# Order Processing

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

***Note****: In spite of appearances, this is a high-level overview of the process. Most of the steps have many lower level details and options based on the current system-settings, the type of Stock Item being dealt with (especially if this is a Bill of Materials item), and other details on the Transaction and Transaction Lines.*

*Items highlighted in red are those which will be problematic for SQL Stored Procedures because they require GUI interaction, call hook-points, or use Real48 values.*

1. **[*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)

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.

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.

If the number has less than 6 digits, left-pad it with zeroes to 6 digits.

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.

Look up the ‘AFL’ record in EXCHQNUM.

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:

SDN.DOCUMENT.thNetValue

SDN.DOCUMENT.thTotalLineDiscount

SDN.DOCUMENT.thSettleDiscAmount

SDN.DOCUMENT.thPPDGoodsValue

SDN.DOCUMENT.thPPDVATValue

SDN.DOCUMENT.thSettleDiscTaken

SDN.DOCUMENT.thSettleDiscPerc

SDN.DOCUMENT.PostDiscAm

SDN.DOCUMENT.thPostDiscTaken

SDN.DOCUMENT.thAmountSettled

SDN.DOCUMENT.thVariance

SDN.DOCUMENT.thTotalReserved

SDN.DOCUMENT.thTotalInvoiced

SDN.DOCUMENT.thTotalOrdered

SDN.DOCUMENT.thTotalCost

SDN.DOCUMENT.thCurrSettled

SDN.DOCUMENT.thReValueAdj

* + - 1. 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).
      2. Clear SDN.DOCUMENT.thVATPostDate to a string of spaces.
      3. Clear SDN.DOCUMENT.thPostedDate to a string of spaces.
      4. Set SDN.DOCUMENT.thOldCompanyRate and SDN.DOCUMENT.thOldDailyRate to zero.
      5. If SDN.DOCUMENT.thFixedRate is 0 (false), set SDN.DOCUMENT.thDailyRate to the system daily rate for the currency.
      6. [*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.*
      7. Set SDN.DOCUMENT.thCompanyRate to zero.
      8. Copy SDN.DOCUMENT.thVATCompanyRate and SDN.DOCUMENT.thVATDailyRate from the System VAT Currency rates (EXCHQSS.VATCurr holds the System VAT Currency).
      9. Copy SDN.DOCUMENT.thOriginalCompanyRate and SDN.DOCUMENT.thOriginalDailyRate from the System Currency rates for the transaction currency.
      10. Set SDN.DOCUMENT.thUseOriginalRates to zero.
      11. 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.
      12. Store the new Transaction Header.
      13. [*AuditNotes: TAuditNote.WriteAuditNote()*] Add an Audit Note.
          1. Add a new record to the TransactionNote table

Set TRANSACTIONNOTE.NoteFolio to SDN.DOCUMENT.thFolioNum

Set TRANSACTIONNOTE.NoteDate to the current date, formatted as YYYYMMDD.

Set TRANSACTIONNOTE.NoteType to 2.

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.

Set TRANSACTIONNOTE.NoteUser to the current user’s login name.

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:**

**STOCK.stQtyAllocated**

**STOCK.stQtyPicked**

* + - 1. 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.*
      2. 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:**

**STOCKLOCATION.stQtyAlloc**

**STOCKLOCATION.stQtyPicked**

* + - 1. Set DETAILS.tlStockDeductQty to zero.
      2. **Store for sending to Stored Procedure:**
         1. **DETAILS.tlStockDeductQty**
      3. Save STOCK.
    1. **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.)
    2. **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
    3. **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.)
    4. **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.)
    5. 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**
    6. **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.
    7. **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
    8. **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.
    9. **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.
    10. 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**
    11. [*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

If SOR.DETAILS.tlUsePack is True, use STOCK.stCostPrice, otherwise use STOCK.stCostPrice divided by STOCK.stPurchaseUnits.

[*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.

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.

* + - * 1. 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.

**Store for sending to Stored Procedure:**

**SOR.DETAILS.tlCostPrice**

* + - * 1. [*SysU2: FreeStock()*] Calculate the Free Stock

If EXCHQSS.FreeExAll is True, then the free stock is STOCK.stQtyInStock

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

* + - * 1. 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. The result is the Stock Required [Real48].
        2. 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].

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.

* + - * 1. For Purchases (POR) Bill of Materials items only (STOCK.stType = ‘M’), where STOCK.stShowKitOnPurchases it True, set SOR.DETAILS.tlStockDeductQty to zero.
        2. For Sales Bill of Materials items which which 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).
        3. STOCK.stQtyAllocated = STOCK.stQtyAllocated + DETAILS.tlStockDeductQty.
        4. **Store for sending to Stored Procedure:**

**SOR.DETAILS.tlStockDeductQty**

**STOCK.stQtyAllocated**

* + - 1. Update the FIFO valuation (these calculations make extensive use of Real48 values, although the fields in the database are Doubles).
         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.*
      2. If Stock Locations are in use, update the Stock Location record with the new Allocated and Picked values.
         1. STOCKLOCATION.lsQtyAlloc = STOCKLOCATION.lsQtyAlloc + DETAILS.tlStockDeductQty.
         2. STOCKLOCATION.lsQtyPicked = STOCKLOCATION.lsQtyPicked + DETAILS.QtyPicked \* DETAILS.QtyMul.
         3. Save STOCKLOCATION.
         4. **Store for sending to Stored Procedure:**

**STOCKLOCATION.lsQtyAlloc**

**STOCKLOCATION.lsQtyPicked**

* + - 1. 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

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.

**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.

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).

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.

[*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.

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.

Search JOBMISC for JOBMISC.RefPFix of ‘J’, JOBMISC.SubType of ‘A’, where JOBMISC.var\_code1 matches SOR.DETAILS.tlAnalysisCode.

Copy JOBMISC.AnalHed to JOBDET.JType.

* + - * 1. Search JOBHEAD for the record matching JOBDET.JobCode.
        2. If JOBHEAD.ChargeType is 3, copy JOBDET.Charge from JOBDET.Cost multiplied by JOBDET.Qty, rounding to 2 decimal places.
        3. 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.

[*DiscU3U: Calc\_StockPrice()*] Scan for Stock discounts (see Appendix 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).

[*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.

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.

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).

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.

[*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
       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.

Calculate the unit price in Stock currency. Real48

If DETAILS.tlCostPrice is zero, calculate and store it from the unit price multiplied by DETAILS.tlQtyMul.

Read STOCK.stQtyInStock, and get the Quantity Outstanding on the new line – this will actually be the delivered quantity.

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.

STOCK.stQtyInStock = STOCK.stQtyInStock – DETAILS.tlStockDeductQty.

**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).

Determine the FIFO Mode, based on STOCK.stValuationMethod and update the FIFO records.

* + - * 1. If Stock Locations are in use, update the Stock Location record.

STOCKLOCATION.lsQtyInStock = STOCKLOCATION.lsQtyInStock – DETAILS.tlStockDeductQty.

Save STOCKLOCATION.

**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:**

**SDN.DETAILS.VAT**

**SDN.DETAILS.IncNetValue**

**SDN.DETAILS.NetValue**

**SDN.DETAILS.VATCode**

* + - 1. Save the new line.
      2. 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.
    1. Deal with picked items on the original SOR.DETAILS line:
       1. If this is a back-to-back Purchase Order, we need to locate the matching Sales Order and pick the equivalent quantity on it.
          1. 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.
          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:

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.

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.

Round the Cost Price to EXCHQSS.NoCosDec decimal places.

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.

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.

Save the original Sales Order line.

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).

[*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).

If stValuationMethod is not ‘R’, check to see if this Stock item has Bin Numbers.

[DelvRunU: TSOPRunFrm.B2BBNos()]

* + - 1. Transfer any serial number records from the Order line to the Delivery line.
         1. If this Stock item doesn’t use serial numbers, check to see if Bin numbers need to be transferred instead.

Locate any Bin records against the Stock item, based on the Stock Folio, the Location, and whether or not the Sold flag is set

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.

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.

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).

Accumulate the number of items taken so far.

[*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. 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. 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. Create a new Match Payment record.
     1. Copy the details from the SDN/PDN.
     2. Save the EXCHQCHK record.
  10. Find any Note records against the original transaction and copy them to the delivery note.
  11. 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.

1. **Deliver Picked Orders** – Creates Delivery Notes for all Orders that have Picked lines on them, optionally consolidating all Orders that are against the same Trader into a single Delivery, moving the Orders into Order History if they are full delivered or written-off.
   1. Prepare custom hook points.
   2. Find all Sales/Purchase Order Transactions where the Hold Flag is one of 0, 3, 4, 5, 32, 35, 36, 37, 132, 133, 164, or 165.
   3. For each Transaction found (most of this is the same as Deliver This Order, but allows for multiple orders to be consolidated):
      1. If enabled, call a hook-point to allow a plug-in to reject the transaction.
      2. Scan the transaction lines for picked/written-off lines. Only continue with this transaction if at least one such line is found.
      3. For each Transaction Line:
         1. Check Order Payments to see if this Line requires a refund. Do not process the line if it does (leave the transaction outstanding, but deliver any other lines).
         2. If we don’t have a Transaction Header yet, create one now.
            1. Find the Head Office Trader record (using the acInvoiceTo field).
            2. If we are consolidating transactions onto a single delivery, search for an existing transaction header against the trader, SOP run number, and currency, and use this header.
            3. If we couldn’t find an existing transaction, or we are not consolidating, create a new transaction header.

Copy the Transaction details from the SOR to a new Transaction Header.

Change the Document Type to SDN/PDN.

Store the new Transaction Header.

Add an Audit Note.

* + - 1. Adjust the Stock to put the item back into Stock.
         1. Close any open Stock dialog.
         2. Update the Stock record with the new Allocated and Picked values.

STOCK.stQtyAllocated = STOCK.stQtyAllocated – DETAILS.tlQty.

STOCK.stQtyPicked = STOCK.stQtyPicked – (DETAILS.tlQtyPicked \* DETAILS.tlQtyMul).

* + - * 1. Update the FIFO valuation (these calculations make extensive use of Real48 values, although the fields in the database are Doubles).

Determine the FIFO Mode, based on STOCK.stValuationMethod and update the FIFO records.

* + - * 1. If Stock Locations are in use, update the Stock Location record with the new Allocated and Picked values.

STOCKLOCATION.lsQtyAlloc = STOCKLOCATION.lsQtyAlloc – DETAILS.Qty

STOCKLOCATION.lsQtyPicked = STOCKLOCATION.lsQtyPicked – DETAILS.QtyPicked \* DETAILS.QtyMul

Save STOCKLOCATION

* + - * 1. Set DETAILS. tlStockDeductQty to zero.
        2. Save STOCK
      1. 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).
      2. 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 1.5.4. Real48
      3. 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.
      4. 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 1.5.6.
      5. If 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. DETAILS.tlQtyDel = DETAILS.tlQtyDel + DETAILS.tlQtyPicked
         2. DETAILS.tlQtyWOff = DETAILS.tlQtyWOff + DETAILS.tlQtyPickedWO
         3. DETAILS.tlQtyPicked = 0
         4. DETAILS.tlQtyPickedWO = 0
      6. Accumulate the transaction amount delivered and/or written-off for this run, excluding VAT – calculate the new outstanding value using the same basis as 1.5.4 (but with the new QtyDel and QtyWOff values), and subtract the previous outstanding value originally calculated by 1.5.4.
      7. Accumulate the transaction amount delivered and/or written-off for this run, including VAT if Syss.IncludeVATInCommittedBalance is True, using the same basis as 1.5.5 (but with the new QtyDel and QtyWOff values), and subtract the previous outstanding value originally calculated by 1.5.5. Real48
      8. Accumulate the transaction amount written off for this run, excluding VAT – calculate the new outstanding value using the same basis as 1.5.6 (but with the new QtyWOff value), and subtract the previous outstanding value originally calculated by 1.5.6.
      9. Accumulate the transaction amount written off for this run, including VAT is Syss.IncludeVATInCommittedBalance is True, using the same basis as 1.5.7 (but with the new QtyWOff value), and subtract the previous outstanding value originally calculated by 1.5.7.
      10. Amend Stock allocation to hold the number of items that will still be allocated by the line after the current delivery and any write-off has been applied.
          1. Close any open Stock dialog
          2. Calculate the unit price in Stock currency. Real48
          3. If DETAILS.tlCostPrice is zero, calculate and store it from the unit price multiplied by DETAILS.tlQtyMul.
          4. 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 Syss.UseWIss4All is True) or STOCK.stQtyAllocWOR. Real48
          5. Calculate how many items will still be outstanding on the line after taking the current delivery and write-offs into account.
          6. If there is insufficient stock for the quantity outstanding on the line, adjust the DETAILS.tlStockDeductQty to be no more than the Free Stock.
          7. If there was sufficient stock for the quantity outstanding on the line, set tlStockDeductQty to the quantity outstanding. In conjunction with the previous rule, this means that tlStockDeductQty holds the number of items that can be delivered from the line after the current delivery has been completed, based on the number of undelivered items and the number of items available in Stock.
          8. STOCK.stQtyAllocated = STOCK.stQtyAllocated + DETAILS.tlStockDeductQty.
          9. Update the FIFO valuation (these calculations make extensive use of Real48 values, although the fields in the database are Doubles).

Determine the FIFO Mode, based on STOCK.stValuationMethod and update the FIFO records.

* + - * 1. If Stock Locations are in use, update the Stock Location record with the new Allocated and Picked values.

STOCKLOCATION.lsQtyAlloc = STOCKLOCATION.lsQtyAlloc + DETAILS.tlStockDeductQty.

STOCKLOCATION.lsQtyPicked = STOCKLOCATION.lsQtyPicked – DETAILS.QtyPicked \* DETAILS.QtyMul.

Save STOCKLOCATION.

* + - * 1. Save STOCK.
      1. 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.
      2. Save DETAILS.
      3. Create the DETAILS record for the new SDN/PDN.
         1. Copy the existing DETAILS record to a new record.
         2. Take the item out of Stock.

Close any open Stock dialog.

Calculate the unit price in Stock currency. Real48

If DETAILS.tlCostPrice is zero, calculate and store it from the unit price multiplied by DETAILS.tlQtyMul.

Read STOCK.stQtyInStock, and get the Quantity Outstanding on the new line – this will actually be the delivered quantity.

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.

STOCK.stQtyInStock = STOCK.stQtyInStock – DETAILS.tlStockDeductQty.

Update the FIFO valuation (these calculations make extensive use of Real48 values, although the fields in the database are Doubles).

Determine the FIFO Mode, based on STOCK.stValuationMethod and update the FIFO records.

If Stock Locations are in use, update the Stock Location record.

STOCKLOCATION.lsQtyInStock = STOCKLOCATION.lsQtyInStock – DETAILS.tlStockDeductQty.

Save STOCKLOCATION.

Save STOCK.

* + - * 1. Calculate the VAT for the line.

Call the customisation hook point 190001, 1, if enabled, to allow plug-ins to provide the VAT calculation.

If there was no hook point call, calculate the VAT normally.

* + - * 1. Save the new line.
        2. Update Job Actuals if this Transaction Line has a Job Code. Add a new Job Actuals record if one does not exist yet.

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.

If a record was not found, create a new one.

Copy the details from DETAILS to the new or existing record.

Save the Job Actuals record.

* + - 1. Transfer any serial number records from the Order line to the Delivery line.
         1. If this Stock item doesn’t use serial numbers, check to see if Bin numbers need to be transferred instead.
  1. 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. Calculate the totals for the new Delivery transaction
     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. Create a new Match Payment record.
     1. Copy the details from the SDN/PDN.
     2. Save the EXCHQCHK record.
  10. Find any Note records against the original transaction and copy them to the delivery note.
  11. 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 1.5.9 from thTotalOrderOS on the original transaction.
  14. Save the original transaction.
  15. Add an audit note for the transaction
  16. Delete any lines with a line number of -1 (BoM lines).
  17. For each Transaction Line:
      1. Make adjustments for Bill of Material items.
  18. Post to History the written-off amount calculated by 1.5.12.

1. **Invoice This Delivery** – creates an Invoice from a Delivery Note, deleting the original Delivery Note.
   1. Prepare custom hook-points.
   2. Display confirmation dialog, and exit if the user does not want to continue.
   3. Delete any hidden BoM lines on the Transaction (lines with a line number of -1).
   4. For each line on this transaction:
      1. If we don’t have a Transaction Header yet, create one now.
         1. Find the Head Office Trader record (using the acInvoiceTo field).
         2. Copy the Transaction details from the SDN to a new Transaction Header.
         3. Change the Document Type to SIN/PIN.
         4. Store the new Transaction Header.
         5. Add an Audit Note.
      2. Delete the line from the original transaction.
         1. Put the item back into Stock.
            1. Close any open Stock dialog
            2. STOCK.stQtyInStock = STOCK.stQtyInStock + DETAILS.tlDeductStockQty
            3. Update the FIFO valuation (these calculations make extensive use of Real48 values, although the fields in the database are Doubles).

Determine the FIFO Mode, based on STOCK.stValuationMethod and update the FIFO records.

* + - * 1. If Stock Locations are in use, update the Stock Location record.

STOCKLOCATION.lsQtyInStock = STOCKLOCATION.lsQtyInStock + DETAILS.tlDeductStockQty.

Save STOCKLOCATION.

* + - * 1. Set DETAILS. tlStockDeductQty to zero.
        2. Save STOCK.
      1. If the line has a Job Code:
         1. Delete the Job Actual records.
      2. If this is an SIN line (rather than PIN) delete the line.
    1. If this is the first line on the transaction, add a description line for the new SIN/PIN.
       1. Create an empty transaction line, and copy the basic details from the transaction header.
       2. Retrieve the original SOR reference from DETAILS.tlSOPLink (see 1.5.16.3), and add this to the contents of the Description field.
    2. Create the DETAILS record for the new SIN/PIN.
       1. Copy the details from the current (just deleted) line.
       2. Clear the tlQtyDel, tlQtyPicked, and tlQtyWOff fields.
       3. Adjust the stock.
          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.
       4. 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.
       5. Save the DETAILS record.
       6. Add Customer Stock Analysis, if required.
    3. Adjust the FIFO records to point to the new Invoice.
       1. For each record against the original Delivery Note:
          1. Copy DocRef, DocABSNo, and DocFolioK from the PIN details.
          2. Save the FIFO record.
    4. Transfer any Serial Number records from the PDN to the PIN.
       1. Find any Serial Number records and adjust them.
       2. If no Serial Number records were found, check for any Bin Location records instead.
    5. Deal with any negative FIFO (?).
    6. Set the Transaction Header run number to 0 (unposted).
    7. Set the Transaction Header CustSupp field to the correct character (‘S’ for Purchases, ‘C’ for Sales).
    8. Calculate CIS Tax.
    9. Calculate Transaction Header totals.
       1. Reset the total fields to zero.
       2. For each line on the Transaction:
          1. Accumulate the totals.
          2. If the ‘protect date’ custom buttons hook-point is not enabled, copy the transaction date to the line date.
          3. Update Job Actual records.
          4. Save the line.
    10. Update PPD totals for the Transaction.
    11. Update any BoM lines.
        1. Delete any existing hidden BoM lines.
        2. For each transaction line:
           1. If this is a BoM item, add the component lines.
    12. Set the Allocation Status character
    13. If the Delivery Note had an Order Payments status, update the Order Payments status on the Invoice.
    14. Save the Transaction Header.
    15. Post the balance to History (‘U’, ‘V’, and ‘W’ for the current Trader).
    16. Post the delivery balance to History (‘U’, ‘V’, and ‘W’ for the current Trader).
    17. Update the Matching Records to point to the Invoice.
    18. Move any Note records to the Invoice.
    19. Copy any Letters & Links records to the Invoice. (Does this mean that there are Letters & Links records left behind after the Delivery note has been deleted?).

1. **Invoice All Deliveries** – Creates invoices for all Delivery Notes, optionally consolidating the Delivery Notes into a single Invoice for each Trader, and deleting the original Delivery Notes.
   1. Prepare custom hook-points.
   2. Find all unposted Sales/Purchase Order Transactions where the Hold Flag is one of 0, 3, 4, 5, 32, 35, 36, 37, 132, 133, 164, or 165.
   3. For each Transaction (most of this is the same as Invoice This Delivery, but allows for multiple deliveries to be consolidated):
      1. If enabled, call a hook-point to allow a plug-in to reject the transaction.
      2. If the run is filtered by Stock Location, scan all the lines and only continue if at least one line was against the supplied Stock Location code.
      3. Determine the Order Run Number.
      4. Delete any hidden BoM lines on the Transaction (lines with a line number of -1).
      5. For each line on this transaction:
         1. If we don’t have a Transaction Header yet, create one now.
            1. Find the Head Office Trader record (using the acInvoiceTo field).
            2. If we are consolidating transactions onto a single delivery, search for an existing transaction header against the trader, SOP run number, and currency, and use this header. Do not consolidate transactions if there are Order Payments against the transaction.
            3. If a new Transaction Header is required:

Copy the Transaction details from the SDN to a new Transaction Header.

Change the Document Type to SIN/PIN.

Store the new Transaction Header.

Add an Audit Note.

* + - 1. Delete the line from the original transaction.
         1. Put the item back into Stock.

Close any open Stock dialog

STOCK.stQtyInStock = STOCK.stQtyInStock + DETAILS.tlDeductStockQty.

Update the FIFO valuation (these calculations make extensive use of Real48 values, although the fields in the database are Doubles).

Determine the FIFO Mode, based on STOCK.stValuationMethod and update the FIFO records.

If Stock Locations are in use, update the Stock Location record.

STOCKLOCATION.lsQtyInStock = STOCKLOCATION.lsQtyInStock + DETAILS.tlDeductStockQty.

Save STOCKLOCATION.

Set DETAILS. tlStockDeductQty to zero.

Save STOCK.

* + - * 1. If the line has a Job Code:

Delete the Job Actual records.

* + - * 1. If this is an SIN line (rather than PIN) delete the line.
      1. If this is the first line on the transaction, add a description line for the new SIN/PIN.
         1. Create an empty transaction line, and copy the basic details from the transaction header.
         2. Retrieve the original SOR reference from DETAILS.tlSOPLink (see 1.5.16.3), and add this to the contents of the Description field.
      2. Create the DETAILS record for the new SIN/PIN.
         1. Copy the details from the current (just deleted) line.
         2. Clear the tlQtyDel, tlQtyPicked, and tlQtyWOff fields.
         3. Adjust the stock.

Calculate the unit price in Stock currency. Real48

If DETAILS.tlCostPrice is zero, calculate and store it from the unit price multiplied by DETAILS.tlQtyMul.

* + - * 1. Calculate the VAT for the line.

Call the customisation hook point 190001, 1, if enabled, to allow plug-ins to provide the VAT calculation.

If there was no hook point call, calculate the VAT normally.

* + - * 1. Save the DETAILS record.
        2. Add Customer Stock Analysis, if required.
      1. Adjust the FIFO records to point to the new Invoice.
         1. For each record against the original Delivery Note:

Copy DocRef, DocABSNo, and DocFolioK from the PIN details.

Save the FIFO record.

* + - 1. Transfer any Serial Number records from the Delivery Note to the Invoice.
         1. Find any Serial Number records and adjust them.
         2. If no Serial Number records were found, check for any Bin Location records instead.
      2. Deal with any negative FIFO (?).
      3. Set the Transaction Header run number to 0 (unposted).
      4. Set the Transaction Header CustSupp field to the correct character (‘S’ for Purchases, ‘C’ for Sales).
      5. Calculate CIS Tax.
      6. Calculate Transaction Header totals.
         1. Reset the total fields to zero.
         2. For each line on the Transaction:

Accumulate the totals.

If the ‘protect date’ custom buttons hook-point is not enabled, copy the transaction date to the line date.

Update Job Actual records.

Save the line.

* + - 1. Update PPD totals for the Transaction.
      2. Update any BoM lines.
         1. Delete any existing hidden BoM lines.
         2. For each transaction line:

If this is a BoM item, add the component lines.

* + - 1. Set the Allocation Status character.
      2. If the Delivery Note had an Order Payments status, update the Order Payments status on the Invoice.
      3. Save the Transaction Header.
      4. Post the delivery balance to History (‘U’, ‘V’, and ‘W’ for the current Trader).
      5. Update the Matching Records to point to the Invoice.
      6. Move any Note records to the Invoice.
      7. Copy any Letters & Links records to the Invoice.

# 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')

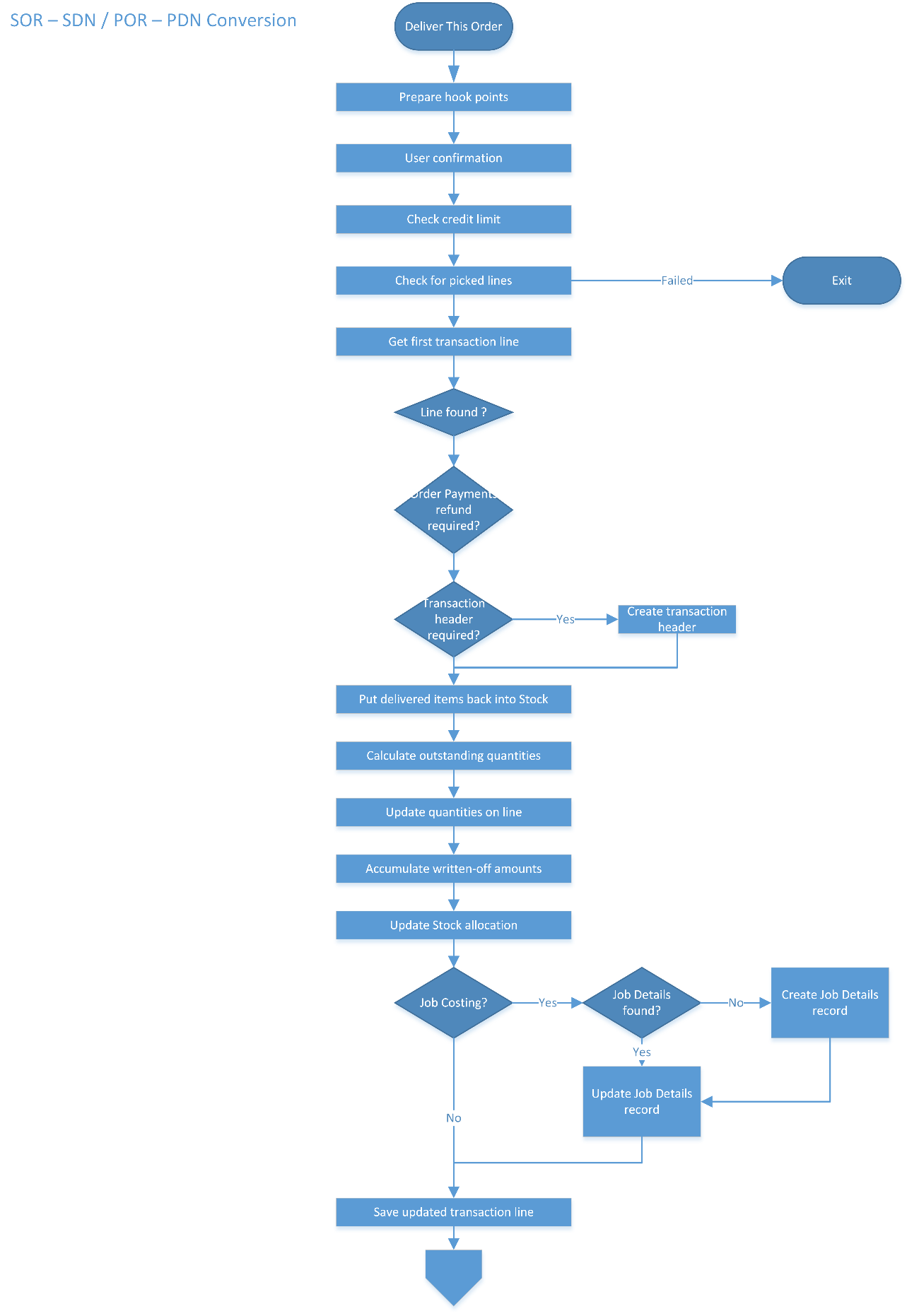
And ((UseDates = 0) Or ((UseDates = 1) And (StartDate <= '%s') And (EndDate >= '%s')))

And (Currency = 0)

) As NumConsolDiscounts

FROM temp\_StockTree

# Appendix 3 – Overview Flowchart



(Cont. from previous page)

