# A Practical Introduction to Navision Customizations

by Miklos Hollender
miklos.hollender@gmail.com
http://navitips.blogspot.com

## *The Basics*

The "Entry" tables contain the business transactions - filter for *Entry* in Object Designer, Tables. You should only enter an Entry by writing into a Journal (filter for *Journal*) table and calling the posting codeunit. With your NSC licence you can modify and delete data in Entry tables, but never write a program that does it because you can have trouble running it with the customer's licence. If you really must modify or delete data from an Entry, filter codeunits for *Edit* and look at them. But it's dangerous and should be avoided 99,9% of the time.

The core of Navision's business logic is posting Journals, which will create Entries. Based on these core are the Documents, like an Invoice. Documents create and post journals, for example, an Invoice posting could be approximated as as posting a General Journal for the header and one Item Journal for each lines. In reality, it's more complex, because of Dimensions etc. but this model will do for such an introductory material. Besides creating and posting Journals, Documents save themselves as Posted Documents. The tablenames are a bit strange: for example, an unposted invoice is a the Sales Header/Line with a Document Type field of Invoice, and a posted invoice is in the a Sales Invoice Header/Line.

## *Financial Transactions Overview*

The Primary Key of all Entry tables is always Entry No.

G/L Entry and VAT Entry tables contain G/L and VAT transactions. They are very simple, no explanation is required I think. Cust. Ledger Entry is the customer transaction table, each line is an invoice or a payment. Behind Cust. Ledger Entry there is Detailed Cust. Ledg. Entry in a 1:N relationship. Detailed Cust. Ledg. Entries show applications between payments and invoices. In the Cust. Ledger Entry the important field is Remaining Amount, which shows how much of a given invoice is not yet paid (not yet applied). (Or how much of a given payment is not yet applied to invoices.) It is a FlowField, summing up Detailed Cust. Ledg. Entries.

Never forget that FlowFields will only be calculated automatically if shown on a Form or Report. In every other cases, use the CALCFIELDS function.

FlowFilters are also very important. You can recognize FlowFilter fields easily, because by naming standard they always contain the noun "Filter", like "Location Filter", and of course the FieldClass property of the field (in Table Designer) is FlowFilter too. For example, you can write that "time machine report" which is able to show how much debt had a given customer towards us at the end

of the financial year even three months after - without writin a line of code! Such a report is very important for preparing the Balance Sheet at the end of the financial year. You just run the Report Designer with the wizard option, put the Remaining Amount (and a few other informational fields) to the report, put Posting Date and Date Filter fields in the RequisitionFilterFields property, and it's done. If the end of the financial year is, say, 06/12/31, and the user filters for <061231 in the Posting Date, he will get all invoices and other customer transactions dated before the end of year, and if he filters for <061231 in the Date Filter, then only those applications (payments) will be counted in the Remaining Amount that happened before 06/12/31, therefore the „time machine" have landed on perfectly on 06/12/31. Yes, it is truly so easy. This might be the reason many developers love Navision.

Vendor Ledger works exactly the same as Customer Ledger.

## *Inventory Transactions Overview*

Each inventory movement creates one Item Ledger Entry: for example, each Sales Shipment Line or each Purch. Rcpt. Line creates one. Item Ledger Entries have 1:N relationship with Value Entries. Value Entries provide the cost (and the sales) for Item Ledger Entries. For example, you receive an Item with a Purchase Receipt, you will get an Item Ledger Entry and a Value Entry, with only Cost Amount (Expected) because it is not yet invoiced. When you post the invoice, you will get another Value Entry with a Cost Amount (Actual). Actual simply means „invoiced" or generally, „proven, documented". When you post shipping charge to that Receipt with another Purchase Invoice with and Item Charge line, you will get another Value Entry. When half year quality of these goods got slightly spoiled in your warehouse, and you have to lower their value, which you do with Revaluation Journal, it also creates another Value Entry. These will be summed up in the Cost Amount (Expected) and Cost Amount (Actual) FlowFields in the Item Ledger Entry – the first one is Expected, all others are Actual.

Incoming and outbound Item Ledger Entries are connected by Item Application Entry if you use FIFO int he Costing Method field ont he Item Card, or if directly apply them to each other (like in the case of Return Orders and Credit Memos). They are connected indirectly in the Average Cost Adjustment table if you are using Average costing.

Whenever an Item goes out of stock (i.e.an outbound Item Journal Line is posted), it's cost will some kind of very simply calculated rough average cost. It gets connected with the incoming entries in the aforementioned tables. If you manually choose Apply-to or Apply-from entry (usually on Credit Memos to ensure correct cost reversal), then it will be connected with that entry in the Item Application Entry table, regardless of the costing method. When you later run the Adjust Costs - Item Entries batch job, which will run Codeunit Inventory Adjustment, it will create adjustment entries to correct the costs. It means it inserts Value Entries where the field Adjustment is True.

Cost adjustment has some serious bugs int he 3.X versions, so better be prepared to debug it even if you use a later version. Make sure you make a report running on Items, fields Sales (LCY) and COGS (LCY) and calculate the margin percentage from them, and have the client run it every week or so. The basic idea is that client usually knows the expected margin percentage of their Items, therefore they can identify if it's suspiciously high or low. This way you can spot errors early in the

financial year, because it's not very pleasant if you start to debug costing three days before the client has to prepare his Balance Sheet, because it might take long to get it right…

To debug Inventory Adjustment, filter it for a single Item in the MakeSingleLevelAdjustment function, and put and ERROR in whenever PostItemJnlLine is called, because Inventory Adjustment makes it's adjustment entries by calling this codeunit, and by throwing error there you can examine which code has ran in the debugger... Make sure you read the Costing White Paper before: http://mibuso.com/dlinfo.asp?FileID=382

There is another important transaction table for inventory, it's called Warehouse Entry. If you use Bins withing a Warehouse, besides the Item Ledger Entry table, inventory transactions will be parallelly registered, and in the Warehouse Entry and Bin Content tables too.

## *Overview of Journals*

Their primary key is always Journal Template Name, Journal Batch Name, Line No. The value of Line No. comes automagically from the AutoSplitKey property of the forms - if you create lines by code, your code has to create Line No.-s as well. It's a tradition to increase it by 10000, to allow space for splitting the key if the user inserts new records between two existing records.

Journals are very well centralized. All financial postings - Customer, Vendor, G/L, VAT, Fixed Asset etc. - entries come from the General Journal Line table. All inventory movements including manufacturing come from the Item Journal Line table.

There are some other, less important journals as well.

### Overview of General Journal Line

For the General Journal, the most important fields are Account and Bal. Account. Both can either be a G/L Account or an analytical account: Customer, Vendor, Bank, Fixed Asset. Whenever you can, you MUST post to the analytical account and let it manage the G/L automatically. Never let users - by the "Direct Posting" checkmark in the Chart of Accounts - to post directly to Customer or Bank G/L Accounts. And whenever a user calls you that the Customer analytical account on the Customer Card and the customer G/L account shows different amounts, filter G/L Entries for System-Created Entry = No. because it might turn out they took out this checkmark and posted to the G/L account directly.

Another important field is Document Type. 99% of the time, we use Payment for bank and cash transactions and the empty option for simple G/L Postings and the other options are not often used, because Document Types of Invoice etc. are usually done through Documents, not Journals.

A good example of General Journal use is to write off a debt: the Account Type will be Customer, the Account No. will be the Customer No., the Bal. Account Type will be G/L, Bal. Account No. will be some G/L Account. Because an Invoice is considered "paid" in a technical sense if written-off (you don't want to send more Reminders for example), you apply it to the Invoice with Applies-To Document Type and Applies-to Document No., and set Document Type to Payment.

Before go-live, unpaid Invoices should be imported in the General Journal, Account Type is Customer/Vendor, Bal. Account is an appropriate G/L Account used for opening the Balance Sheet. General Journals are posted by Codeunit 12, the code is mostly elegant and readable, it's a good idea to read it a bit.

## Ovewview of Item Journal Line

Item Journal Line posts all possible inventory movements, both in quantity and in value. It has several forms on it - Item Journal for positive and negative adjustments, Phys. Inventory Journal for physical inventory, Item Reclass. Journal for moving Items between Bins, Locations, Dimensions or Serial/Lot No.-s, Revaluation Journal for changing inventory value, Consumption and Output Journals for Manufacturing - but the table is the same.

The problem is usually with Positive and Negative Adjustments. Usually you want to post surplus inventory, missing inventory, scrapped inventory to different G/L accounts. You can do it with defining Gen. Bus. Posting Groups for them, and setting the Inventory Adjmt. Account in the General Posting Setup. However, a customization is usually necessary to make it mandatory to fill this field.

Item Journals get posted by Codeunit 22. It has very complicated code. I'd rather not suggest to try to understand it atf irst: if you got to make some minor changes such as moving a new field from the Journal to the Entries, just search for INSERT in this Codeunit and put that code before the INSERT.

## Overview of Documents

A Document is something that has a header and lines and can be posted. It creates Journals and posts them, and saves itself into a posted document. Most fields are moved to the posted document usually by the magical function TRANSFERFIELDS.

Let's look for example on posting an Invoice, for example, an Invoice which has Item lines, a kind of a retail Invoice, where you sell the goods with the Invoice directly, without making an Order before.

First it will run Codeunit 414, which will Release the Invoice - it will calculate the VAT and save in the lines into Amount and Amount Including VAT.

Then the Codeunit 80 will run which posts everything related to sales: both invoices and shipments. It will reserve a posted invoice number by the Posting No. Series field in the Sales Header, and save it in the Sales Header as Posting No. All transactions will use this as a Document No. The saving of the number is COMMIT-ted to avoid another process using the same number.

Then it will post one or more Gen. Journal Lines (usually one), Document Type is Invoice, Posting Date, Document Date etc. a lot of fields get copied from the Sales Header. Account is Customer, Bal. Account is a G/L Account which it will figure out from General Posting Setup. After the General Journal Line is created, it will call Codeunit 12 to run on it. Codeunit 12 will create Cust Ledger, Detailed Cust. Ledg., VAT and G/L Entries. However, it is not COMMIT-ted yet, so these

records right now are just hanging in the Commit Cache. But to avoid another process using the same Entry No. for the aforementioned entries, the entry tables will get LOCKTABLE-ed.

(This is why users often complain of performace issues... Native database always locks the whole table, so if somebody is posting an Invoice, then everybody else doing something that would write in the same tables will have to wait. On SQL Server there is record level locking, not only table level, which is a good idea, but there you often have deadlocks as of Navision 4.0 SP1.)

So, after this, Codeunit 80 looks at the Invoice lines. It will intelligently guess whether these goods were already shipped or not (for example looking at Quantity to Ship or Shipment No. fields). If they had not been shipped, it will create an Item Journal Line for each lines and post it. Entry Type is Sales, Source Type and Source No. is the Customer, etc. It will call Codeunit 22 to post Item Journal Lines, which will create Item Ledger Entries and Value Entries. If it has been already shipped, then some fields will be put different into the Item Journal Line so the Codeunit 12 will know it only has to create Value Entries. These records also hang on in the Commit Cache.

Then Codeunit 80 will grab a Sales Invoice Header, transfer the fields with TRANSFERFIELDS, insert it, insert the lines... and when everything is OK, the whole transactions will be COMMIT-ted. If there is an ERROR in the process, everything will be rolled back - except for Posting No. in the Sales Header, because that has been COMMIT-ted earlier.

For Purchase it's the same, only with Codeunit 90 instead of 80.

You can expect a similar logic for Transfer Orders as well. For Production Orders, it’s a bit different, because in Manufacturing all postings are made with journal forms that reference the Item Journal Line, so it’s a lot simpler. It’s like posting simple Item Journals with referencing the relevant Production Order.

## *An Example Customization Task*

Let's assume our client said that for each Sales Orders, he wants to assign a Responsible User. This user will be a warehouse employee who has to make sure everything is picked, packaged, and delivered correctly and on time.

What the client does not say but we have to know by ourselves:

- It is sure he will also want to move this information to the Sales Shipment as well, to be able to check out who was responsible for a problematic Shipment

- It is almost sure he will ask two weeks later that he wants to be able to filter those reports that run on Sales Order Lines (Sales Line table) and Sales Shipment Lines for this new field, so we have to transfer this data there as well. We can expect it even when the client does not say it, because if we imagine ourselves in his shoes, we would want to be able to check how much late order lines a given employee has, wouldn’t we?

1. Create this new field in the Sales Header, Sales Line, Sales Shipment Header and Sales Shipment Line tables. Field number must be between 50000 and 100000, and it should be the same field number in the Sales Header and Sales Shipment Header tables. Field Name is Responsible User. Data Type is Code (almost all identifiers are of this type: it's a kind of "sanitized" text type, with

the built-in magic that it can be sorted as an integer as well), length is usually 20 (some built-in Code types have only 10 but I think this is wrong practice). Press Shift-F4 for Properties. Set the Caption. Set TableRelation to User. Save and compile the object.

2. Add it to the relevant forms. Be sure to set Editable property to No on Sales Shipment header and line forms.

3. From the Sales Header to Sales Shipment Header it will move automatically by the magic of TRANSFERFIELDS (this is why we set up Field No. to be the same). For the Sales Line and Sales Shipment Line, we have to put it there by code.

4. For moving this field from the Sales Header to Sales Line, we have to add two kinds of functionality:

1) when a new line is inserted, it needs to inherit this field from the header
2) when this field is changed in the header, all lines must be changed.

In both of the cases we will write logic to the table, not the form, because in this case it will work even when the record is not inserted from the form, but from code or a DataPort. Design Sales Header table, go to the field Responsible User, press F9. Create a new local variable (in the View Menu) called SalesLine, type is record, subtype is Sales Line. Write the code in the OnValidate trigger of Responsible User:

```
CLEAR(SalesLine);
SalesLine.SETRANGE("Document Type",rec."Document Type"); // rec is a built-in
//name for current record, and it is optional, not mandatory, but good for
//readability
SalesLine.SETRANGE("Document   No.",rec."Document No."); //check out SETRANGE,
//SETFILTER in the help
IF SalesLine.FIND('-') THEN REPEAT // check out FIND in the help
    SalesLine.VALIDATE("Responsible User",rec."Responsible User");
    //check out VALIDATE in the help
    SalesLine.MODIFY(TRUE);
    //check out MODIFY in the help
UNTIL SalesLine.NEXT=0;
//check out NEXT in the help
```

(Commit is implicit here.)

This code will make run when the Responsible User is changed in the header, and will push this change into the lines.

The next thing is to make new lines automatically inherit this field from the header.

Therefore we go to the Sales Line table, OnInsert trigger, local variable SalesHeader, and write:

```
IF SalesHeader.GET(rec."Document Type",rec."Document No.") THEN BEGIN
//get is retrieving a record by the primary key
    VALIDATE(rec."Responsible User",SalesHeader."Responsible User");
END;
```

5. To move this field from the Sales Shipment Header to the Sales Shipment Line, we need to follow different logic, because by standard, these documents are inserted as INSERT and not INSERT(TRUE), therefore OnInsert trigger is not fired, therefore we should not write code there.

The best you can do is to go to Codeunit 80 and search for INSERT, case-sensitively.
You will find SalesShptLine.INSERT. Good, this is what we needed. Insert the following code BEFORE the line containing INSERT:

```
SalesShptLine."Responsible User" := SalesShptheader."Responsible User";
```

## *Homework: developing Returnable Packings Management*

### Business case

For each shipment, the quantity of packings need to be automatically calculated, written on the Shipment Note and their Inventory decreased. It is necessary to know how much is the balance of returnable packings at a customer. Packings given but not returned must be periodically invoiced.

### Logical Design

Packings can be handled as regular Items. Users need to be able to mark which Items are packings, choose a packing to use for each Items and enter how much needs to be used for a given unit of the sold Item.

A new menu item is needed on the Sales Order to automatically create lines with packings. If you have 5 different Items on the Order which only use 2 different kind of packings, only 2 lines need to be created.

As the balance of packings at the customer will never be positive (it's unlikely they would give back more than they had received), returns must be managed with negative Sales Shipments so both shipments and returns can be pulled into a Sales Invoice with Get Shipment Lines and their difference invoiced.

For this reason, Sales Shipment Lines need to be marked if they contain packings.

The balance of packings will be a FlowField in the Customer Form showing the sum of Qty. Shipped Not Invoiced for those Sales Shipment Lines that contain packings.

Packings will need to be set to Standard Costing in order to avoid troubling users with Exact Cost Reversal but still have correct costs, and Exact Cost Reversal must not apply to them.

### Development

Create a Boolean field „Returnable packing" in the Item table and Card form. In the OnValidate, throw an ERROR if Costing Method is not Standard.

Create a Code20 field, named „Returnable Packing  to Use" which has TableRelation to Item table filtered to Returnable Packing = CONST Yes.

Create a Decimal Field called „Returnable Packing Qty.” .

In Sales Orders, create a new menu Item with code, preferably putting the code in a codeunit for reusability, setting the TableNo property to Sales Header and calling it with CodeunitVariable.RUN(Rec);

This code loops through the lines – SETRANGE, FIND(’-’), NEXT – and inserts the packings as Items (Type=Item, No. = Item.”No.”) in the Order.

This is a tricky task to do: whenever multiple Sales Order Lines use the same packing, you should sum the packing up in one line.

The best thing you can do is to create the „Returnable Packing to Use” field int he Sales Line as well and populate it from the Item in the No. – OnValidate trigger of the Sales Line. Create a new key for the Sales Line as „Document No., Returnable Packing to Use”, and in this codeunit loop through the lines using this key (SETCURRENTKEY). In the loop, keep track of whether this field has changed ( IF LastRetPackToUse <>SalesLine.”Returnable Packing to Use” THEN… at the beginning of the loop, LastRetPackToUse := SalesLine.”Returnable Packing to Use” at the end of the loop) and act accordingly. Sum up SalesLine.”Quantity (Base)” * Item.”Returnable Packing Qty.” in a decimal variable and INSERT the new lines to the order.

Don’t forget to call CurrForm.UPDATE(TRUE); in the menu item code, after the Codeunit has ran, to let the user see the new lines.

It is also necessary to mark those Shipment Lines that contain returnable packings, to allow users to filter them out at invoicing on the Get Shipment Lines form, or, later on, filter for only packings and invoice them all at once. You can just move the Returnable Packing to Use field from the Sales Line to the Sales Shipment Line like in the previous example. You need to create this field of course to be able to move data into it.

Finally, create a FlowField in the Customer table and Card form, called „Returnable Packing Balance”, which will SUM the Qty. Shipped Not Invoiced of Sales Shipment Lines WHERE the Sell-To Customer No. is FIELD No., and Returnable Packing to Use is FILTER <>’’ (two aposthrophes, noting an empty string). Run the Customer form, look at the error message and create a key on Sales Shipment Line accordingly.

Design Codeunit 80, search for „Exact Cost Reversal” and change it so it does not throw an error message if the Returnable Packing to Use field on the Sales Line is <>’’ (two apostrophes).

## Notes

In a production system, you might want to consider adding a Boolean field too to Sales Shipment Line, and filter for that instead of the Code field „Returnable Packing to Use” because it might be slightly better for performance.

That was it! ☺

Good luck for further exploration. Be sure to read the Application Designer's Guide, the Solution Developer's Guide, the Essentials, the White Papers, and finally all the other manuals when you have the time. But the best thing you can read is standard Navision code. To be able to customize Navision properly and productively analyse standard Navision code so much that when somebody has a customization request, you can instantly identify a similar feature in the standard and copy the ideas from it. „Steal with pride, improve to perfection!” ☺