icell and iconnect data structure

# INTRODUCTION

The aim of this document is to provide the technical personal an overview of the structure of data in the iConnect database within the iCell Point of Sale System.

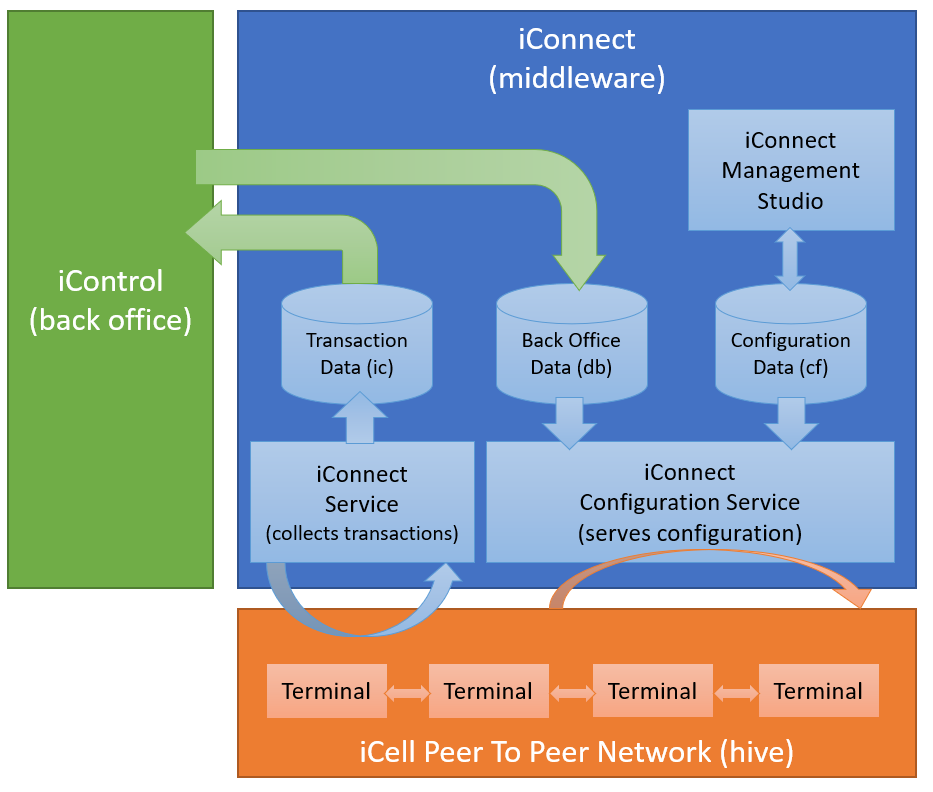
# OVERVIEW OF the FINCHCORP POINT OF SALE SYSTEM

The following diagram shows the simplified data flow of data to and from the tills via iConnect.

The core processes are:

* iControl (back office software, outside the scope of this document)
* iConnect (middleware, includes the database and a number of software modules)
* iCell (till software, peer to peer, operates serverless)

The general flow of data is shown in this diagram:



The main three types of data involved:

* Transaction Data is collected from the tills by iConnect Service and accessed by iControl (and external views), these tables are all prefixed with ‘ic’ (e.g. icTransaction) and are sourced from the transaction XML flies collected by iConnect Service.
* Back Office Data is sent to iConnect by iControl and contains data the till requires for its operation (e.g. product lists, cashier lists etc). This data is delivered to iCell by iConnectConfigService. These tables are all prefixed with ‘db’ (with a few exceptions).
* Configuration Data is edited and configured via iConnect Management Studio and contains all the settings and non core data required by the till (e.g. till user interface definitions, payment gateway configurations, printer configurations etc). These tables are all prefixes with ‘cf’. Configuration data plays no part in reporting and will be outside the scope of this document.

Database Setup:

* SQL Server Express 2016 or higher/later.
* An SQL database schema ‘iConnect’ which holds the core data for iCell
* An SQL database schema ‘iControl’ which is a transition database used to move data from Firebird iControl to iConnect
* Transaction Data, Back Office Data and Configuration data tables
* A number of support views and stored procedure including the XML to table translation procedures.

# backoffice data

The following describes the main back office tables (db) used by the software.

**Technical Note**: Tables of the form dbLocationXXXX are documented interchangeably with tables of the form dbXXXX (with the exception of the 4 hierarchy tables). The configuration service automatically sees if a table starts with dbLocation and adds a locationid=n filter to the query. Given this is automatic, I will document the root table names and leave it up to the programmer to determine if the table is location based or not. If both kinds of table exist, then use the Location based table. A couple of exceptions to this rule exist, and these differences will be described where appropriate.

Note that most Backoffice tables have two fields ChangeType and ChangeDate – these are for internal change tracking and can be ignored.

Each of the iConnect back office tables has a view in the SQL iControl database with identical name and almost identical field structure (the views do not have ChangeType and ChangeDate.

These views select data from the Firebird linked server defined in SQL Server, and create views identical to the iConnect targets. From this point there are two SQL Stored procedures that do the data migration:

* **dbMergeiControlFBtoiControl**: This copies the Firebird SQL tables to SQL iControl copies
* **dbMergeiControlToiConnect**: This merges the iControl views (add/update/delete) into the iConnect database.

The above two stored procedures are called by the iControl software.

## The hierarchy

The hierarchy is a set of 4 tables that defines the business hierarchy of the company.

Each table has a number of common fields

* Id: The unique Id for this entry
* Name: A descriptive name for the entry
* ParentId: The Id of the parent entry (e.g. LocationId)
* EntryService: The Id of the iConnect Service responsible for collecting from this entry (e.g. TillService)
* Enabled: A Y or N flag determining if this entry is enabled.

The 4 tables are:

* dbTills: A list of all the unique tills, these till ids are assigned to physical machines via the cfMachines table.
* dbLocations: A list of all the locations, with additional business (ABN, address) and the class of the location (e.g. Bar, Retail etc).
* dbVenues: No additional information
* dbBusiness: There is no ParentId field in this table.

## CUSTOMERS

The dbCustomers table is a list of account and members maintained in iControl and has the following fields:

* Id: Unique ID of the customer
* Type: Location (e.g for stock transfers), Debtor (for account holders)
* Name: Full name of customer
* Tags: Tags used to limit or find customers
* Stock: Flag determining if the customer is used for stock transfers (‘Y’) or sales (‘N’)
* Limit and Balance: Provide financial details about the customer’s account. These will be 0 for stock customers.
* Security: Cashier Security level required to perform some actions on this customer (e.g. view balances).

## CASHIERS

The dbCashiers and dbCashierTokens table store information about till users (cashiers) and their login details. Tokens are unique sequences that a cashier uses to login (normally a user id or a wrist scan code etc).

The fields in dbCashiers are:

* Id: A unique Id for the cashier
* Name: A short name for the cashier used on dockets
* FullName: The full name of the cashier
* Security: Security level of cashier, this is used by the iCell user interface to disable certain functions.

The fields in the dbCashierTokens are:

* CashierId: Unique Id of the cashier this token belongs to
* TokenId: Token Id that when scanned/typed points to the cashier.
* TokenType: e.g. KEYBOARD for entering via pin pad on till, or WRIST for proximity sensors.
* TokenPassword: For a KEYBOARD token, an optional password to validate the cashier token.

## INSTRUCTIONS

Instructions allow cashiers to add options to products (e.g. chips, sauce, steak cooking etc). These tables formalise these options.

The table dbInstructions identifies the instructions that can be assigned to products:

* Id: An ID that is assigned to a product to (via the dbLocationProducts table)
* Priority: For Muliple instructions of the same name, this prioritises them (orders them).
* GroupId: Points to the dbInstructionGroups table unique ID
* Minimum/Maximum : Minimum and maximum number of options from this group to choose (minimum of 0 means optional, minimum/maximum of 1 means you must choose 1).

The table dbInstructionGroups identifies the actual options available to select on the instruction picker:

* Id: The Id pointed to by GroupId in dbInstructions
* Values: A comma separated list of the actual selectable items for this group
* Minimum/Maximum : Default values for limits of selection, can be overridden by the values in the specific instruction.

## BEHAVIOURS

Behaviours are unique to iCell and they control the pricing of products on the till, because iCell operates in serverless mode – the till has to be able to determine the pricing of products independently.

Behaviours historically came from iControl and ended up in the dbBehaviours table, the entries in this table were parameters for rules defined in the behaviours file stored in the cfTemplates table.

This system has recently been replaced with a consolidated system where some behaviours are configured via iConnect Management Studio and others by iControl. They are now stored in cfBehaviours and cfBehaviourInstances.

The contents of these tables is outside the scope of this document, except the fact that the sales information may refer to an instance to document a special price that occurred. Treat this ID as just a promotion or special ID, with no need to dereference it.

## THE CATALOG

The catalog is a comprehensive set of data describing many aspects of the available products. In addition a few tables within the products table structures break the rules on naming relating to location.

### Product core information

dbProducts   
Holds primary information about each product which is global in nature.

* Id: Primary unique ID for the product (defined by iControl)
* Parent Id: Primary unique ID for the product parent (e.g when a product might be sold in cartons, six packs singles)
* Name: The general name of the product used on receipts
* FullName: The full name of the product without length limits
* ButtonCaption: The name to be used on button captions on the till.
* Description: Full description of product (For menu boards)
* Tax: A Y/N flag determining if this product is taxed, note that complex taxation rules can be put into behaviours.
* Enabled: A Y/N flag determining if the product is enabled site wide.
* Size: The volume or weight of the individual container of the product (the minimum consumption unit) e.g. 375ml
* Container: The descriptive name of the above container e.g. a bottle
* PackageQty: How many of this item are contained within the item delivered from the supplier (e.g. 4 six packs in a carton delivered).
* Package: Description of how many of the minimum consumption unit exist in the package sold to the customer (e.g. a 6PK)
* Barcode: Primary barcode, APN or PLU of the item at this point.
* WebName: Name customised for display on web stores (via icaas)
* WebDescription: Description customised for display on web stores (via icaas)
* RRP: The recommended retail of the product (before supplier, retailer discounts).
* Supplier: The name of the supplier of the product.

dbLocationProducts

Holds overrides and information about products unique to locations.

* Id/LocationId : Uniquely specify the product id and location Id for this override
* Cost: Cost of product from supplier
* Sell: Normal sell of product at this location
* InstructionId: Optional pointer to instruction used to configure this item at point of sale.
* PurchaseMin: The minimum retail purchase quantity (informational only – not enforced).
* PurchaseMax: The maximum retail purchase quantity (informational only – not enforced)
* Enabled: A Y/N flag determining if this product is enabled at this location.

dbProductAPNs

Holds alternate IDs for products (APNs, PLUs etc)

* Id: Primary Product Id this entry relates to.
* APN: Scancode, barcode, PLU or other ID that if scanned maps to the primary product id
* Quantity: If this APN is scanned, what quantity of the product it represents.
* Enabled: A Y/N flag determining if this entry is enabled.

dbProductOverride

Holds name/description overrides sourced from a till (using the DBUPDATE command at the till), this allows for temporary changes without changing the iControl data.

* Id: Primary product Id for the product this entry relates to
* Name: Override name (required)
* Description: Override description (null if no override)

dbProductAttributes

This table holds custom information about a product that is inherent to the product (e.g. Country or year for wine, brand etc).

* ProductId: Primary Product Id for the product this entry relates to
* Name: Name of attribute (e.g. Country)
* DataType: The datatype for this attribute (e.g. s for string, i for integer)
* Value: The value of the attribute (e.g. Australia)

dbProductProperties

This table contains custom information about a product that is not inherent, generally it will contain various groups and hierarchies the end user configures the product. Note – the locations version of this table does not adhere to the normal naming convention (for historical reasons) and is dbLocationProperties

* ProductId: Primary product Id for the product this entry relates to
* Name: Name of the property (e.g. DEPT)
* Value: Value of the property (all properties are considered to be strings, e.g. BULK BEER)

cfSundries

This table is managed via iConnect Management studio, and contains a customisable list of non-product items (e.g. COVER, DELIVERY, TIP etc) which are added to the catalog and can be used at the till as if they are products.

* Id: A unique non numeric identifier for this sundry, this will be used as the product ID when adding this sundry to a transaction.
* Name: The name of the sundry, used on receipts etc.
* Tax: A Y/N flag to determine if this ‘product’ is taxed.
* Price: An optional price for this sundry if appropriate.

### product pricing information

dbLocationPricing

There is no non-location version of this table, so only the location version is described.

This table stores various price points or price levels that this product is normally sold at.

* ProductId: Primary Product Id for the product this entry relates to
* LocationId: The Location ID this pricing is valid at.
* Name: The identifier for this pricing level, generally of the form PromoName/Level – for example PROMO/2
* Value: The price the item will sell for if this pricing level is selected for (before behaviours)
* Enabled: A Y/N flag to determine if this price level is available.

dbTillPricingOverride

Much like the dbProductOverride, this table stores pricing overrides initiated by a till (using the DBUPDATE command), however these changes are customisable on a till by till basis. These price overrides will override those given in the dbLocationPricing table with the same productid/name.

* ProductId: Primary Product Id for the product this entry relates to
* TillId: The till id where this price override is valid.
* Name: The name of the price to override (see dbLocationPricing)
* Value: The price this item will sell for if this override is active (before behaviours)

dbLocationPromoPricing

This table stores simplified promo pricing (that can replace the behaviour system for trivial conditions). Note that this table may be deprecated in the near future.

* ConditionalId: A unique Id (for the location) for this conditional pricing, consider it the promo/special ID
* StartDate/EndDate: The active date range for this promotion.
* CustomerType: If not null, further limits the promotion to a specific customer type (e.g. SILVER)
* Enabled: A Y/N flag to determine if this pricing is enabled).

dbLocationConditionalPricing

The modern version of dbLocationPricing, this table allows definition of trivial promotions where the flexibility of the behaviours system is not required. This is currently only informational and delivered to the web stores for display.

* ProductId: Primary Product Id for the product this entry relates to
* PromoId/PromoGroup: Define the unique ID of this promotion.
* PromoDescription: The friendly description of the promotion.
* PromoStart/PromoEnd: The valid date range for the promotion.
* PromoQuantity: How many items trigger the promo price.
* PrmooSell: The per item normal sell price if this promotion is active.

#### PRODUCT STOCK CONTROL

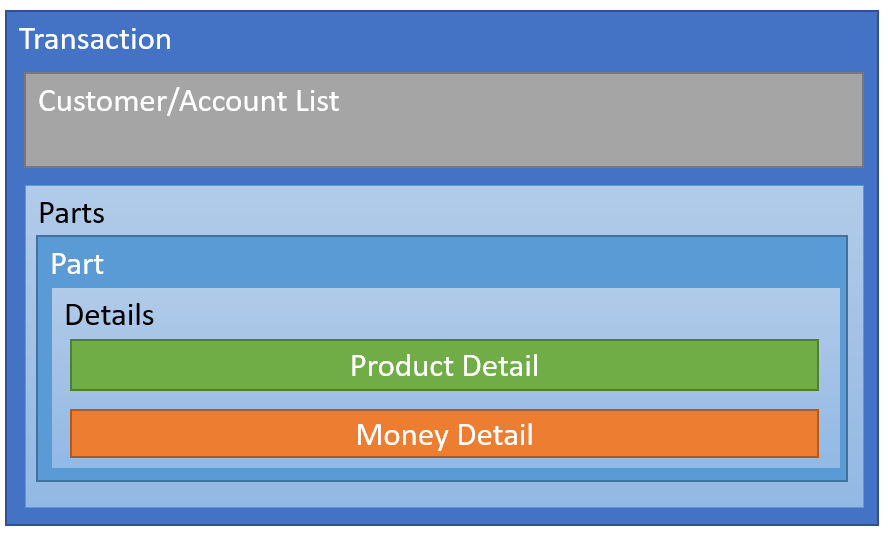
The tables dbLocationStockLevels, dbProductLocations, dbProductRecipes and icStockLevels are all part of stock control and stock levels in iCell and will be discussed in the iCell data part of this document.

# ICELL DATA

## ICELL TRANSACTION FILE OVERVIEW

Transactions are stored on the till whilst active, they are then queued on the till when complete and collected by iConnect Service regularly and stored in the database. Both the original XML and an extract of the data into table form are stored in the database.

Firstly, the transaction format is hierarchical in nature as the diagram shows:



The transaction consists of the following main parts:

* Transaction: Stores information relevant to the entire transaction (business information, net value etc)
* Customer/Account List: A list of all the customers/accounts/loyalty entities involved in this transaction.
* A list of Transaction Parts, a part is defined to be a section of the transaction which is complete, even if the entire transaction is ongoing. In the current part, pricing of ALL items in the part can fluctuate as behaviours alter pricing based on purchases. In a completed part – changes are frozen, and it can be treated as a ‘sub transaction’ which is complete.
* Each Part has a list of details, these details describe individual activities taken on the transaction (e.g. sold 3 beers, received $50 etc).

The Transaction file form is ‘net zero’ format, meaning that any complete transaction must sum to zero. Primarily this means that any value leaving the till is considered to be negative in value (so a sold product is -1 item), and value entering the till is considered to be positive in value (so a returned product is +1 item). This is universal throughout the file format and data and is sometimes counterintuitive as the number of sold products can seem to be a negative number.

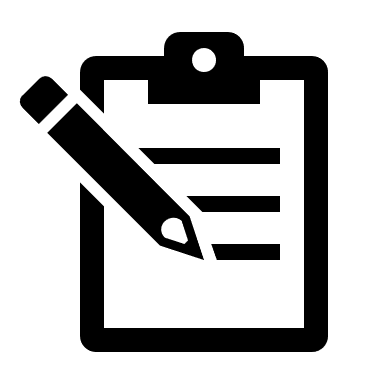
It does that by breaking each detail into one of three kinds:

* Product Detail: These details involve the sale or return of actual items.
* Money Detail: These details involve the receipt or refund of actual money, or some money substitute (e.g. account charge, loyalty redemption etc)
* Notify Detail: These details are considered to be neutral, they are activities that neither add nor remove value to the transaction (e.g. a cashout request before being processed).

## ICELL TRANSCATION DATA

The way this data makes its way into the database is as follows:

* The Till finalises a transaction and places it in the local Queue folder
* The iConnect Service responsible for that till/location collects the transaction into its server queue path.
* The iConnect Service then processes the queue into the database
* The database has a function icInsertTx which receives the transaction
* It first places the transaction into the icTransactionQueue including the entirety of the XML.
* It then attempts to process the XML into various tables structured in the same hierarchical layout as the previous section explained.

Many of the fields in the database are string representations of enumerated types, the values of these will be documented in the last section of this document. Those fields will be marked with a .

Non core tables (such as the email table, the error logging tables, the collected reports table are currently outside the scope of this document.

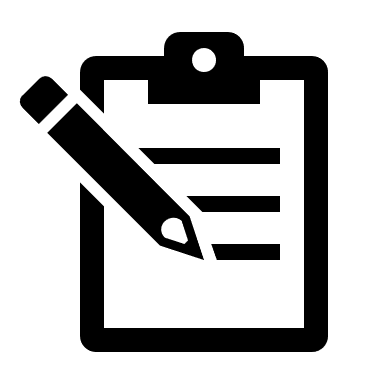
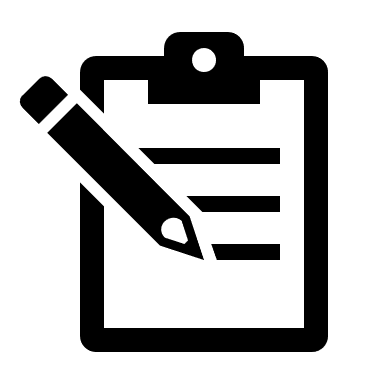
icTransactionQueue

This table stores the incoming transaction files in their raw format, and includes information about the processing status of the transaction.

* TxId: Unique 9 character transaction Id
* TxStatus: Determines the status of this transaction (E = error or incomplete, Y = processing complete, T = till time error)
* TxLast: Last attempt/activity for this transaction in the queue
* TxSuccess: Date and time that the transaction was successfully processed.
* TxCompleted: Date and time when iControl has completed processing this transaction
* TxError Columns: Details of any error that occurred on processing, NULL if no errors.
* TxData: XML typed raw XML data.
* TxSessionId: iControl internal use.

icTransaction

This table stores the top level information about the transaction.

* TxId: Unique 9 character transaction id
* TxType: The type of transaction it is (how it functions whilst active) for example Bistro
* TxClass: The state of the transaction (e.g Tax Invoice, or cancelled). 
* TxVersion: The till version number that created this transaction
* TxFormat: Version number of the tx format for breaking changes in format.
* TxTill, TxLocation, TxTillName,TxTillLocation: The primary till/location information for this transaction
* TxCashier: The primary cashier ID responsible for this transaction
* TxStart, TxEnd: The range of date/times covered by this transaction
* TxProduct, TxMoney, TxStock, TxLoyalty, TxTaxation, TxCost, TxQuantity: Aggregate fields added at insert to increase query performance.
* TxTradingDate: Calculated field to determine the date of this transaction for trading purposes (generally trading date ends 5am the following day).
* TxModernFormat: Should always be 1, old transaction files didn’t have any version information.

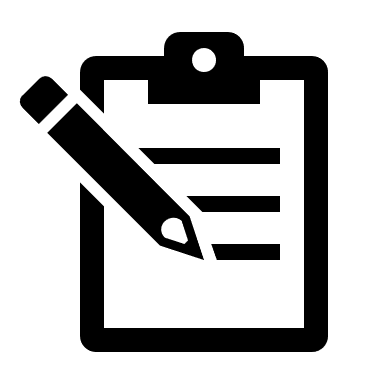
icTransactionIndex

This table stores an index of common search items, this table is updated by the initial insert procedure to speed up common transaction searches (e.g. in the journal viewer). The data is the same kind of data that is stored in the local till index folder, and in the header comment of transactions. This data is copied directly from the <index> entity in the transaction file.

* TxId: Unique 9 character transaction id
* Start, Last: Transaction start date time, and last transaction data time – for a completed transaction last and end are the same.
* TillId, TillName, LocationId, LcoationName: Details of the primary till and location for the transaction
* CashierId, CashierName: Details of the primary cashier responsible for the transaction.
* TableId, TableName: Details for the table for the transaction (or null if bar tab)
* AccountId: Details of the first or primary account/customer/loyalty entity for the transaction
* Tag: A tag applied to a transaction (e.g. WEB, WEDDING etc) to allow grouping or filtering.
* TxTotal: The total product on the sale
* TxOwing: The amount owed (will be NULL or 0 for completed transactions)
* TxLimit: The maximum spend available on a transaction (enforced by till).
* Payments: Comma separated list of Payment types used.

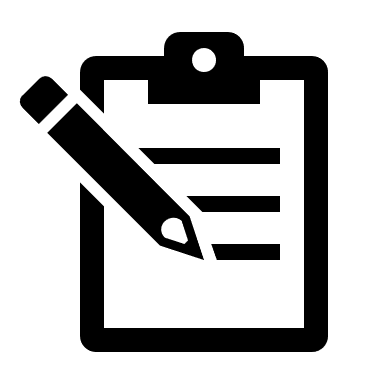
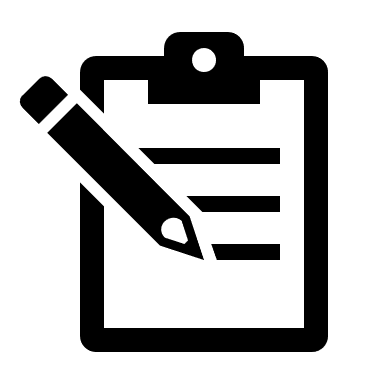
icTransactionPart

This stores the core information about each part, parts don’t have a lot of information about them, most of the information is stored in the details within it.

* TxId: Unique 9 character transaction id
* PartId: Unique (for transaction) integer part id
* PartType: Determines the part type (commonly Start, Augment or Complete) 
* PartName: Descriptive name for this part.
* PartTable: Table ID for this part (for when customers might move between tables, different per part).
* PartTime: When this part was started.
* PartLocationId, PartLocation: Location Where this part was started.
* PartPricing: Pricing level for this part **(defunct)**
* PartCustomer fields : Legacy information about customer associated with part **(defunct)**

icTransactionDetail

This table stores the core information common to most detail entries.

* TxId: Unique 9 character transaction id
* PartId: Unique (for transaction) integer part id
* DetailId: Unique (for transaction/part) integer detail id
* DetailType: The type of this detail (this might be sale, refund etc) 
* DetailCreator: Where the command to add this detail originated on the till. 
* DetailTill: The till that entered this detail (remember transactions can move around the hive, from till to till)
* DetailQty: The quantity associated with this detail, remember negative numbers meaning ‘leaving the till’, positive numbers mean ‘entering the till’. Most money details have a +1 or -1 quantity, although it is possible for 5 $10 notes to be stored with a quantity of 5 (meaning $50 taken), however this is generally not used.
* CashierId, CashierName: The cashier details for the cashier logged in when this detail was created.
* AccountKey: A pointer to the transaction account list for the account associated with this detail.
* DetailTime: The date/time this detail was created.
* ProcessedBy, Processed At: Used for account report transactions
* TabCharge: The owning TxId when tab charging allows partial transactions to be delivered to iConnect.
* CashierToken: The token the cashier had used to sign in with for this detail.
* DetailComments: Optional custom comments added to this detail.

icTransactionDetailMoney

This table stores information relevant to money detail items only.

* TxId, PartId, DetailId : Points to the detail this is associated with.
* PaymentId: Payment ID, unique if the payment interface provides one, null if none is provided.
* PaymentMethod: Points to a payment method defined in the gateways till configuration.
* PaymentBin: The ID of the bin, sub type etc the payment was made to.
* PaymentName: For authenticated payments (e.g. loyalty) the name of the customer, otherwise the same as the payment id.

icTransactionDetailProduct

This table stored information relevant to product detail items only.

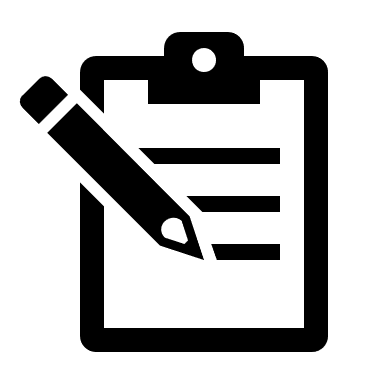
* TxId, PartId, DetailId : Points to the detail this is associated with.
* ProductId: The product id or sundry id of this product detail
* ProductSell: The normal sell (after conditional pricing, before behaviours)
* ProductCost: The cost of the product or null if not available or appropriate.
* ProductAPNs: The product APN associated with the product
* Barcode: The product barcode actually scanned
* ProductParent: The product id of the parent product.

icTransactionDetailPrice

Each **product** detail currently has four ‘pricing’ sections associated with it, these four sections are generated by the behaviour system and are:

* BASE: A semi-permanent pricing set for the product when entered
* PRICE: A fluid, changing price altered by the behaviour system for all products in a part each time any changes are made to that part.
* LOYALTY: Loyalty calculations done by the behaviour system.
* TAX: Taxation calculations done by the behaviour system.

The fields for the pricing sections are:

* TxId, PartId, DetailId : Points to the detail this is associated with.
* PriceId: One of the above four price Ids
* PriceValue: The final value per item (e.g. final price for base/price, the final loyalty or tax for the others).
* PriceDelta: For base/price this price section generated.
* PriceSender: Where this pricing change originated on the till 
* PriceSource: The behaviour id that triggered this price section if the sender is Behaviour
* PriceFormula: The formula that determines the resultant value
* PriceName: The friendly name (from the behaviour) for the change.

Price formula are of the following form : <adjustment>@<source>

Adjustments examples:

* =100 – this is an absolute amount (this defaults to =100$ or set to $100)
* -10%@DEFAULT/1 – take 10% off the DEFAULT/1 price level
* -10% - take 10% off the price we are at (so for the price section, it takes 10% of the result of the base section).
* ~10% - for tax, determine the tax from the source amount assuming it already includes the given % of tax.

icTransactionDetailGroups

Each **product** detail preserves the groups from the product properties tables to allow for processing of behaviour that depend on them. This table stores those detail groups in a format similar to the way the original tables stored them.

* TxId, PartId, DetailId: Points to the detail this is associated with.
* GroupName: Name from properties table
* GroupValue: Value from properties table.

icTransactionAttributes

Each transaction allows for the addition of custom attributes at the till. Some of these are common across all sites, others custom to a particular business. They have arbitrary names and values.

* TxId: Points to the transaction the attribute is associated with
* AttrName: The Name given for the attribute (from the transaction file)
* AttrValue: The Value of the given attribute.

icTransactionAccount

All customers, account holders and loyalty members involved in a transaction are stored in the accounts section, and then this table. Each detail that has an association with such an entry points here from the account column.

* AccountKey – unique identifier for this account (combination of payment interface/method and unique customer id for from that loyalty vendor)
* MethodId – the payment interface id for this account defined in the gateways till configuration.
* AccountId/AccountSubId – define the account id and where supported the sub id (e.g. Room Number/Guest Number)
* AccountName/AccountSubName – define the name and sub account name associated with the account.
* AccountCard – define the customer friendly card id for the customer.
* IsRemote – if ‘N’ it is sourced from the customers configuration at the till (see back office data section) if ‘Y’ it is from a non local payment gateway or provider interface.
* IsStock – if ‘Y’ then this account is a stock customer (for stock transfers)
* AccountVendor – stores the vendor used by the payment gateway for this account.

## STOCK CONTROL DATA

A combination of iControl data and iCell data gives information on stock levels at a detailed level.

dbProductLocations

This table stores the stock locations for products based on the product and sell locations.

* ProductId: The product id that this entry refers to
* SellLocation: The location at which this product is sold.
* StockLocation: The location at which the stock for this product is held.

dbProductRecipes

This table stores the ingredients, and locations of stock for items that are made up of various combinations of other items.

* SellProductId: The product ID that this compound item is sold under
* IngrProductId: The product ID of an ingredient required for the compound product
* StockProductId: The product ID of the source stocked ingredient.
* IngrQty,IngrCtnQty,StockCtnQty : Define the makeup of the ingredients from the in stock items.
* Ratio: A calculated field that determines the ratio of each ingredient required.

dbLocationStockLevels

This table stores the start of day (or shift) stock levels from the back office (iControl). This is the baseline which the iCell data stock levels are used to determine the actual stock levels.

* ProductId: Product Id that this entry refers to
* LocationId: The location id that this entry refers to
* Qty: The quantity of this item currently in stock.
* QtyOnHold, QtyOnOrder : Future use

icStockLevels

This table stores the stock changes on a per transaction basis, combined with the back office stock levels this gives reasonable stock level estimates.

* ProductId: The stock product id that this entry refers to
* LocationId: The location that the product was sold from
* Source: The transaction ID that created this stock entry
* LastUpdate: The last time this entry was changed
* Qty: The quantity of the STOCK item that was involved (-ve value is sold, +ve value is refunded).

## ENUMERATED TYPE REFERENCEClipboard with solid fill

This section will document some of the commonly found values for each of the enumerated type

### DETAIL TYPES – MONEY

* ttRefund: Money (or equivalent) is refunded to a customer bank, account or loyalty system
* ttReceipt: Money (or equivalent) is charged to a customer bank, account or loyalty system
* ttChange: Change is given when amount tendered exceeds amount owed.
* ttRounding: Rounding for under minimum currency limit (e.g 5c for cash)
* ttCashout: A cashout is given from an EFTPOS vendor
* ttDeposit: A direct deposit (without other processing) into an account
* ttWithdraw: A direct withdrawal (without other processing) from an account.
* ttBanking: Banking entry
* ttReconcile: Backing entry

### DETAIL TYPES – PRODUCT

* ttSale: A normal sale of a product, customer received goods or services.
* ttReturn: A normal return of a product, customer returns goods.
* ttSundry: A custom sundry item (except covers)
* ttCover: A cover sundry item.

### PART TYPES – PRODUCT

* tpStart: An initial part on a new transaction (should be limited to icell only)
* tpAugment: A subsequent part in an ongoing transaction (should be limited to icell only)
* tpComplete: A complete part that finalised normally.
* tpCancel: A part in a cancelled transaction
* tpVoided: A part in a voided transaction.
* tpPending: A part which has pending changes in it (icell only)
* tpNoSale: A part in a no sale transaction
* tpBanking: A part in a banking report transaction.
* tpIncomplete: An incomplete part.
* tpNotification: A notification part in a notification transaction.
* tpTab: A tab part in a tab transaction.
* tpEmpty: A purposely empty part (no details).

### Transaction ClassES

* tcNew: A new transaction with no product or money (should be only seen till side).
* tcManifest: A stock transfer
* tcInvoice: A list of products without payment.
* tcReceipt: A payment
* tcTaxInvoice: A list of products with full payment, suitable for use as a tax invoice.
* tcNoSale: No sale at till (use to open cash drawer).
* tcCancelled: A sale with no payments that was cancelled.
* tcVoided: A sale that was voided.
* tcTabCharge: A partial tab, the full transaction to be send later.
* tcNotification: A no action transaction to deliver notifications to iConnect.
* tcBanking: A baking summary transaction.
* tcReversal: A reversal of a previous transaction.

### Transaction TYPES

* ttBar: A non table, normal transaction
* ttBistro: A bistro style, ongoing transaction
* ttNormal: A normal table transaction.

### Senders/Creators (commonly used only)

* asBehaviour: The detail was created by a behaviour rule (e.g. adding a free product).
* asButton: The detail was created by a cashier action (e.g. normally pressing a button)
* asSystem: The detail was created by the system (e.g. rounding).