBATCH.BatchRec is False, simply increment the count.
        6. If the Serial/Batch count is equal to or greater than SDN.DETAILS.tlSerialQty, we have found all the Serial/Batch records we need, so exit, otherwise get the next SERIALBATCH record.
  2. [*DelvRunU: TxFrBNos()*] Update Multi-Bins
     1. Only continue if SDN.DETAILS.tlSerialQty is zero, or if STOCK.stValuationMethod is not ‘R’.
     2. Do not continue if STOCK.stMultiBinMode is False.
     3. Do not continue if SDN.DETAILS.tlBinQty is zero.
     4. For Sales transactions, scan MLOCSTK for all records where RecPFix is ‘I’, SubType is ‘R’, and MLOCSTK.varCode2 matches with STOCK.stStockFolio plus SDN.DETAILS.tlLocation plus CHAR(1).
     5. For Purchase transactions, scan MLOCSTK for all records where RecMFix is ‘I’, SubType is ‘R’, and MLOCSTK.varCode2 matches with STOCK.stStockFolio.
     6. For each record found:
        1. For Sales, only include Bin records where SDN.DETAILS.tlABSLineNo matches with SERIALBATCH.SoldLine. For Purchases, only include Bin records where SDN.DETAILS.tlABSLineNo matches with SERIALBATCH.BuyLine.
        2. For Sales:
           1. Copy MLOCSTK.brOutDoc to MLOCSTK.brOutOrdDoc
           2. Copy MLOCSTK.brSoldLine to MLOCSTK.brOutOrdLine
           3. Copy SDN.DOCUMENT.thOurRef to MLOCSTK.brOutDoc
        3. For Purchases:
           1. Copy MLOCSTK.brInDoc to MLOCSTK.brInOrdDoc
           2. Copy MLOCSTK.BuyLine to MLOCSTK.brInOrdLine
           3. Calculate the discount amount for the line. Real48
           4. Calculate MLOCSTK.brBinCost
           5. Copy SDN.DOCUMENT.thCurrency to MLOCSTK.brCurCost
           6. Copy SDN.DETAILS.tlCompanyRate to MLOCSTK.brSerCompanyRate
           7. Copy SDN.DETAILS.tlDailyRate to MLOCSTK.brSerDailyRate
           8. Copy SDN.DETAILS.tlTriRates to MLOCSTK.brTriRates
           9. Copy SDN.DETAILS.tlTriEuro to MLOCSTK.brTriEuro
           10. Copy SDN.DETAILS.tlTriInvert to MLOCSTK.brTriInvert
           11. Copy SDN.DETAILS.tlTriFloat to MLOCSTK.brTriFloat
           12. If MLOCSTK.brSerCompanyRate is zero, copy it from the System Currency rate.
           13. Copy SDN.DOCUMENT.thOurRef to MLOCSTK.brInDoc
        4. Save the MLOCSTK record.
        5. If this is a Sales transaction, update the count of Bin records found by adding MLOCSTK.brQtyUsed.
        6. If this is a Purchase transaction and MLOCSTK.brBatchChild is False, update the count of Bin records found by adding MLOCSTK.brBuyQty.
        7. If the Bin count is equal to or greater than SDN.DETAILS.tlBinQty, we have found all the Bin records we need, so exit, otherwise get the next MLOCSTK record.
  3. Get the next transaction line.

1. [*SOPCT3U: Reveal\_SOPDoc()*] Set the Run Number to -40 for Sales Deliveries or -50 for Purchase Deliveries, and set the DETAILS.acCustSupp flag appropriately (‘C’ or ‘S’).
2. Allow VAT to be over-ridden by plug-ins.
3. [*MiscU: CalcInvTotals()*] Calculate the totals for the new Delivery transaction, totalling the values from each Transaction Line.
   1. For each Transaction Line:
      1. Calculate and accumulate the line totals.
      2. Recalculate the total VAT.
      3. Recalculate the Discount Amount.
      4. Update any PPD Totals.
   2. Delete any lines with a line number of -1 (BoM lines).
   3. For each Transaction Line:
      1. Make adjustments for Bill of Material items.
4. Set the allocation status to #0.
5. If the original Sales Order was using Order Payments, set the Order Payments status on the new transaction (to opeDeliveryNote).
6. Save the Transaction Header.
7. If a ‘Delivery’ hook point has been assigned, call it now.
8. If an ‘Invoice Created’ hook point has been assigned, call it now.
9. [*LedgSupU: Match\_Payment()*] Create a new Match Payment record in the FinancialMatching table.
   1. FINANCIALMATCHING.DocCode = SDN.DOCUMENT.thOurRef
   2. FINANCIALMATCHING.PayRef = SDN.DOCUMENT.thRemitNo
   3. FINANCIALMATCHING.AltRef = SDN.DOCUMENT.thYourRef
   4. FINANCIALMATCHING.SettledVal = Matching Value converted to Base Currency
   5. FINANCIALMATCHING.OwnCVal = Matching Value (in transaction currency)
   6. FINANCIALMATCHING.MCurrency = SDN.DOCUMENT.thCurrency
   7. Save the FinancialMatching record.
10. [*NoteSupU: CopyNoteFolio()*] Find any TransactionNote records where NoteFolio matches with SOR.DOCUMENT.thFolioNum. For each record:
    1. If TRANSACTIONNOTE.NType is not ‘3’:
       1. Create a new record, copying all the details over.
       2. TRANSACTIONNOTE.NoteFolio = SDN.DOCUMENT.thFolioNum
       3. Save the new record.
11. [*LettrDlg: CopyLinkFolio()*] Find any Letters & Links records against the original transaction and copy them to the delivery note.
12. If the original transaction is still outstanding, set the thBatchLink field to ‘SOR’ + thDueDate.
13. Subtract the value calculated by rule 1.6.9 from thTotalOrderOS on the original transaction.
14. Save the original transaction.
15. Add an audit note for the transaction.
16. If an Order Payments refund is required, as determined under rule 1.5.1, display the Order Payments Refund Window to allow the user to apply the refund.
17. Delete any lines with a line number of -1 (BoM lines).
18. For each Transaction Line:

1.23.1 Make adjustments for Bill of Material items.

1. Post to History the written-off amount calculated by 1.6.12.

# Appendix 1: 900 Codes

(Copied from the Exchequer Help file)

If you wish to set a specific Date in each month for payment terms, or in an automatic transaction, you can use the special '900 codes'. Similarly, you can use 800, 700, 600 and 500 in the same way.

Instead of entering the number of days, enter a 3-digit code to give you a specific date, based on the examples below:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Code | Due date | Examples | Invoice date | Due date |
| 9dd | Next month. | 915  920 | 23 Jan  23 Jan | 15 Feb  20 Feb |
| 999 | End of next month. |  | 23 Jan  23 Feb | 28 Feb  31 Mar |
| 8dd | 2 months. | 810 | 23 Jan | 10 Mar |
| 7dd | 3 Months. | 710 | 23 Jan | 10 Apr |
| 6dd | 4 months. | 610 | 23 Jan | 10 May |
| 5dd | 5 months. | 510 | 23 Jan | 10 June |

# Appendix 2: Stock Discounts

The following SQL queries are currently used to scan for discounts against a Stock Code. There are two separate queries because Customer Discounts are now in a separate table, whereas Supplier Discounts are still in the EXSTKCHK table. The % entries are place-holders which are replaced with the required parameters at runtime.

## Suppliers

WITH temp\_StockTree (tmpStockCode, tmpParentCode) AS

(

SELECT stCode, stParentCode

FROM [COMPANY].Stock

WHERE stCode = '%s'

UNION ALL

SELECT stcode, b.stParentCode

FROM temp\_StockTree AS a, [COMPANY].Stock AS b

WHERE cast(a.tmpParentCode as varbinary(16)) = b.stCode

)

SELECT tmpStockCode,

(Select Count(RecMFix)

From [COMPANY].ExStkChk

Where (RecMFix = C) And (SubType = '%s')

And (ExStChkVar1Computed like '%CustCode' + tmpStockCode + '%s')

And ((CUseDates = 0) Or ((CUseDates = 1) And (CStartD <= '%s') And (CEndD >= '%s')))

And (CustQBCurr = %d)

) As NumInCcyDiscounts,

(Select Count(RecMFix)

From [COMPANY].ExStkChk

Where (RecMFix = C) And (SubType = '%s')

And (ExStChkVar1Computed like '%CustCode' + tmpStockCode + '%s')

And ((CUseDates = 0) Or ((CUseDates = 1) And (CStartD <= '%s') And (CEndD >= '%s')))

And (CustQBCurr = 0)

) As NumConsolDiscounts

FROM temp\_StockTree

## Customers

WITH temp\_StockTree (tmpStockCode, tmpParentCode) AS

(

SELECT stCode, stParentCode

FROM [COMPANY].Stock

WHERE stCode = '%s'

UNION ALL

SELECT stcode, b.stParentCode

FROM temp\_StockTree AS a, [COMPANY].Stock AS b

WHERE cast(a.tmpParentCode as varbinary(16)) = b.stCode

)

SELECT tmpStockCode,

(Select Count(CustCode)

From [COMPANY].CustomerDiscount

Where

(CustCode = '%s')

And (StockCode LIKE tmpStockCode + '%CustCode')

And ((UseDates = 0) Or ((UseDates = 1) And (StartDate <= '%s') And (EndDate >= '%s')))

And (Currency = %d)

) As NumInCcyDiscounts,

(Select Count(CustCode)

From [COMPANY].CustomerDiscount

Where

(CustCode = '%s')

And (StockCode LIKE tmpStockCode '%s')

And ((UseDates = 0) Or ((UseDates = 1) And (StartDate <= '%s') And (EndDate >= '%s')))

And (Currency = 0)

) As NumConsolDiscounts

FROM temp\_StockTree

# Overview Flowchart

