

## **Retek Price Change Update [pcext]**

### **Design Overview**

The Price Change Extract (pcext) module selects permanent price change records, which are set to go into effect tomorrow and updates the RMS item/location and item/zone tables with the new prices. The price changes processed by this program include both price change events set up in the pricing module and price changes generated by an items zone group changing. In addition to the updates of the retail prices in the system, records are also written for price history and transaction-level stock ledger. If an item/zone price change results in a change to the price in the base zone, then the retail at all warehouse locations will also be updated. (Note: If the price change is for an item above the transaction level, then this price change also affects all of the transaction level items which are below this item.) For ITEM zone group changes, all stores are updated with the price from the base zone. For regular price changes the new retail is applied to the appropriate stores within the zone. Once this program has processed a price change, the status of the price change is changed to Delete.

If the Batch with Online Users indicator is 'Y', meaning users can be in the system at any given time, functionality to check for records locked by other users is implemented to ensure that any record being updated or deleted is not currently being used. Checks were also implemented to ensure that in cases where a price change has multiple records associated to it, if one of the associated records was not processed due to locks, all the other records relating to it will not be processed. Any locked records that are encountered are written to the batch\_lock\_log table, and will be skipped. If no locks are encountered, processing will continue, and records relating to the price\_change that will take effect the next day will be processed.

<b>Tables</b>	<b>Index</b>	<b>Select</b>	<b>Insert</b>	<b>Update</b>	<b>Delete</b>
PERIOD	No	Yes	No	No	No
PRICE_SUSP_HEAD	No	Yes	No	Yes	No
PRICE_SUSP_DETAIL	No	Yes	No	Yes	No
PRICE_ZONE	No	Yes	No	No	No
PRICE_ZONE_GROUP_STORE	Yes	Yes	No	No	No
ITEM_ZONE_PRICE	Yes	Yes	Yes	Yes	Yes
ITEM_SUPPLIER	No	Yes	No	No	No
ITEM_MASTER	Yes	No	No	Yes	No
ITEM_LOC	Yes	No	No	Yes	No
ITEM_LOC_SOH	Yes	No	No	Yes	No
PRICE_HIST	No	No	Yes	No	No
SUP_DATA	No	No	Yes	No	No
TRAN_DATA	No	No	Yes	No	No
BATCH_LOCK_LOG	No	No	Yes	No	Yes

### **Scheduling Constraints**

Processing Cycle: Phase 3 - after pccrxt

Scheduling Diagram: No changes

Pre-Processing: N/A

Post-Processing:	N/A
Threading Scheme:	The threading mechanism will be price change. The v_restart_price_chg view will be used.

### **Restart Recovery**

The Logical Unit of Work for pccxt is price change transaction type (Regular or Zone price changes), price change and item. Processing is done as master-detail, however, price\_susp\_detail records are updated as they are processed and so commits can occur at any time during the processing (i.e. the commit process does not need to occur after an update to the header record).

### **Shared Modules**

ITEM\_ATTRIB\_SQL.GET\_BASE\_COST\_RETAIL – gets the base cost/retail in the primary currency.

SUPP\_ITEM\_SQL.GET\_PRI\_SUP\_COST – gets primary supplier and unit cost for a given item or item/location.

CURRENCY\_SQL.CONVERT\_BY\_LOCATION – converts currencies

UOM\_SQL.CONVERT – converts to the standard unit retail.

STKLEDGR\_SQL.TRAN\_DATA\_INSERT - inserts records to tran\_data table

PRCCHGTYP - determines type of price change (markup, markdown, etc.)

### **Function Level Description**

#### **init()**

This function fetches the system date for tomorrow. It also fetches the vdate and the btch\_w\_usr\_ind value from the system\_options table. This function checks the value of the btch\_w\_usr\_ind field. If the indicator is set to 'Y', the program deletes from the batch\_lock\_log table based on the program\_name and thread\_val.

#### **process()**

This function will loop through the records retrieved by the driving cursor. Each item within a price change will be processed. Once the item has been processed, the price\_susp\_detail record is updated to indicate that it has been processed. If the price change is a zone price change, then the item\_zone\_price table is purged of records for that item that have the old zone information. Finally, if all of the items within the price change have been processed the price\_susp\_head record status is updated to deleted (to be deleted). Functions called from process() include: SIZE\_ARRAYS(), GET\_ADDITIONAL\_ITEM\_INFO(), PROCESS\_ZONE(), PROCESS\_ITEM(), POST\_ALL(), SET\_PSH\_PSD\_UPDATE(), RETEK\_FORCE\_COMMIT(), and SET\_Izp\_DELETE().

Prior to processing, if system\_options.btch\_w\_usr\_ind is set to 'Y', the records retrieved by the driving cursor are written to a temporary table called price\_extract\_lock\_temp. The records in this table are then checked for locks. If

locked records are encountered, the psh\_row\_id of the last record in the array is copied to a local variable, and a process remaining flag is set if the first rowid in the array is the same as the previous rowid from the first array fetched. A record is written to the batch\_lock\_log table, and the price\_extract\_lock\_temp table is purged. The value in the local variable is then compared to the next array of retrieved records. If any of the rowids in the array match the rowid contained in the variable, all the records in the array are written to the batch\_lock\_log table and are skipped for processing. If no locks are encountered, processing continues.

**Get\_additional\_item\_info()**

Fetches additional item information which is not fetched by the driving cursor. Includes calls to get\_stock\_on\_hand(), get\_cost\_at\_location(), get\_base\_zone(), and get\_old\_retails().

**Get\_stock\_on\_hand()**

Retreives the total of the stock\_on\_hand plus the in\_transit\_qty and whether the item is on clearance or not for each location selected.

**Get\_cost\_at\_location()**

If the item is not a pack item, the unit cost is selected from the item\_loc table. If it is a pack item, a call is made to package function GET\_PRI\_SUP\_COST.

**Get\_base\_zone()**

For regular price changes, the base zone can be retrieved from the item\_zone\_price table, but for zone price changes the base zone must be retrieved from the price\_zone table.

**Get\_old\_retails()**

Gets old pricing information from the item\_zone\_price table and fills the pos\_mods array for future inserts.

**Get\_base\_cost\_retail\_prim()**

Called only if the current zone is the base zone. Gets the base cost/retail in the primary currency.

**Vendor\_funded\_credit()**

-- Not applicable in phase 2 --

**Convert\_uom()**

Wrapper function to call package function UOM\_SQL.CONVERT. (In order to use the package function to convert unit retails, the uom's are inversed.)

**Set\_psh\_psd\_update()**

Updates the array pt\_psh\_psd\_update with the new rowid and changes the status to 'D' for deleted. It then increments the counter ior\_psh\_psd\_update.l\_count by 1.

**Post\_psh\_psd\_update()**

Contains the SQL statements which updates the status of the price\_susp\_detail and price\_susp\_head tables to a status of 'D' for deleted.

**Set\_izp\_delete()**

Updates the array par\_izp\_delete with the new item and zone\_group\_id, and then increments the counter par\_izp\_delete.l\_count by 1.

**Post\_izp\_delete()**

Contains the SQL statements which delete records from table item\_zone\_price. The counter par\_izp\_delete.l\_count is then set to 0.

**Set\_im\_update()**

Updates the array par\_im\_update with the new rowid and retail\_zone\_group\_id and then increments the counter par\_im\_update.l\_count by 1.

**Post\_im\_update()**

Contains the SQL statements that update the retail\_zone\_group\_id, last\_update\_datetime, and last\_update\_id on the item\_master table. The counter par\_im\_update.l\_count is then set to 0.

**Set\_izp\_update()**

Updates the array par\_izp\_update with the new item, zone\_group\_id, zone\_id, unit\_retail, selling\_unit\_retail, selling\_uom, multi\_units, multi\_unit\_retail, and multi\_selling\_uom. The counter par\_izp\_update.l\_count is then incremented by 1.

**Post\_izp\_update()**

Contains the SQL statements that update the unit\_retail, selling\_unit\_retail, selling\_uom, multi\_units, multi\_unit\_retail, multi\_selling\_uom, last\_update\_datetime, and last\_update\_id on the table item\_zone\_price. The counter par\_izp\_update.l\_count is then set to 0.

**Set\_il\_update()**

Updates the array par\_il\_update with the new rowid, unit\_retail, selling\_unit\_retail, and selling\_uom and then increments the counter par\_il\_update.l\_count by 1.

**Post\_il\_update()**

Contains the SQL statements that update the unit\_retail, selling\_unit\_retail, selling\_uom, last\_update\_datetime, and last\_update\_id on the item\_loc table. Counter par\_il\_update.l\_count is then set to 0.

**Set\_izp\_insert()**

Updates the array par\_izp\_insert with the new item, zone\_group\_id, zone\_id, unit\_retail, selling\_unit\_retail, selling\_uom, multi\_units, multi\_unit\_retail, multi\_selling\_uom, and base\_retail\_ind and then increments the counter par\_izp\_insert.l\_count by 1.

**Post\_izp\_insert()**

Contains the SQL statements that insert records into the item\_zone\_price table. The counter par\_izp\_insert.l\_count is then set to 0.

**Set\_ph\_insert()**

Updates the array par\_ph\_insert with the new tran\_type, reason, item, loc, loc\_type, unit\_cost, unit\_retail, selling\_unit\_retail, selling\_uom, action\_date, multi\_units, multi\_unit\_retail, and multi\_selling\_uom and then increments the counter par\_ph\_insert.l\_count by 1.

**Post\_ph\_insert()**

Contains the SQL statements that insert records into the price\_hist table. The counter par\_ph\_insert.l\_count is then set to 0.

**Set\_sd\_insert()**

Updates the array par\_sd\_insert with the new dept, supplier, day\_date, tran\_type, and amount and then increments the counter par\_sd\_insert.l\_count by 1.

**Post\_sd\_insert()**

Contains the SQL statements that insert records into the sup\_data table. The counter par\_sd\_insert.l\_count is then set to 0.

**Post\_all()**

Includes calls to additional functions which post all changes to the database.

**Process\_zone()**

Does any processing that needs to happen only once per zone and performs housekeeping for the previous Logical Unit of Work

**Common\_zone\_processing()**

This does the work that is common to both if branches in function process\_zone(), such as converting the selling unit retail to the standard unit retail and prepares the data for insert/update into table item\_zone\_price. If a price is not changing, it is set to the old price. If the item is in approved status, it calls the function set\_ph\_insert to insert record into table price\_hist.

**Process\_item()**

Implements business logic on the item whose price is changing. The item is skipped if it is a clearance item. If the zone group has changed, the table item\_master is updated with the new zone group. Otherwise, if it is a regular price change, it ensures that the zone group didn't change. If the unit retail has changed, the single price changed and there exists an item\_loc record and the item is not a pack, the function set\_il\_update() is called to update the item\_loc table. If the item is in approved status, and an item\_loc record exists, call function set\_ph\_insert() to insert into table price\_hist.

**Convert\_by\_loc()**

The CURRENCY\_SQL.CONVERT package will convert currencies according to passed parameters. The intent of this function is to reduce the amount of code by allowing the package to be called generically from anywhere in pccxt.

**Update\_wh\_base()**

Loop through all item/warehouse combinations. First convert the new unit retail to the warehouse currency by calling convert\_by\_loc. If the retail is different, update the item/warehouse table with the new retail and call insert\_tran to insert tran\_data records for the price change.

**Insert\_tran()**

This function is called every time the retail price is changed at a stock location. If the total stock quantity (stock on hand + transfer quantity) is greater than zero then the transaction type (markdown, etc) is determined by the calc\_tran\_type function. The retail information is converted into the primary currency, and then tran\_data (via the STKLEDGR\_SQL.TRAN\_DATA\_INSERT package) records are inserted.

**Calc\_tran\_type()**

The PRCCHGTYP stored procedure is called to determine the tran\_data type code.

**Size\_arrays()**

Sizes the arrays to hold data from the different fetches.

**Free\_arrays()**

Frees the memory allocations of the arrays and global variables.

**Check\_lock()**

This function is called within the process() if the system\_options.btc\_h\_w\_usr\_ind value is 'Y'. This function checks for locks on the price\_susp\_head, price\_susp\_detail, item\_master, item\_loc, item\_zone\_price and price\_deal\_temp tables, using criteria similar to the criteria used in the driving cursor. If locks are encountered on any of these tables, the function returns RECORD\_LOCKED.

**I/O Specification**

N/A

**Technical Issues**

N/A