Cartonize

Documented By Tim Turner

Contents

[Functions 6](#_Toc415061905)

[Allocate\_Inv 6](#_Toc415061906)

[CartonizeChairs 6](#_Toc415061907)

[Cartonize\_Pieces 7](#_Toc415061908)

[Cartonize\_Order 10](#_Toc415061909)

[Consolidate 12](#_Toc415061910)

[Create\_CasePick 13](#_Toc415061911)

[Create\_MoveQ 16](#_Toc415061912)

[Create\_OB\_Ship\_Dtl 17](#_Toc415061913)

[Create\_OB\_Shipment 17](#_Toc415061914)

[Create\_picks 17](#_Toc415061915)

[Create\_PiecePick 17](#_Toc415061916)

[ExternalCartonize 18](#_Toc415061917)

[Generate\_Repl 19](#_Toc415061918)

[Get\_CarrM 19](#_Toc415061919)

[Get\_CartonMast 19](#_Toc415061920)

[GetRightSizeCarton 20](#_Toc415061921)

[Get\_PiecePick\_InvRID 20](#_Toc415061922)

[Sort\_For\_Bulk 20](#_Toc415061923)

[Split\_Product 20](#_Toc415061924)

[TotVolRem\_To\_Carton 21](#_Toc415061925)

[Update\_OBS\_Alloc\_Status 22](#_Toc415061926)

[Array Information 22](#_Toc415061927)

[InvAvail(X,J) 22](#_Toc415061928)

[ProdPcs(X,J) 22](#_Toc415061929)

[PieceVol(X,J) 23](#_Toc415061930)

[Cartons(x,y) 23](#_Toc415061931)

[Sub 23](#_Toc415061932)

[Assign\_Product\_To\_Largest\_Carton 23](#_Toc415061933)

[Create\_Audit\_Outbound 23](#_Toc415061934)

[Create\_OBS\_SQL 24](#_Toc415061935)

[Delete\_Error 24](#_Toc415061936)

[Display\_PieceVol 24](#_Toc415061937)

[DisplayProdPcs 24](#_Toc415061938)

[LOG\_ERROR 25](#_Toc415061939)

[Remove\_Allocation 25](#_Toc415061940)

[Sort\_Array 25](#_Toc415061941)

[Update\_OBS\_QC 26](#_Toc415061942)

[Global Variables 27](#_Toc415061943)

[AllocationStatus 27](#_Toc415061944)

[AllowMultThreads 27](#_Toc415061945)

[CarrierID 27](#_Toc415061946)

[CartonCnt 27](#_Toc415061947)

[CartonID 27](#_Toc415061948)

[CartonIDX 27](#_Toc415061949)

[Cartons(2000,4) 27](#_Toc415061950)

[cn 27](#_Toc415061951)

[CodAmt 27](#_Toc415061952)

[Connectstr 27](#_Toc415061953)

[Const LargestCartonIdx = 0 27](#_Toc415061954)

[Const PieceVolColCnt = 6 27](#_Toc415061955)

[cpwd 27](#_Toc415061956)

[CrLf 27](#_Toc415061957)

[CrLf = Chr(13) & Chr(10) 27](#_Toc415061958)

[crypto 27](#_Toc415061959)

[CurrInvAvailIdx 27](#_Toc415061960)

[DepAmt 27](#_Toc415061961)

[Done 27](#_Toc415061962)

[dsn 27](#_Toc415061963)

[DtTm 27](#_Toc415061964)

[ErrLoop 27](#_Toc415061965)

[Freight 27](#_Toc415061966)

[Fso 27](#_Toc415061967)

[FunctionName 27](#_Toc415061968)

[GoodsCost 27](#_Toc415061969)

[I 28](#_Toc415061970)

[InOrderId 28](#_Toc415061971)

[InsertStmt 28](#_Toc415061972)

[InsertStmtD 28](#_Toc415061973)

[InvAvail(2000,5) 28](#_Toc415061974)

[InvAvailCnt 28](#_Toc415061975)

[J 28](#_Toc415061976)

[LargestVolIdx 28](#_Toc415061977)

[LineNm 28](#_Toc415061978)

[LineVl 28](#_Toc415061979)

[logfile 28](#_Toc415061980)

[logfile\_str 28](#_Toc415061981)

[MaxCPL 28](#_Toc415061982)

[MaxThreadsAllowed 28](#_Toc415061983)

[MM\_systbl 28](#_Toc415061984)

[Nm 28](#_Toc415061985)

[OBS 28](#_Toc415061986)

[OBSCartonCnt, 28](#_Toc415061987)

[OBSCaseCnt 28](#_Toc415061988)

[OBSD 28](#_Toc415061989)

[OBSDCartonCnt 28](#_Toc415061990)

[OBSDCaseCnt 28](#_Toc415061991)

[Order\_Cancel\_Request 28](#_Toc415061992)

[orderby 28](#_Toc415061993)

[OrderCnt 28](#_Toc415061994)

[OrderErrCnt 28](#_Toc415061995)

[PaymentTerms 28](#_Toc415061996)

[PID 29](#_Toc415061997)

[PieceVol(5000,7) 29](#_Toc415061998)

[PieceVolCnt 29](#_Toc415061999)

[PPSortType 29](#_Toc415062000)

[PreviousProd 29](#_Toc415062001)

[ProcessXferOrder 29](#_Toc415062002)

[ProdPcCnt 29](#_Toc415062003)

[ProdPcs(5000,10) 29](#_Toc415062004)

[pwd 29](#_Toc415062005)

[Rs 29](#_Toc415062006)

[rs1 29](#_Toc415062007)

[rs2 29](#_Toc415062008)

[rsBRT 29](#_Toc415062009)

[sel\_clause 29](#_Toc415062010)

[sql 29](#_Toc415062011)

[sql1 29](#_Toc415062012)

[sqlBRT 29](#_Toc415062013)

[Status 29](#_Toc415062014)

[StrError 29](#_Toc415062015)

[StrText 29](#_Toc415062016)

[sysid 29](#_Toc415062017)

[sysname 29](#_Toc415062018)

[ThreadStarted 29](#_Toc415062019)

[TotVolRem 29](#_Toc415062020)

[TraceData 29](#_Toc415062021)

[TraceMsg, 29](#_Toc415062022)

[TracePoint 30](#_Toc415062023)

[trc 30](#_Toc415062024)

[uid 30](#_Toc415062025)

[UnitsPerCase 30](#_Toc415062026)

[Vl 30](#_Toc415062027)

[whr\_clause 30](#_Toc415062028)

[X 30](#_Toc415062029)

|  |  |
| --- | --- |
| Program | Cartonize |
| Variables | Global Variables |
| Main Location | 1-509 |
| What it does | Takes an order and separates it into cartons |

First thing it does after declaring variables (lines 7-35) is connect to the database, sets up trace log(85-87) check to see if more threads are needed(106-133) sets up a new threaded if needed(135-191). It then grabs the status from **Get\_CartonMast**.

The next section(191-474) is one giant if statement that only executes if Get\_CartonMast returned true( which it should usually do). It starts of setting order count to 0. It grabs ( something from the database) then loops through. Each iteration of the loop starts by construcing a query, (219-224) then adds onto it (225-242) it then checks to see if there is anything else to process(250-264) if there isn’t it exits the loop. It get the host order status(265-270) executes query( 274) error checking if the query fails exits if It fails( 276-284) checks something else and exits loop if it returns nothing( 286-300) if the query has data it sets the variables from the database.

# Functions

## Allocate\_Inv

|  |  |
| --- | --- |
| Passed Variables | ShipID  OrderNum  OrderLine  LocID  ProdID  Qty  AllocType |
| What it Does | Inserts information into the Inventory Allocation table |
| Returns | R- Error  C- Complete Through Picking, Ready to be shipped |
| Function Location | 2610- 2659(49) |
| Function Variables | Sql-the insert sql statement  Rs  IARid- set to NextRID  DtTm- set to U\_MakeTimeStamp(Now) |
| Function who call this one | Create\_picks, Create\_CasePick |
| Function This Function Calls | None |

First the Function Calls Trc and set FuncName to “Allocate\_Inv(2615), then it creates a timestamp(2616). It sets IARID to 0 then sets it to NextRID(“Inventory\_Allocation”)(Don’t know where this is) if IARID is 0(2621) then the program failed to get the next Inventory allocation RID so it will close the connection to the database Return a status of “R” then exit the function(2621-2630). If IARID is not 0 then we are good to proceed. We will construct a Insert Query where we will be inserting information into the INVENTORY\_ALLOCATION Table( 2632-2642). After we double check that the information was inserted into the database(2646-2654) and if an error was reported it will return a status of “R” then exit the function(2654-2655). Otherwise everything is ok and we will return a Status of “C” and end the function.

## CartonizeChairs

|  |  |
| --- | --- |
| Passed Variables | CurrentProd- current product being cartonized  CHairsOrdered-Number of Chairs ordered |
| What it does | Cartonizes chairs |
| Function location | 6426-6486(60) |
| Function variables | Cartontype- type of carton to put the chair in(C1,C2,C3,C4)  PiesesLeft- Pieces still to be Cartonized  PieceEachVol- Volume of one Piece |
| Returns | Nothing? |
| Functions who call this one | Cartonize\_Pieces |
| Functions this function calls | Sort\_Array, Display\_PieceVol |

First it checks how many chairs were ordered and assigns a box accordingly (6429-6537). It then checks to see that CartonID is 0(6439) if it is that means there are no other cartons in this order so it assigns the Chair to the correct box(6440-6445). If CartonID is not 0(6446) then it will next check if the Volume to cartonize is greater or equal to the number of pieces left to cartonize(IE, check to see if the previous box has room for another chair). If it is then it will put the current chair in the previous chairs box. If there is no room left in the current box(6454) then it will create a new box and put the piece in it(6455-6461). (LOOK UP THE REST LATER FIND A REFERENCE TO SEE WHAT INFORMATION IS CHANGING, PROB IN CARTONIZE PIECES)

## Cartonize\_Pieces

|  |  |
| --- | --- |
| Variables Passes | None |
| Function Location | 3091-3461(370) |
| What it does | Consolidate controls whether to put different products in the same box (if there is room) or not. |
| Variables | **Consolidate**-bool value to see if we need to Consolidate  **i**-for loop counter  **CurrentProd**-ID of a unassigned product line  **EachVol**- the volume of one item(total volume/Number of items)  **PiecesToConsolidate**-number of Items left to Consolidate into the current carton  **VolumetoConsolidate**- volume of items that need consolidation( **EachVo**l\***PiecesToConsolidate**)  **SortSeq**- how we want to sort the array(“ASC” or “DESC”)  **SortColumn**- where we want to start sorting from(1 or 56)  **PrevCarton**- the previous carton( Current Carton -1)  **PrevCartonVOlRem**-Volume remaining in the previous carton  **PrevCartID**- Carton ID for the Previous Carton  **Cube**- total volume remaining for product/pieces not yet cartonized  **chairFound**-bool value if a chair was ordered sets to true otherwise false  **chairsOrdered**- number of chairs ordered  **success** – bool Value of if ExternalCartonize failed or not.  **Sql**- Select all from Host Orders  **rs3**-Not Used? |
| Functions that call this one | **Cartonize\_Order,** |
| Functions this function calls | **ExternalCartonize, Split\_Product, GetRightSizeCarton, Assign\_Product\_To\_Largest\_Carton, Sort\_Array, Display\_PieceVol,**CartonizeChairs**,** TotVolRem\_To\_Carton**,** Assign\_Product\_To\_Largest\_Carton**,** |
| Returns | C- Complete  E- Error? |

First thing is it calls the system and sees if we need to Consolidate (3098-3106). Then it makes sure that at least one of every product in the order that needs to be cartonized can fit in the largest box (3111-3125). If not then we have a problem and we need bigger boxes. If this happens the function will return “E” and exit.

If everything in the order can fit in a box then it will check to see if there are any chairs in the order (3130-3141). If there are it will set **chairFound** to “True” and input the number of chairs in the order to **chairsOrdered**. It will then check if the global variable **TotVolRem** is larger than the Volume of the largest carton (3146).If it is, it will call **ExternalCartonize**, Passing in the global variable **InOrderId**, and return a success status (3150). It then checks if **ExternalCartonize** Succeeded or the order requires More than one box (If Success! =True) (3162). If one of those is true it will return “C”, and exit the function.

If **ExternalCartonize** failed or the order fits into one box the program will begin to loop while **TotVolRem** is greater than 0(3167). It will find the next unassigned product (3173-3180) by looping through **PieceVol (i, J)** looking at (**i**, 3) to see if it was assigned a **CartonIDX**,. Once it finds one that =”” (not assigned a carton) it will set CurrentProd to **i**. Next it will check that the current product is a chair (3186-3188). If it is (3189-3221) it will call **CartonizeChairs**(3192), and set **Consolidate** to “True”.

If consolidate is false but it is not a chair (3194) it will call **Split\_Product** Passing **CurrentProd**. If “False” is returned then a split is not necessary and it will assign to the best fit carton. the program will increment **CartonID** by 1 and call **GetRightSizeCarton** passing in the total volume for the current product(PieceVol(CurrentProd,1))(3205). **GetRightSizeCarton** will return (**Something)** and store it in **CartonIDX**.If **CartonIDX** is -1 then the smallest carton was too small, but we will use it anyway because this is the last of the pieces to cartonize. So we will subtract 1 from **CartonCnt** and store the result in **CartonIDX.** This will make sure we are using the smallest carton when we assign this piece to a carton (3211). We then assign the piece to a carton (3213-3217).

If **Consolidate** is “True” (3221), then we will try and consolidate the remaining pieces into open cartons. We start by checking if this is the first product in the order (3231) if that is true( **CurrentProd** >0) then we start by getting the previous carton(3233) and grabbing its ID (3234) and assigning it to **PrevCartID**. We then loop through **PieceVol(PrevCarton,5)** ( the cartonID of the previous carton) to find the first product line that was assigned the carton we need.

Once the carton is found then we need to get the Volume remaining in the carton(3248). Then we check to see if at least one item will fit in the box (3253,3254) first we check if the total volume remaining(**TotVolRem**) can fit in one box(3258) we do this by calling **TotVolRem\_To\_Carton** passing **TotVolRem.** If a -1 is returned then the remaining total volume won’t fit in one carton and we move on. If EachVOl wasn’t greater than the volume left in the previous carton, we will run **TotVolRem\_To\_Carton** again and try to fit at least one piece in the previous carton. If **TotVolRem\_To\_Carton** returned a -1 in the previous steps, then that means the remaining pieces wont fit in a existing carton, so we have to assign as many as we can to a new one. We begin this by checking to see if this is the first product in the order ( first product line) (currentProd=0). If that is true then we will call **Split\_Product** . If split product returns “false then we do not have to split the order into multiple boxes, in which case we will call Assign\_Product\_To\_Largest\_Carton and increment CartonID by 1.

If this is not the first product in the order (3298) then we will try and consolidate into the previous carton. We do the same as we did above. We use the current carton to get the previous carton (3304,3305). Once we have that we find what type of carton the previous carton is (3310-3314). Then we calculate the volume remaining in that carton (3319) and then we calculate how many pieces can fit in that carton (3323-3325). We calculate how many items can fit in a carton by first getting the volume of one shirt. We do this by taking the quantity of shirts and dividing it by the total volume of all the shirts and assigning that value to **EachVol** (3323) then we take the total volume of all the shirts we need to consolidate(**EachVol**) and divide it by the the total volume remaining in the current box. Doing this will give us the number of pieces that can fit in the remaining space, we store this value in **PiecesToConsolidate**(3324). Finally we take the pieces we can fit in the box(**PiecesToCOnsolidate** ) and multiply it by the volume of a piece(**EachVol)** and this will give the total volume that we need to consolidate, we store this in **VolumeToConsolidate**(3325).

If **PiecesToCOnsolidate** =0(3336) that means that no pieces of that type can fit in that box. So we will create a new carton for the product line. We do this by calling **Split\_Product**, if it returns false then we will call **Assign\_Product\_To\_Largest\_Carton** to create a new carton for this product and increment the **CartonID** by 1(3339).

If **PiecesTOConsolidate** does not = 0 the new can fit some shirts in a box (yay)(3341). We need to adjust the piece Vol array to reflect this. We begin by checking if we have a perfect fit (piecevol(currentProd,2)=pieces to consolidate). If this happens then we do not need to split the product at all and can just assign PieceVol(Previous Carton, 3 and 5) to the current carton and remove the extra volume from PrevVol(PreviousCarton,4).

If there will still be still more room after we consolidate into this box then we need to create another product line so we can fit more items in this box. We do this by inserting the information about the current product into the PieceVol array at the next free index, we insert all relivant information leaving index 3, 4, and 5 empty for now (3364-3370) then we increment **PieceVolCnt** to free up the next index in the PieceVol array. Next we need to adjust the volume of the carton we were previously consolidating too(PieceVol(PrevCarton,4)) next we update the current carton some more by inserting **VolumeToConsolidate** into (x,1), **PiecesTOConsolidate** then the previous cartonID into (x,3) and the previous CartonIDX into (X,5). After that we set up the paramters for sort array (3389-3395) and call **Sort\_Array** and after sort array is finished we call **Display\_PieceVol.**

Finally if Piece Vol(CurrentProd,2) < PiecesToConsolidate then we will assign the product to the previous carton. We first subtract from the total volume of the current carton, the remaining volume of the previous carton. then set CartonID and CartonIDX from the previous crates to the current ones.

All of the actions above were if the TotalVolumeRemaining would not fit into one box, but if it will fit into one box we do the following. First we increment CartonID by one(3421) then call **GetRightSizeCarton** and pass it **TotVolRem** which will return thetype of carton to put all the pieces in. if the smallest carton we have was too big (3425). Then we assign it the smallest carton. next we set the volume remaining to this carton(3435). Then assign all remaining product lines to this carton(3439-3442). Last thing we do before we go through the loop again is to calculate the total volume remaining for all unassigned pieces. If we are finished we will exit the loop return a “C” and exit the function.

## Cartonize\_Order

|  |  |
| --- | --- |
| Variables Passed | OrderID |
| Function Variables | AllPiecesBoxed  Found  Cubeperuom- cube volume of all the piece picks  Zone  Aisle  Bin  Level  COmpareVal1  CompareVal2  CompareIdx  SortColumn – where to start the sort  SortSeq- tell us if it is a ascending or descending sort  Rs  Tot\_Unit\_Price  Tot\_Units  Rs2  Sql2 |
| Function location | Line 2665- 3083(418) |
| What it does | Everything |
| Function that call this function | Main |
| Functions this function call | Log\_Error, Cartonize\_Pieces, Create\_OB\_Shipment, Sort\_Array,DisplayProdPcs, Display\_PieceVol Cartonize\_Order. |
| Returns | C- Complete  E- error  R |

After we declare the function variables (2669-2672) we do something (2674-2684). After that we set **Cartonize\_Order** to “C” and set the trace function name to “Cartonize Order(2692). After all that we select a product in the order (2697-2699). And then validate that the product exists (2702-2716) if the product doesn’t exist then we will set **Cartonize\_Order** to “E”. if **Cartonize\_Order** is “E” then we will print an error message to the debuglog and exit the function. If it is still “C” then we will move on.

After the SQL statement is check to make sure it holds data we the construct a new SQL statement that will grab all the products in the order (2737-2745). After we execute the query(2749) we check to make sure it returned no error(2752-2759) if it did we print a error message to the debuglog and return a status of “R” and exit the function.

Next we null out all the data in the **ProdPcs(X,J)** and **PieceVol(X,J)**array(2764-2770), and set **ProdPcCnt**, **CartonID**, **PieceVolCnt** to 0. Next we begin looping the order until we hit the end of file (2776). We begin the loop by setting **LineNm**, and **LineVl** to “”. Next we loop through the order and assign a Product\_ID to **ProdPcs(ProdPcCnt,1)**, a Quantity of items to **ProdPcs(ProdPcCnt,2)**, and a piece pick location to **ProdPcs(ProdPcCnt,9),** then it increments the **ProdPcCnt** by 1. If the product is a chair however(2785) then it will leave **ProdPcs(ProdPcCnt,9)** blank for now as we will come back to it later.We keep doing this until the table is empty.

Next we double check that the information was entered into the array by checking if **ProdPcCnt** is 0. If it is then no records were found and we exit the function (2798-2802). Next we close the connection if it is still open (2805-2807).

Next we will grab the number of eachs per case. We do this by first constructing a query leaving the where clause blank (for now) next week start a loop. First we grab the product ID from to **ProdPcs(**I**,0).** If it does not return anything we send an error to the debuglog and set **Cartonize\_Order** to “E. if no error is returned then we set grab the Units per Case from the database and store it in **UnitsPerCase.** Then set **ProdPcs(**I**,0)**to **UnitsPerCase.** After **UnitsPerCase** is assigned we will look to see how many full case picks this order needs for each product. We do this by dividing **ProdPcs(**I**,1 )(**Number of items in the order**)** by **UnitsPerCase.** This will give us the number of cases needed and we will store this in **ProdPcs(**I**,2).**  After that we check to see if the number of cases in this order is greater then MaxCPL. If so we will pass that information to LineVl and call **LOG\_ERROR** and sets **Cartonize\_Order** to “E”. if no error was returned then check how many piece picks we will need for this order. To do this we do a MOD of **ProdPcs(**I**,1)** by **UnitsPerCase**  Then we store that information in **ProdPcs(**I**,3)** . after that we close the connection to the database(2882-2884) and if Cartonize\_Order had been set to “E” at all during the last few steps it will print the information to the debug log and exit the function.

Next we set GoodsCost to 0. Then loop through all of the products in the order (2907) we are trying to get the total price volume and total price weight per product for the products. To do this we start by querying the product master tablebase for each item in the order(2909-2916). After we query the database and it returns a table, we check to make use the table has information in it, if it does (2918) we move on. We check if the value ”CUBEPERUOM” returned a non-zero value from the database(2922). If it did not we send an error to the debug log and and set Cartonize\_Order to “E”. if it did give us a non-zero value then we will pass that’s data into Cubeperuom, then multiply that by **ProdPcs(**I**,3)**(Number of piece picks), and store that value in **ProdPcs(**I**,5)**(2929-2936). After that we save the weight of a piece in **ProdPcs(**I**,6)** and the case weight in **ProdPcs(**I**,7)**.

Next we need to find the price of this order. We start by querying the “HOST\_ORDER\_DETAIL” table in the database and it grabs the units order and multiplies it by the unit price. After that it sets Tot\_Unit\_price and Tot\_Units to zero. Then it reads in the file and adds up the total unit price and the total number of units(2958-2962) after it is all added up we check to see that the total units is greater than 0. If it is then we insert into **ProdPcs(**I**,8)**(Units price per each) Tot\_Unit\_Price divided by Tot\_Units . if Tot\_Units was 0 then we insert a 0 into **ProdPcs(**I**,8)**(2963-2974) If the program did not find any information for Cube per unit of mesure then we call LOG\_ERROR and set Cartonize\_Order to “E” and exits the function.

Next we call DisplayProdPcs passing it ProdPcs(X,J). after that we set PieceVolCnt to 0( 3026). Next we look at all the products that need piece picks(3029-3040) after we find a product that has a piece pick we insert **ProdPcs(**I**,0)** into PieceVol(I, **0**), **ProdPcs(**I**,5)** into PieceVol(I, **1**),  **ProdPcs(**I**,3)** into PieceVol(I, **2**), and **ProdPcs(**I**,9)** into PieceVol(I, **6**). After PieceVol(X,J) is populated with the data we will sort PieceVol(X,J).(3045-3051). We first figure out if we are doing ascending or decending sort, and then we call Sort\_Array passing it PieceVol(X,J) SortColumn and SortSeq. After that we call Display\_PieceVol passing it PieceVol(X,J). After that we call Cartonize\_Pieces and that will return a status we will store in Status(3065). After the pieces have been cartonized, we sort PieceVol(X,J) again in an ascending sort(3071), and then display it by calling Display\_PieceVol(3071,3072). Last thing we do is call Create\_OB\_Shipment passing it the OrderID. That will return a status that we will pass to Cartonize\_Order and then exit the function.

## Consolidate

|  |  |
| --- | --- |
| Variables Passed | Volume |
| Function Variables | x- For counter |
| Function location | 4210-4226(16) |
| What it does | This routine attempts to consolidate the Total Remaining Volume to a carton that has room. If there is not enough room it returns a -1 |
| Function that call this function | TotVolRem\_To\_Carton, |
| Functions this function call | None |
| Returns | The index of a carton with valuable Volume or -1 |

The function starts of by starting a loop that will go through all the pieces in the order. It looks through PieceVol(X,4) for the first carton that still has volume that can fit this order. If it finds one it passes x to Consolidate and exits the function. If it does not find one, it passes a -1 to consolidate and ends the function

## Create\_CasePick

|  |  |
| --- | --- |
| Variables Passed | NewProd- bool value was a diff prodId was encountered in Outbound Shipment  SHipment  NumCasesReq  PcPkLocID-PIECE PICK LOCATION  ProdID-product id  ShipID  OrderNum  OrderLine  QtyToAlloc  CarrierID |
| Function Variables | sql  sql1  sql2  sql3  rs  MQParms  Status  I  InvAvailQty  ReplQty  TotAvailQty  WherePrimary- look in area C D E  WhereSecondary  WherePiecePick  OrderPrimary-order by locaction ID Receipt Date and Decsription  OrderSecondary  Dim Done  J  CurrLocID  CurrIdx  TempIdx  TempArr(1000,5)  InvCnt-inventory count  BulkPicking- bool value, allow bulk picking |
| Function location | 1187-1833(646) |
| What it does | NewProd is True or False and is used to deterimine whether or not a diff prodId was encountered in Outbound Shipment detail's Record set. If so then we need to get avail inv recs for the number of Cases needed for that product (NumCasesReq) and fill in the InvAvail array. The InvAvail array is used to alloc inv so that we can create MoveQueue recs. Storing the avail inv for a specific product in an array and then assigning the inv from the array is more efficient than going to the Db for each inv record needed. |
| Function that call this function | Create\_picks |
| Functions this function call | Allocate\_Inv, Sort\_For\_Bulk |
| Returns | -2: ERROR  0: not enough inventory  1: sucess |

The function starts off with checking if NewProd is true, if it is it will execute lines 1226- 1727. First we construct 3 queries. The first is looking for available inventory for this item. The second is to look for Replens not assigned to a pick/shipment. The third and final SQL statement gets any Replens that have been picked but not dropped off and not generated by Piece Pick Replen. After the queries are set we populate WherePrimary, OrderPrimary, WHereSecondary,OrderSecndary, And WherePiecePick all with strings(1290-1293). Next we set some rs values(1301-1308) and check if e are doing bulk picking, if we are then we don’t limit ra.MaxRecords. If we are not we set MaxRecords to NumCasesReq. Next we null out the InvAvail(X,J) and TempArr (1312-1317). After that we execute the first SQL statement in the preferred area, we select the preferred area by using WherePrimary and OrderPrimary As search options. After that we execute the statement(1324-1329). After it is executed we do an error check. If an error is found we close the connection, send the error to the debuglog return a -2 and exit the function. If no error is found we move on. Next we set InvCnt to 0 and read all the data in the table until the end of file(1346,1347). We insert information into InvAvail(X,J). InvID into InvAvail(InvAvailCnt,0), LocID into InvAvail(InvAvailCnt,0),MQRID into InvAvail(InvAvailCnt,0),and MQSTSTUS into InvAvail(InvAvailCnt,0)(1350-1360). After that we increment InvAvailCnt and InvCnt by 1(1367,1369).

After the loop is finished we check if NumCaseReq is greater than InvAvailCnt. If so then we will look for unpicked Replens in primary locations. We do this by executing sql2 looking in primary locations. Then do what we did above, we execute the statement. After it is executed we do an error check. If an error is found we close the connection, send the error to the debuglog return a -2 and exit the function. If no error is found we move on. Next we set InvCnt to 0 and read all the data in the table until the end of file(1346,1347). We insert information into InvAvail(X,J). InvID into InvAvail(InvAvailCnt,0), LocID into InvAvail(InvAvailCnt,0),MQRID into InvAvail(InvAvailCnt,0),and MQSTSTUS into InvAvail(InvAvailCnt,0)(1350-1360). After that we increment InvAvailCnt and InvCnt by 1.

If NumCaseReq is still greater than InvAvailCnt(1431) then we do the above paragraph again except this time use WhereSecondary and OrdersScondary instead of WherePrimary and Order Primary, after that if NumCaseReq is still greater than InvAvailCnt(1481) we will look for Replens in secondary locations by executing Sql2 with WhereSecondary and OrdersScondary.

If we still need cases we will next look in piece pick locations, ececuting SQL1 and using WHerePiecePick as the search option. Then like before error check, if InvCnt is not 0 populate InvAvail(X,J), and move on.

Next if we still need more cases we will use any Replens that have already been picked. To do this we just execute SQL3 then do the same thing as the rest (1581-1627). Next we print InvAvail(X,J) to the debuglog(1629-1640). After that we want to sort the locations in InvAvail(X,J) so a driver doesn’t have to drive all over the place. We start this process by setting CurrIdx and TempIdx to 0 and done to “False”. Then we begin a loop. We first thing we do is populate TempArr with the information in InvAvail(X,J). then we overwrite the information in InvAvail(X,J) with “Done”. Next comes the complicated part, we start at currentIDX+1 ( the next index in the array we haven’t set to “Done”) then we loop through InvAvail(X,J). we are looking for a a location that match the one in CurrLocId. If we find one we insert the information inInvAvail(X,J) into TempArr and then set all the values of InvAvail(X,J) at that index to “done, we do this for all location Ids that match.

After that we move on and see if there are more locations that need to be sorted or if we are done. To do this we first check to see if InvAvail(X,1) = “DONE” if it does then we move onto the next index, if they are all done then Done(the variable) is set to True otherwise it is set to false and we loop through the while loop again to sort more. Once we finished sorting the temp array, we insert that information into InvAvail(X,J). next if we are bulk picking then we need to put the bulk picks at the top of InvAvail(X,J). so we call Sort\_For\_Bulk passing it InvAvail(X,J) and InvAvailCnt. If we are not bulk picking we just move on. Next we print the information to the debug log and end the loop for if NewProd = true

Next we want to look in InvAvail(X,J) and allocate the inventory to it by creating a move queue. To do this we start by checking if we have the available inventory to fill this order, if we don’t then we return a 0 and exit the function. If we are getting this inventory from a piece pick location then we have to allocate that inventory, to do so we call Allocate\_Inv, which will return a status. If the status is not “c” then we will print an error return a -2 and exit the function.

If we are getting this inventory from a replenishment and it hasn’t been picked yet then we need to delete it. We will insert a pick instead. We also need to update Inv to indicate it is no longer replenishment. We start this by checking if anything in InvAvail(X,J) is in the move queue(InvAvail(I,2)>0) and if it has a status of “N” if we find one then we will delete the item from the move queue, then update the inventory table.after we are done with that we will set a bunch of MQParms, and call Create\_MoveQ. If it returned a false status we will print an error, return a -2 and exit the function. Otherwise we will increment CurrInvAvailIdx by 1, return a 1 ad end the function.

## Create\_MoveQ

|  |  |
| --- | --- |
| Variables Passed | MQ- a class, it hold lots of information |
| Function Variables | Sql-insert into movequeue statement  sql1- sql + more information  rs  DtTm- date and time  MQRID  CarrMast- object  Counter-number of times fail to insert counter |
| Function location | 2511-2601(90) |
| What it does | Creates a entry in the move queue table |
| Function that call this function | Create\_CasePick |
| Functions this function call | Get\_CarrM, |
| Returns | True or false |

After declaring all the variables, setting the trace function name, and creating the CarrMast object, we construct the SQL statement that will insert the information into the Move Queue table. After theSql statement is constructed we need to figure out a lot of the variables to insert into the statement. We start of by checking if MQ.tyPe is not a replen then we call Get\_CarrM and store what it returns in CarrMast. After that we check if CarrMast.Type= Parcel, if it does we set MQ.Location to “Conveyor”, otherwise we set it to “Shipstage”.

Next we set DtTm to right now, set counter to 0, and Create\_MoveQ to false. Then we begin a loop and keep going while Create\_MoveQ is false and Counter is less than 3. In this loop we start off by setting MQRID to 0 the next RID in Move\_Queue. then we check to see that MQRID is not 0. If it is we print a error, set Create\_MoveQ to false and exit the function. After that we construct SQL1. We do this by taking SQL and adding on more information. After that we execute the query, error checked if it worked, if it didn’t do what we print an error to the debug log, set Create\_MoveQ to false increment Counter by 1 and sleep for 20 seconds before trying again. If it did work we set Create\_MoveQ to true and end the function.

## Create\_OB\_Ship\_Dtl

|  |  |
| --- | --- |
| Variables Passed | OBS  LineNo  PVIdx- Product index  Uom- unit of mesurment |
| Function Variables | Textstr, i |
| Function location | 3748-3845(97) |
| What it does | Create a entry in the OB shipement Details table |
| Function that call this function | Create\_OB\_Shipment |
| Functions this function call |  |
| Returns | True or False |

Once we enter the function we check to see if Uom = CSE if it does we increment OBSDCaseCnt, otherwise we increment OBSDCartonCnt . next we assign Fields to Outbound Shipment Detail Dict, execute the query appending the fields we just added, error check if it worked. If it didn’t we report the error, return false, and exit the function. If it did work we return true and end the function.

## Create\_OB\_Shipment

|  |  |
| --- | --- |
| Variables Passed | OrderID |
| Function Variables | LineNo |
| Function location | 3467-3744(277) |
| What it does | Creates an entry in the outbound Shipment table |
| Function that call this function |  |
| Functions this function call | Create\_OB\_Ship\_Dtl, Create\_OBS\_SQL |
| Returns | R |

First the fuction grabs the information about the order from the Host\_Orders table(3479-3486). If it doesn’t find it, the program will send an error to the debug log, set Create\_OB\_Shipment to R and exit the function(3489-3494) the program then construsts an Insert Statement(3503-3516) then appends onto that statement OBS data and stores the SQL statement in InsertStmD. Next we set OBSCartonCnt and OBSDCartonCnt to 0.

Next we want to create one OB Shipment header for each carton. we start off looping through all the cartonIDs. Next we count the number of CartonID’s and put it into OBSCartonCount. After the loop is done we Print data into the debug log and move on. Next e call Create\_OBS\_SQL then execute the sql statement, and do an error check to see if it worked (3611-3616). If it didn’t we print the error to the debuglog, return a status of R and exit the function. If it did work we print that to the debuglog and close the connection. Next we want to create an OB Shipment Detail for each product in the carton. we do this by first looping through all the pieces once we find one that matches the ID we are currently on( remember we are still in a loop) then we increment lineNo, then call Create\_OB\_Ship\_Dtl , check what status it returned, as long as it wasn’t “E” we begin the loop again.

Last thing we want to do is create OutBound Shipment Headers for full cases. To do this we start by setting Line No to 1 OBSCAseCnt to 0 and OBSDCaseCnt to 0. Next we begin a for loop going through to ProdPcCnt. Then we begin another loop to create a record for every case of a particular product. To do this loop we start by incrementing CartonID by 1, and OBSCaseCnt by 1 and call Create\_OBS\_SQL. After we execute the Query we do an error check , and if there is an error, we return a “R” and exit the function. If Create\_OBS\_SQL was successful then we call Create\_OB\_Ship\_Dtl, and as long as that passes we will wither continue the loop or exit it. After we exit the loop, we print data to the debuglog and exitthe function.

## Create\_picks

|  |  |
| --- | --- |
| Variables Passed |  |
| Function Variables | OBSDCartonLineCnt  OBSDCaseLineCnt  OBSDCartonLineErrCnt  OBSDCaseLineErrCnt  OrderCnt  MQParms  Sql\_Inv  rs\_inv  rs3  ProdMast  LocMast  InvAvailQty- inventory avliable in the piece pick location  TotAvailQty- total quantity for a product between piece pick location and in the move queue  CurrOrderNum  PieceQtyToAllocate  ReplQty  ReplRemQty  ReplGenQty  ReplReqQty  LineStat  OrderStat  PPOrderStat  NewProdID  CurrProdID  ProdCaseCnt  Sqlc  Rsc  ProdStat |
| Function location | 515-1168(653) |
| What it does |  |
| Function that call this function | Update\_OBS\_Alloc\_Status, Get\_PrdM |
| Functions this function call |  |
| Returns | R- error |

After all the variables are declared, the trace log is set, the objects are created, and the connection is set up (517-531) we start off with Piece picking. We first set PieceQTYToAllocate, OrderCnt, OBSDCartonLineCNt, and OBSDCartonLineErrCnt all to 0. Next we prepare our Select statement and insert it into sql. Next we create a new object then use that object to execute the SQL statement. If the statement returned an error, then we return an “R” and exit the function. If not we move on. Ext we start piece pick creation.

We start by setting PPOrderStat and OrderStat to “A” and OrderCnt to 0. Then begin a while loop to read in the file (592). We start the while loop by checking if we are working on the next order, if we are we set it’s status fields. We do this by setting the PiecesQtyToAllocate to the number of ordered units and the LineStat to “A”. next we call Update\_OBS\_Alloc\_Status. If the status returned is not “C” then we we return the status and exit the function. Otherwise we set the CurrOrderNum.

Next we need to check if the product was not found. First we call Get\_PrdM to set ProdMast. After ProdMast is set we check to see if no information was inputed. We do this by checking if ProdMast.Product\_ID is “”. If it is we call the error log, return a status of “E” and exit the function. Otherwise we move on(623-638).

Next we want to check the product has a valid piece pick location. We do this like the step above. Check if the field is set to “”, if it is call the error log, return status “E” and exit the function, otherswise we move on.

Next we call need to Get\_Loc to get the piece pick location. Then check to see if it exists. If it doesn’t’ we print and return an error and exit the function. After we have the location we then get the available inventory at piecepick locations. We do this by calling Get\_Avail\_Inv. We next get any Replenishments in the move queue by calling by calling Get\_ReplQty. Then we take those two values, add them together and store them in TotAvailQty.

After we have the Total available quantity we want to create the initial entry in the Inventory\_Allocation Table. To do this we call Allocate\_Inv. Then we check to make sure it successfully entered the data. If it didn’t we return a status of “R” and exit the function. After the entry is inserted into the inventory allocate table we will begin to do most of the heavy lifting.

We start by checking if there is not enough total available to fill this order, if there isn’t then we will try and replen. We first check if the total available quantity is greater than 0. If it is we will call Create\_PiecePick to return the quantity that was not allocated. After that we check to see if it returned an error, if it did we return an “R” and exit the function, otherwise we will start to generate replenishments. The start that process we call Generate\_Repl, if it returns an error we return “R” and exit the function. Then we check if a replenishment for the full amount was created. If it was not we set the lineStat to “S”.

If no inventory was available anywhere then we sett the line status to “s”(798-805). Otherwise we have enough piece products available to meet this order. So we will call Create\_PiecePick, and if it returned an error we return a ‘R’ and exit the function. Now the order Line Demans should have been met, but if not then we will try and generate a replen to generate additional picks if possible. If it returned an error we exit, and if it couldn’t create a replen we set the LineStat to “S”

Next we check if LineStat was set to S at anypoint. If it was we call the error log, and set PPOrederStat to “S”. Next we update the outbound shipment detail status table with the new information. After we end the while loop (if we are done and have no more orders to fill). After we exit the loop we check if this was the last order in the shipment we do this by calling Update\_OBS\_Alloc\_Status and if it returns a “C” then we return the same and exit the function. After all of that piece pick allowcation is complete, so we can now start on case picking.

We start case picking by setting OBSDCaseLineCnt and OBSDCaseLineErrCnt both to 0. Next we create the sql statement to get all the case orders in the order. We execute the sql statement, check for an eror, and then see if there were cases to pick for this order, if there were we begin a while loop to read until EOF.

In the loop we start by setting LineStat to “A” , incrementing OrderCnt by 1 and NewProdID to false. We check is OBSDCaseLineCnt is greater than 0. If it is then check if CurrProdID is not the one that was gotten from the database. If that is true( cuttentProdID!= the productID from the database). Then we set NewProdID to true, set ProdStat to 1and call Get\_PrdM to set ProdMast. Then we check that PordMast is not empty, if it’s not we will store what’s in ProdMast in CurrProdID. If CurrProdID did equal the product ID from the database then we just call Get\_PrdM , set CUrrProdID, set NewProdID to true and ProdStat to 1.

Next we check to make sure the product was found. If CurrProdID =”” then we increment OBSDCartonErrCNt by one print the error to the debug log, return a “E” and exit the function. As long as we have no error we just move on and count the number of cases needed for this product. First thing we do is construct a query to count the number of cases of this product we need for this order. We execute the query and error check. If we find an error we return a status of “R” and exit the function. If not error was found then we will create a case pick.

To create a case pick we check if ProdStat is one( it will be 1 if there is enough product, or this is the first time we are checking). We then call Create\_CasePick and insert what it returns into Status. After that we check what status was returned. If status is -1 then No product was found at all so we call LOG\_ERROR and set ProdStat to 0 and OrderStat to “s”. If Status was 0 then we THEN WE SET OrderStat to “S” call LOG\_ERROR and set ProdStat to 0. If status was -2 then there was a problem with connecting to the database so we set Create\_Picks to “R” and exit the function.

Next we call Update\_OBS\_Alloc\_Status and set Status to what it returned. If the sataus returned was not “C” then we return status increment OBSDCartonLineErrCnt by 1 and exit the function. Otherwise we increment OBSDCaseLineCnt by 1. Next we update outbound\_Shipment\_Detail status for all orders in the shipment of this product that we just created a move queue for. After all of that we end the loop.

With only a few things left to do we pick up after we are finished the loop by checkin if the OrderStat is “A” if it is we then check if OBSDCASELINECNT is greater than 0 if that is true we will call Update\_OBS\_Alloc\_Status and store the returned value in Status. If Status was not “C” then we return Status and exit the function. At this pointif the order was short we need to look at allocating any non-std case qty inventory. We only need to do this is piece pick allocation was successful.

To begin this process we check if No Outbound Shipment Records were created for the order we do this by checking if OBSDCartonLineCnt and OBSDCaseLineCnt both equal 0. If that is true then weprint the error and return “E”. if OrderStat was “S” and PPorderStat was”A” then we call InvLastReport and store what it returns in Status, after that we check if Status is “E”. if status was “E” print a warning to the log and set Create\_Picks to the Status. If PPORderStat was set to “S” then we need to set Create\_Picks to”E”. last thing we do is check if Create\_Picks is not “R” and is not “E” if both of those are true we set Create\_Picks to “C” and exit the function.

## Create\_PiecePick

|  |  |
| --- | --- |
| Variables Passed | LocID  ProdID  Zone  ShipID  OrderNum  LineNo  QtyToAlloc  CarrierID |
| Function Variables | Sql  Sql1  Sql2  Rs  Rs1  MQParams  Status  I  AvailQty  RemQty  ReqQty |
| Function location | 1828-2063 |
| What it does | Returns t |
| Function that call this function | InvLastReport  Create\_picks |
| Functions this function call |  |
| Returns |  |

After we declare the function variables, set AvailQty, REmQty to 0, and reqQTY to QtyToAlloc(1839-1841). Next we create the recordset and the MQParms object(1846-1872). Now that everything is declared we construct a SQL statement to get available inventory in piece pick locations then execute it (1878-1896). Next we loop through all piece pick inventory to satisfy the requirement (loop 1902-1966). Once in the loop we first check if the Piece Pick QTY for this item is above 0. If it is we will construct a query to see if any of this product is already allocated, we do this by checking the move\_queue(1909,1910). Once we execute and find the quantity that has already been allocated, we will subtract that from the Piece pick quantity in that spot, and store it in AvailQty. After we have the avliable quantity we check to see if it is greater than 0. If it is we will allocate this piece for picking(1930). If what is needed is greater than what is avliable then we will allocate the entire amount, otherwise we just allocate normally. Once the MQParms are set we will call Create\_MoveQ . If Create\_MoveQ returned false then we had an error and we exit the and return a -1. If Create\_MoveQ did not return an error we will loop again if we have more ReqQty or exit the loop if we are done .

Next we need to see if piece pick was not fully allocated, and if it wasn’t we need to check for any replenishments and then use those to fill the order. The first thing we do to do this is construct a SQL statement to get any Replens heading to a piece pick location(1973-1990) then we loop through the results and if any of those Replens are heading to the piece pick location our product is at. If we find any we will allocate all available to the order(2021-2038). Then like before we will call Create\_MoveQ . If Create\_MoveQ returned false then we had an error and we exit the and return a -1. If Create\_MoveQ did not return an error we will loop again if we have more ReqQty or exit the loop if we are done . once finished we will return ReqQty.

## ExternalCartonize

|  |  |
| --- | --- |
| Variables Passed | Shipid |
| Function Variables | Filesys-  Xml-  Success-  n-  curr-  volcartonized- set to zero  tvr-  path-path of the XML file  filename- Unique filename for the XML file  Return- returned status from PPcartonized  Cmd- command line delcare  Cmdline- what to write to CMD  Ctnindex- set to 0  Temppiecevol(5000,7)- |
| Function location | 6287-6424(137) |
| What it does | Calls PPcartonize app then returns with XML data |
| Function that call this function |  |
| Functions this function call |  |
| Returns |  |

First it sets volcartonized to 0 and tvr to Round(totVolRem,3). If it finds an error it sets ExternalCartonize to false. Ctnindex is set to 0 and the temp array is nulled out (6304-6308). Then it creates a file path for the XML document(6310-6313) then it finds PPCartonize.exe and executes it(6317-6322) if PPCartonized returned a 1(6324) then it failed for some reason and the function will exit(6324-6327). Otherwise it will have exited correctly, and start pulling the data from the xml file into the temp array(6332-6368). It will then round up the volume of volcartonized to the 3rd decimal place(6370). It then deletes the XML file from the system(6373-6378) it will then check to see if the entire array was moved successfully by checking if Volcartonized and tvr are the same if they are then it will null out PieceVol and insert all the information in TempPieceVol into it, otherwise it will exit the function then it will execute a SQL statement and exit the function.

## Generate\_Repl

|  |  |
| --- | --- |
| Variables Passed | ShipID  OrderNum  OrderLine  ToLocID  ToZone  ProdID  ByVal  ReplReQty |
| Function Variables | Sql  Sql1  Sql2  Sqlval  Rs  Rs2  DtTm  FromLocID  LocMast  FromZone  OrigRedQty  MQParms  CasesToReplen  SatisfiedPick |
| Function location | 2072-2334 |
| What it does | Looks for available inventory to replen from |
| Function that call this function |  |
| Functions this function call |  |
| Returns |  |

After variables are declared we set Cases ToReplen to 1 and begin constructing 3 sql statements. The first is getting the location of all inventory where quantity is 0. The second updates the inventory table. the third gets the capacity of a location. After all the sql statements are declared we start by executing the first. This will get us a list of replen candidates from inventory. After the query is executed, if a replen candidate is found then we get the location capacity. To do this we execute sql2, and if it returns no errors we move on and set SatisfiedPick to false. Now that w ehave a list of replenishment case record canidates, we will use the first to satisfy the pick demand and the remaining to fill the location to its capacity.

To do this we start by looping through the information we retrieved from the first sql statement. Then we call Get\_Loc to set LocMast, if it returned noting we set FromZone to ‘’ otherwise we set it to the pickzone that was returned, and res LocMast back to nothing. Next we check if the replen for the pick was already created(the first time throught the loop) if it was then we just need to create pepln movequeues to fill to capacity. Next we will calculate the difference between the replen required quanity and the inventory quantity. If it is a negative number then the will subtrace 1 from cases to replen. Next we set the variables in the MQParms object. If the replen is the satisfy an order we assign shipment info to it, otherwise we set that info to null. The we call Create\_MoveQ and set status to what it returned, if It returned false we return a -1 and exit the function.

Next we update inventory to denote as used for replenismennt. Last if satisfiedPick is false the we set it to true, set MQParams object variables and call Create\_MoveQ. Like befoe if it retuned false we return a -1 and exit the function. Otherwise we either begin the loop again or exit the loop. Once we exit the loop we set rs and MQParms to nothing. If replReqQty is grater than or equal to 0 we set generate\_repl to 0. Otherwise we set it to ReplRegQty and the function.

## Get\_Avail\_Inv

|  |  |
| --- | --- |
| Variables Passed | LocID – location id of product  ProdID- product id of product |
| Function Variables | Sql-sql statement to get a the inventory of a certin product and a certin location  Rs- record set for sql statement  InvQty- stores inventory quantity that is returned from the database  InvAllocQty- all inventory that has been allocated for that product |
| Function location | 4327-4371 |
| What it does | Retrieve available inventory( sum of inventory- sum of inventory allocated) |
| Function that call this function | Create\_picks |
| Functions this function call |  |
| Returns | InvQty-InvAllocQty |

After the variables are declared we construct a sql statement to get all of the inventory of a product in a location( both were passed into the finction).after we execute if something was returned we set the quantity returned to InvQty, ifnothing was returned we set InvQty to 0. After we have InvQty we construct another sql statement. This time to get all the inventory that has been allocated for the prodect at the location. We will store this value returned from the database in InvAllocQty, if nothing was returned we will set it to 0. Last thing we do is return InvQty-InvAllocQty and end the function.

## Get\_CarrM

|  |  |
| --- | --- |
| Variables Passed | CarrID |
| Function Variables | Sql- sql statement to get information from carrier table  Rs-record set for what the sql statement returned  I – for loop counter |
| Function location | 4306-4322 |
| What it does | Populates the carrier master buffer |
| Function that call this function | Create\_MoveQ |
| Functions this function call | none |
| Returns | Information from carriers table |

This is a very simple function. We execute a sql statement to get all from the carriers table where carrier id= the id passed to the function. Then we loop through all the returned data, and return what was returned from the database.

## Get\_Loc

|  |  |
| --- | --- |
| Variables Passed | LocID |
| Function Variables | Sql  Rs  i |
| Function location | 4284-4302 |
| What it does | Populates the product master buffer |
| Function that call this function | Generate\_Repl  Create\_picks  Revise\_OBS |
| Functions this function call |  |
| Returns | Information from locations table |

This is a very simple function. We execute a sql statement to get all from the Locations table where location id= the id passed to the function. Then we loop through all the returned data, and return what was returned from the database.

## Get\_PrdM

|  |  |
| --- | --- |
| Variables Passed | ProdID |
| Function Variables | Sql  Rs  I  Caseum-unused  Stdcaseqty-unused |
| Function location | 4262-4279 |
| What it does | Populate product master buffer |
| Function that call this function | Create\_picks |
| Functions this function call | none |
| Returns |  |

This is a very simple function. We execute a sql statement to get all from the Locations table where product id= the id passed to the function. Then we loop through all the returned data, and return what was returned from the database.

## Get\_ReplQty

|  |  |
| --- | --- |
| Variables Passed | LocID  ProdID |
| Function Variables | Sql  Rs  I  replQty |
| Function location | 4375- 4427 |
| What it does | Retrieve replenishment qty from move queue |
| Function that call this function |  |
| Functions this function call | none |
| Returns |  |

This function starts off with a sql statement getting the total quantity of replenishments and pass it to Get\_REplQty. After that we construct a new sql statement to get the total quantity of the replenishments that are allocated by case picks. Then we subtact that from what was already in get\_ReplQty and return that

## Get\_CartonMast

|  |  |
| --- | --- |
| Variables Passed | None |
| Function Variables | none |
| Function location | 4432-4483 |
| What it does | Sets the Different types of boxes |
| Function that call this function |  |
| Functions this function call |  |
| Returns | Status(bool) |

First the function queries the Database are grabs all the different types of cartons we use. Then it stores that information in the cartons array by looping through with a while loop until it hits the end of file. It stores the following information in the Cartons array:

After it loops through it increments Carton Count by one so that way the program know how many different types of cartons we have. If nothing was put into the array(4725-4728) then the program will push a error to the debug log (4727) and set Get\_CartonMast to false. If everything worked like it was meant to then carton count will decrement one. After that the program will print the contents of Cartons to the debug log it will set Get\_CartonMast to true and exit the function.

## GetRightSizeCarton

|  |  |
| --- | --- |
| Variables Passed | Volume |
| Function Variables | I – set to 0, used to keep the while loop going  Maxfit- set to true  CartonIdx- |
| Function location | 4048-4089(41) |
| What it does | Figures out the carton that will best fit |
| Function that call this function | Main |
| Functions this function call |  |
| Returns |  |

The function will loop through all of the cartons available, looking at each one and comparing the volume needed to the volume of the boxes available. If the volume is too big it will increment I can move onto the next box( they are ordering in decending order) at the end there is a check to see if the smallest carton was too big, if it was then it will set it to the smallest carton.

## Get\_PiecePick\_InvRID

|  |  |
| --- | --- |
| Variables Passed | LocID  ProdID |
| Function Variables | Sql  Rs |
| Function location | 4416-4447(31) |
| What it does | Returns a piece pick rid |
| Function that call this function | none |
| Functions this function call | None |
| Returns | \_RID\_ |

This is a very simple function. We execute a sql statement to get all from the Inventory table where product id= the id passed to the function and Location ID = the locID passed in. Then we loop through all the returned data, and return what was returned from the database.

## Revise\_OBS

|  |  |
| --- | --- |
| Variables Passed | ShipID  OrderNum  ProdID  caseQty  InvRid  Location  CarrirerID  AllocationType |
| Function Variables | MQParms  Status  Sql  Sql1  Rs  Rs1  orderNumber  LocMast  AllowCopl  Exceed\_ship\_By  Substitution\_override  Special\_Handling |
| Function location | 5361-5762 |
| What it does | Modifies outbound shipments and outbound shipment detail and insert into move queue when a non-std case qty is used to satify a certain order’s product demand that was short. When this happens we need to adjust the qty on the outboundshiopment details recond to the non-std case qty and delete the old record |
| Function that call this function | InvLastResort |
| Functions this function call | Create\_MoveQ |
| Returns | True or false |

After we declare the variables and set the debug log we construct a sql statement to get the first shorted outbound shipment details entry for this product and use it to allocate the non-std case quantity to. To do all of that, if an OBS record is found then we update the first record found. If the allocationType is “COmpleate” we also delete the remaining shorted OBS/OBSD for this product. If the OBS record is not found it means that we need to create additional OBS/OBSD record to satisfy the requirement for this product.

To do that we execute the sql statement and if something is returned. If something is returned we will update rs with the ordered\_units, allocation\_status and the order number. After rs is updated we need to update outboundshipments. We construct a query and if it returned with something then we set allocation status to “A”.

If nothing was returned the first time then we create a new OBS/OBSD rrecord. To do this we first close rs and construct a new sql statement to get the current max order number, then we increment that number by 1 and store it in OrderNumber. Then we insert a new entry into outbound shipments for the new order number created. Then we insert a new entry into outbound shipment details.

After the information has been updated or added, we check if all short products have been allocated then we delete any remaining set for this product that are still short. To do this we we check if allocation=”complete” if it does we will construct a query to delete the entry from outbound shipments and outbound shipment details. The last thing we do is create a move queue record for this shorted OBSD record. We insert variables into MQParms object, then we call Create\_MoveQ. Last thing we do is check what Create\_MoveQ returned. If it was flse we exit the function with a “R” otherwise we exit with a”C”.

## Sort\_For\_Bulk

|  |  |
| --- | --- |
| Variables Passed | InvArray  IntCnt |
| Function Variables | InvDict  Location\_Zone  BulkZone  Cn  Sql  Rs  Bulk\_zone\_array  D  I  J  NewArrayCOunt  Invrid  Location  Mqrid  Mqstatus  Unknown  Temp  B |
| Function location | 1843-1948(105) |
| What it does | Use the inventory array from Create\_casePick and sort it so that the bulk pick zones come first( this is only for PA) |
| Function that call this function | Create\_CasePick |
| Functions this function call | none |
| Returns |  |

After we declare variables, and set the objects we selects location\_id and picking zone from the locations table. We add all of these to the location\_zone object. Next we load the bulk zones from SysD and store them in Bulk\_zones. Next we take the InvArray and put it’s contents into the InvDict object. After that we null out InvArray and set NewArrayCOunt to 0. Then we repopulate InvArray, first by entering in all the items in the Bulk Picking zones first, then we add the rest of the inventory.

## InvLastResort

|  |  |
| --- | --- |
| Variables Passed |  |
| Function Variables |  |
| Function location | 5137-5361 |
| What it does |  |
| Function that call this function | Create\_picks |
| Functions this function call |  |
| Returns |  |

## Split\_Product

|  |  |
| --- | --- |
| Variables Passed | CurrProd |
| Function Variables | EachVOl- Volume of 1 shirt  MaxEachCarton-max volume of the carton for a type of shirt  FullCartons-  MAxEachPerCartonVol-  SysTbl-  SortColumn-  SOrtSeq-  CIdx- |
| Function location | 4104-4196(95) |
| What it does | This routine is invoked when the product with the largest remaining volume that has not yet been cartonized is too large to fit into the largest carton (by volume). This routine takes the total product Pieces and cartonizes them into the largest box until the remainder is less than the largest carton LCB (Min capacity). |
| Function that call this function |  |
| Functions this function call |  |
| Returns |  |

First the function will calculate the volume for 1 shirt and store it in EachVol(4112). Then it will calculate the max Volume that can fit in that box for that specific shirt(4117). It will then calculate the max number of items of shirts that can fit in that box(4122). After the function calculates the number of full cartons it can create, ignoring the remainder(4129) if there is no remainder(4130) there will be one less carton because” we will keep a carton at the currprod line”. It will then split the product up into new cartons for as many fullCartons were created(4137) for each new full carton it will increment the carton ID by 1, insert the nessisary information into the PieceVol array(4144-4153) then increment PieceVolCount by 1(4155). It then adjusts the original record to what the leftover quantity will be(4265,4166). Then if it created any full cartons it will set up to call sort array(4177-4183) then sort the array(4186) and display the new information(4188). It will then set Split\_Product to true. If it did not need to make any full cartons (Fullcartons==0) then it will set Split\_Product to false and exit the function.

The program then deletes any previous error, and sleeps for 200(unit?) it then runs Cartonize\_Order

## TotVolRem\_To\_Carton

|  |  |
| --- | --- |
| Variables Passed | Volume |
| Function Variables | MaxFit- bool value set to false |
| Function location | 3919-3987(68) |
| What it does | Attempts to find a carton to hold the remaining piece volume |
| Function that call this function | Function Variables |
| Functions this function call |  |
| Returns |  |

This function does basically the same thing as GetRightSizeCarton except that it is not assigning anything to an array. When the function exits it is just passing an index in the Carton array( the Correct size box). The function starts by setting i to 0 and maxfit to false. When runs a while loop for while Volume<=Max Vol Capacity( go until I find a carton where Volume is Greater than the cartons volume). If I finds the carton it will set TotvolRem\_To\_Carton to I and exit the function. If it doesn’t fid one( the smallest box was too big) it will try and consolidate into an existing box. It will do this by running Consolidation function. If Consolidation returns -1( false) then it will put the items in the smallest box, otherwise it will set TotVolRem to whatever Consolidate returned.,

## Update\_OBS\_Alloc\_Status

|  |  |
| --- | --- |
| Variables Passed |  |
| Function Variables |  |
| Function location | 4357-4412(55) |
| What it does |  |
| Function that call this function |  |
| Functions this function call |  |
| Returns |  |

# Array Information

## InvAvail(X,J)

|  |  |
| --- | --- |
| Array Contains | X is int used as a counter, j contains the information below |
| ProdPcs(x,0) | Inventory ID |
| ProdPcs(x,1) | Location ID |
| ProdPcs(x,2) | Move Queue ID |
| ProdPcs(x,3) | Move Queue status |
| ProdPcs(x,4) |  |

## ProdPcs(X,J)

|  |  |
| --- | --- |
| Array Contains | X is int used as a counter, j contains the information below |
| ProdPcs(x,0) | Product ID |
| ProdPcs(x,1) | Summarized Piece (each) quantity |
| ProdPcs(x,2) | Number of full case picks |
| ProdPcs(x,3) | Number of Piece Picks |
| ProdPcs(x,4) | Eaches per case |
| ProdPcs(x,5) | Total volume for left over pieces |
| ProdPcs(x,6) | Weight of a piece of product |
| ProdPcs(x,7) | Weight of a case |
| ProdPcs(x,8) | Unit\_Price\_each |
| ProdPcs(x,9) | Piece Pick Location |

## PieceVol(X,J)

|  |  |
| --- | --- |
| Array Contains | x- counter for items j- contain the following below |
| PieceVol(x,0) | ProductID |
| PieceVol(x,1) | Total Volume |
| PieceVol(x,2) | Pieces to cartonize |
| PieceVol(x,3) | Carton IDX |
| PieceVol(x,4) | Cartons(x,2) volume left in the box |
| PieceVol(x,5) | Carton ID |
| Piece Vol(x,6) | PiecePick Location |

## Cartons(x,y)

|  |  |
| --- | --- |
| Variables | x- is a index y- contain the information below: |
| Cartons(x,0) | Carton Type |
| Cartons(x,1) | Min Usable Volume |
| Cartons(x,2) | Max Usable Volume |
| Cartons(x,3) | Max Weight |

# Sub

## Assign\_Product\_To\_Largest\_Carton

|  |  |
| --- | --- |
| Variables Passed |  |
| Function Variables |  |
| Function location | 4002-4040(38) |
| What it does |  |
| Function that call this function |  |
| Functions this function call |  |

## Create\_Audit\_Outbound

|  |  |
| --- | --- |
| Variables Passed | ord |
| Function Variables |  |
| Function location | 6228-6247 |
| What it does |  |
| Function that call this function |  |
| Functions this function call |  |

## Create\_OBS\_SQL

|  |  |
| --- | --- |
| Function Variables |  |
| Function location | 3851-3908 |
| What it does |  |
| Function that call this function |  |
| Functions this function call |  |

## Delete\_Error

|  |  |
| --- | --- |
| Variables Passed | shipid |
| Function Variables |  |
| Function location | 4833-4841 |
| What it does |  |
| Function that call this function |  |
| Functions this function call |  |

## Display\_PieceVol

|  |  |
| --- | --- |
| Function Name |  |
| Variables Passes | Piecevol |
| Function location | 4451-4498 |
| What does it do | Displays the Piece pick information |
| Function Variables | X- for loop counter  Nullval-“NULL”  Linevl- string of information being printed. |

Another display function. Loops through and displays all the piece pick information( Product ID, Volume, CartonType, VolREm, CartonID, PPLoc)

## DisplayProdPcs

|  |  |
| --- | --- |
| Function Name |  |
| Variables Passed | PieceVol(array) |
| Function Location | 4502-4525 |
| What it does | Displays the information in PieceVol Array |
|  |  |

Display Prod Pcs is a simple sub that just print the information that cartonize collected onto the handhelds.

## LOG\_ERROR

|  |  |
| --- | --- |
| Variables Passed |  |
| Function Variables |  |
| Function location | 4758-4825 |
| What it does |  |
| Function that call this function |  |
| Functions this function call |  |

## Remove\_Allocation

|  |  |
| --- | --- |
| Variables Passed |  |
| Function Variables |  |
| Function location | 4849-4990 |
| What it does |  |
| Function that call this function |  |
| Functions this function call |  |

## Sort\_Array

|  |  |
| --- | --- |
| Function Name |  |
| Variables Passes | Piece vol-Array being sorted  sortColumn- where to start sorting from( 1 or 56)  sortSeq- tells function if it’s “ASC” or “DESC” sort |
| Function Location | 4240-4349 |
| What it does | Sorta the array to be more easily picked |
| Function Variables | Dim1- length of the first dominion of the array  Dim2-length of the second dimension of the array  StrSortCols- sort colum string  LenSSC- length of the number of characters in |
|  |  |

First thing the function does is find out how large the array is(4245,4246), then prints it to the debug log. It then checks if it is an ascending or descending sort(4277-4282) it then goes through and takes the passed in information from the array and makes it easier to read(4277-4324) then it sorts all the information by either ascending or decending(4328-4342)

## Update\_OBS\_QC

|  |  |
| --- | --- |
| Variables Passed | ord |
| Function Variables |  |
| Function location | 6184-6220 |
| What it does |  |
| Function that call this function |  |
| Functions this function call |  |

# Global Variables

|  |  |
| --- | --- |
| AllocationStatus |  |
| AllowMultThreads |  |
| CarrierID |  |
| CartonCnt |  |
| CartonID | number of cartons in an order |
| CartonIDX |  |
| Cartons(2000,4) |  |
| cn | set to afxGetCOnnection |
| CodAmt |  |
| Connectstr | unused |
| Const LargestCartonIdx = 0 |  |
| Const PieceVolColCnt = 6 |  |
| cpwd |  |
| CrLf |  |
| CrLf = Chr(13) & Chr(10) |  |
| crypto | unused(legacy) |
| CurrInvAvailIdx |  |
| DepAmt |  |
| Done |  |
| dsn | Unused(Legacy) |
| DtTm |  |
| ErrLoop |  |
| Freight |  |
| Fso |  |
| FunctionName |  |
| GoodsCost |  |
| I | For Loop Counter |
| InOrderId |  |
| InsertStmt |  |
| InsertStmtD |  |
| InvAvail(2000,5) |  |
| InvAvailCnt | Look in |
| J | For Loop COunter |
| LargestVolIdx |  |
| LineNm |  |
| LineVl |  |
| logfile |  |
| logfile\_str |  |
| MaxCPL | Max cases per line |
| MaxThreadsAllowed |  |
| MM\_systbl |  |
| Nm |  |
| OBS |  |
| OBSCartonCnt, |  |
| OBSCaseCnt |  |
| OBSD |  |
| OBSDCartonCnt |  |
| OBSDCaseCnt |  |
| Order\_Cancel\_Request |  |
| orderby |  |
| OrderCnt |  |
| OrderErrCnt |  |
| PaymentTerms |  |
| PID | used as a unique thread ID for insuring that order is not being worked on by another cartonize thread |
| PieceVol(5000,7) |  |
| PieceVolCnt |  |
| PPSortType |  |
| PreviousProd |  |
| ProcessXferOrder |  |
| ProdPcCnt |  |
| ProdPcs(5000,10) |  |
| pwd |  |
| Rs |  |
| rs1 |  |
| rs2 |  |
| rsBRT |  |
| sel\_clause |  |
| sql |  |
| sql1 |  |
| sqlBRT |  |
| Status |  |
| StrError |  |
| StrText |  |
| sysid | set to 0000 |
| sysname | system name, set to default(used for finding data in the registry) |
| ThreadStarted |  |
| TotVolRem | Total Volume left to be cartonized |
| TraceData |  |
| TraceMsg, |  |
| TracePoint |  |
| trc | Set to WMSysten.TraceEvent |
| uid |  |
| UnitsPerCase | Number of Eaches per case |
| Vl |  |
| whr\_clause |  |
| X | For Loop Counter |