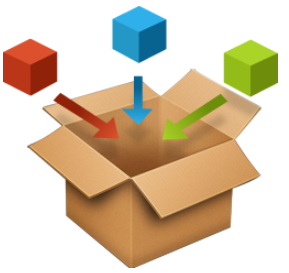
|  |
| --- |
|  |
| Kitting |



Project name:

Intercompany transactions and kitting for AX2012 R3

Prepared by:

Kurt Hatlevik

Contributors:

Table of Contents

[1 Introduction 3](#_Toc419464001)

[2 Prerequisites 3](#_Toc419464002)

[3 Scenario 3](#_Toc419464003)

[4 Intercompany Scenario tested 5](#_Toc419464004)

[4.1 Scenario 1 – Only Kit-lines 5](#_Toc419464005)

[4.2 Scenario 2 – Transfer kits through the intercompany chain 8](#_Toc419464006)

# Introduction

Kitting in an intercompany scenario have been defined as a business requirement at several users of the kitting solution. This document is evaluating the readyness of the kitting solution in a intercompany scenario.

# Prerequisites

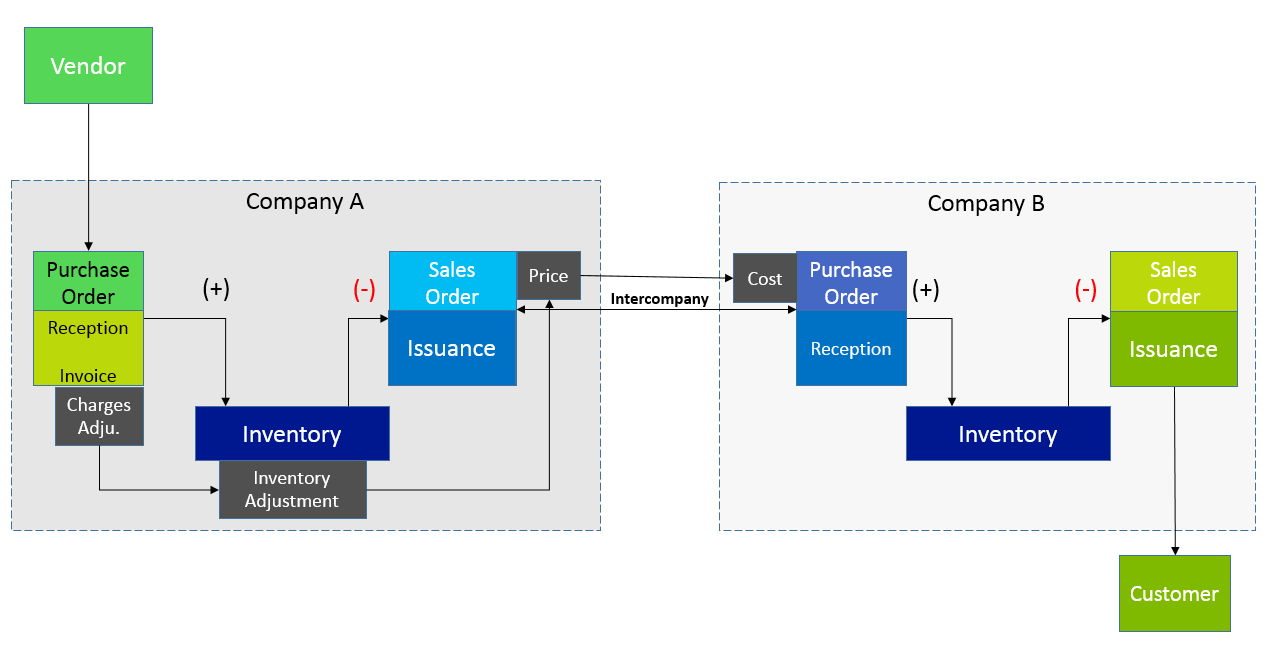
Before installing KIT for Dynamics AX 2012 R3, please ensure that the system meets or exceeds the following Dynamics AX builds.

|  |  |
| --- | --- |
|  | Application build |
| DAX 2012 R3 CU8 | 6.3.1000.309 |
| Kitting solution tested | Build 43 |

# Scenario

The intercompany transactions are a common business practice where companies trading goods between each other. There is company sells the goods, and a company receives the goods.

The scenario assume that there are two sister companies; company A represents the company which purchase goods from an external vendor, and these goods are sold to its sister company B, these goods will be sold for an external customer the two sister companies are agreed that the sales and cost price of the intercompany transactions should be equal. Assume that the item cost in company A has changed after issuing the goods to the sister company B, by an extra charges (freight, transportation) and at the end of the month this changes will be reflected on the item cost after running the inventory recalculation/closing. This will be reflected on the intercompany-sales price on company A, and intercompany-purchase cost on company B.

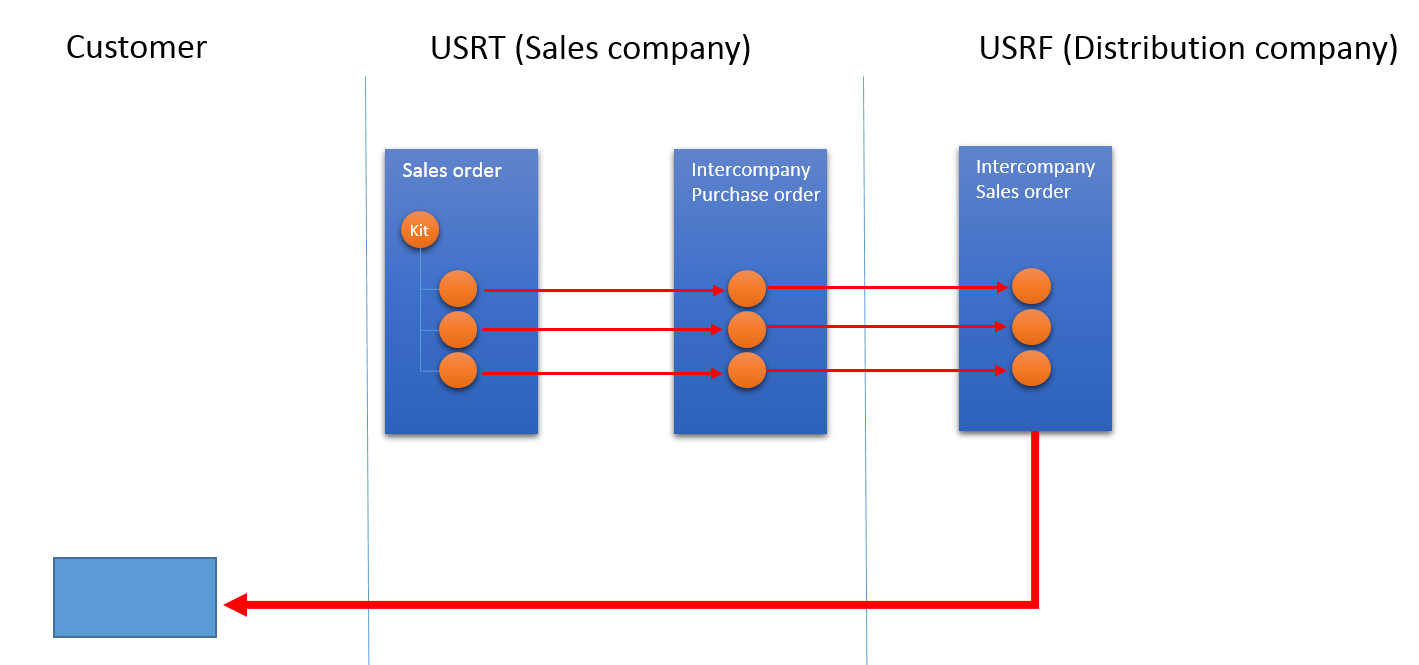


What is particulary interesting is when we have a direct delivery model in the intercompany scenario. Here we have several scenario's that should be tested.

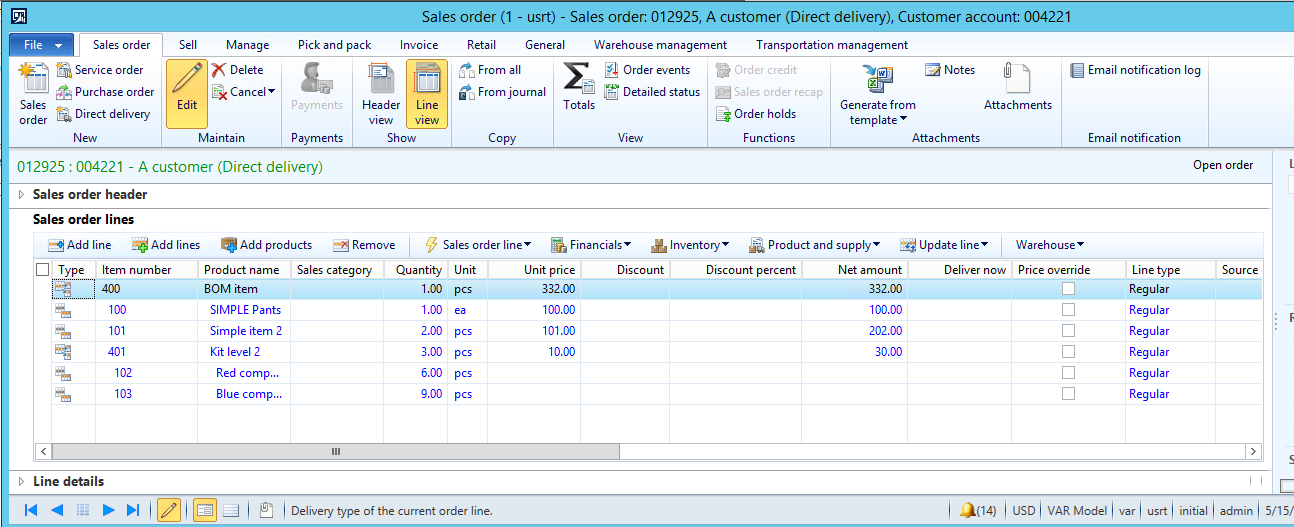
# Intercompany Scenario tested

## Scenario 1 – Only Kit-lines

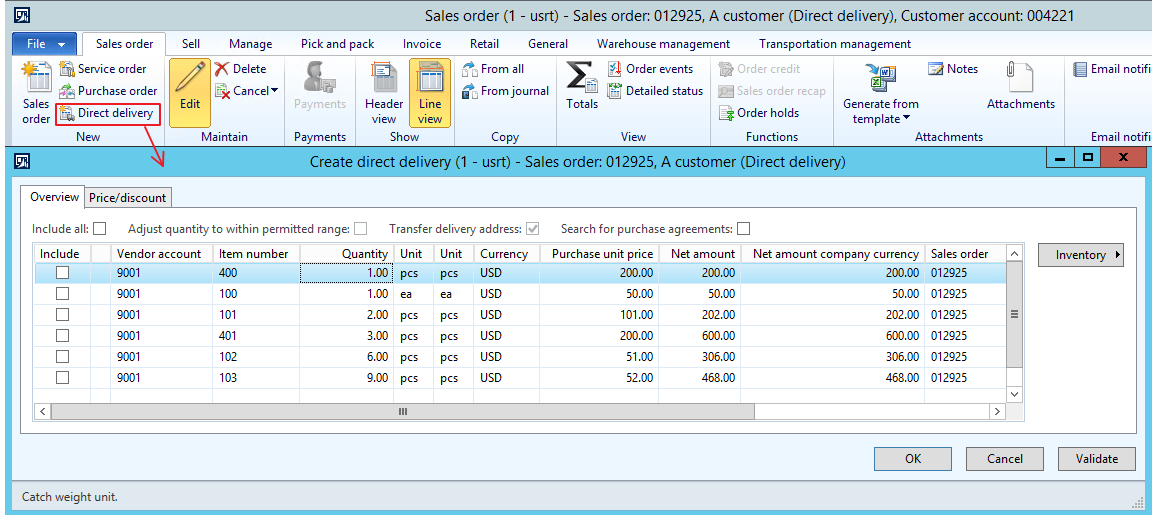
Kit exists in the sales company, but only lines are transferred through the intercompany chain.

SS

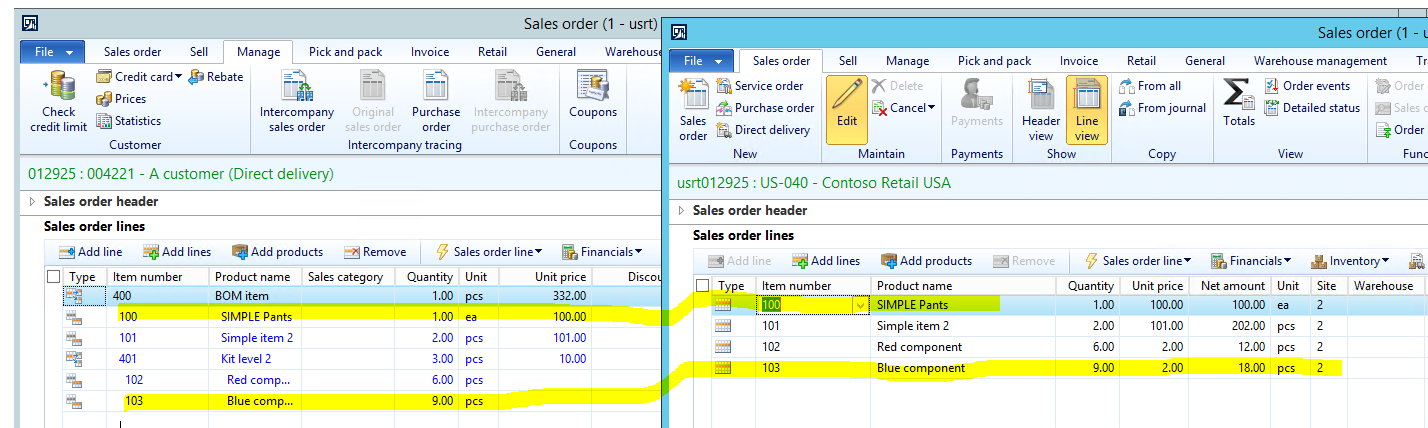
We then see the following sales order in USRT – Sales company;



When trying to create a direct delivery intercompany order, the following selection form can be used;

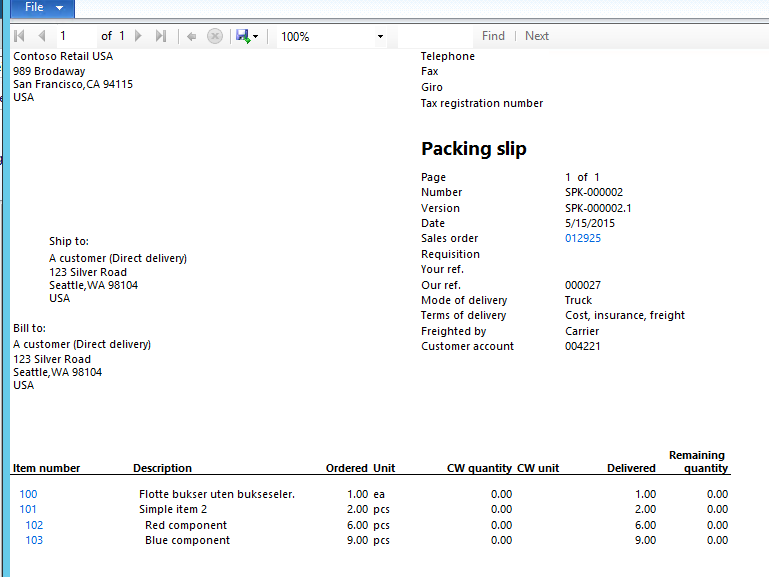


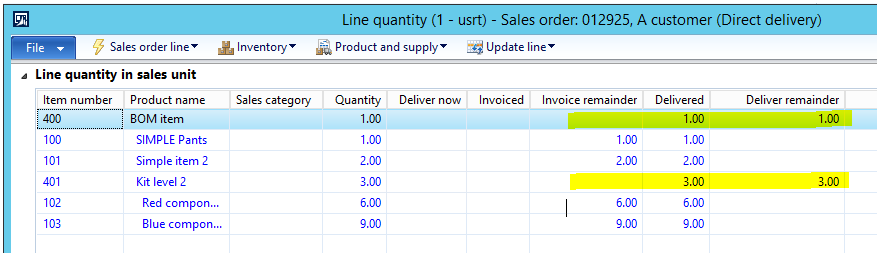
If I only approve to create the "pure item" (100,101,102,103), I see that the intercompany chain is created as expected.



**Picking of the intercompany transactions; - Passed**Validated, and works as expected. All lines are picked in USRF

**Packingslip of intercompany transactions; - Failed**When packingslip updating from USRF, we see that only the kit-lines have been packingslip updated. The Kit should also been packingslip updated. We are in this scenario missing the kit headers in the intercompany chain, since the kit only exists on the sales order in the sales company.

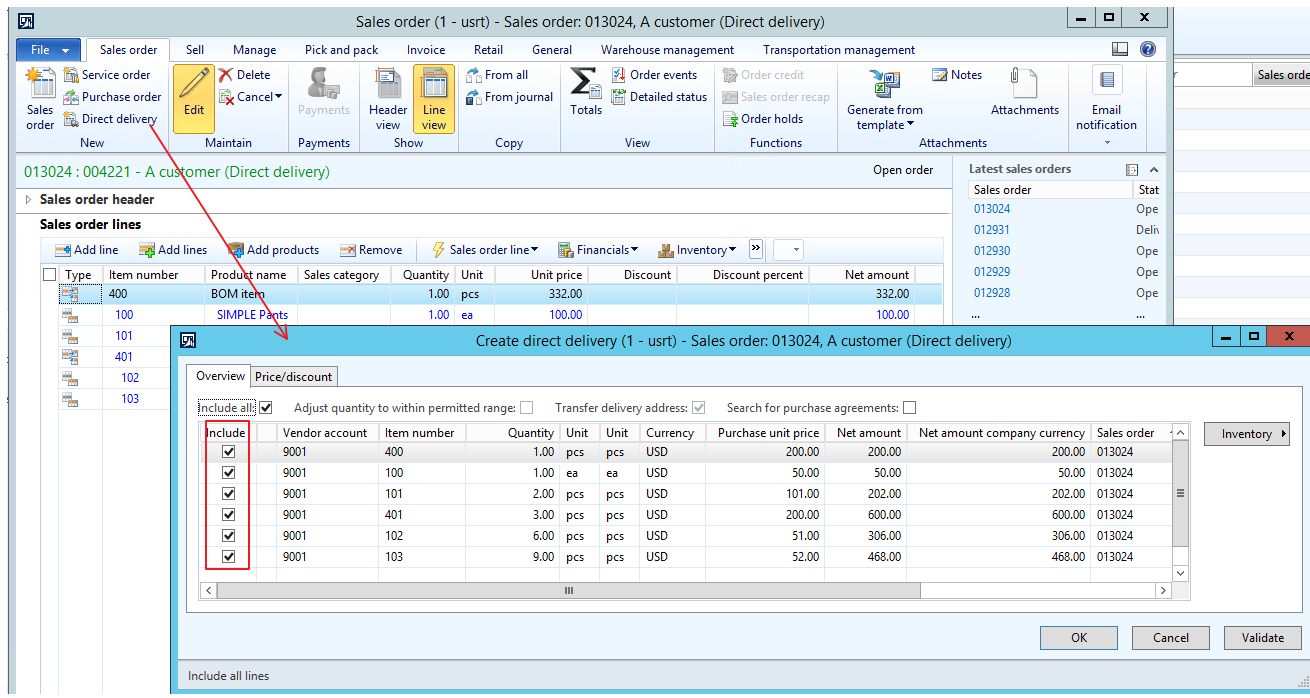
  
We are therefore also missing the correct delivery reminder after packingslip posting a kit in an intercompany scenaio.



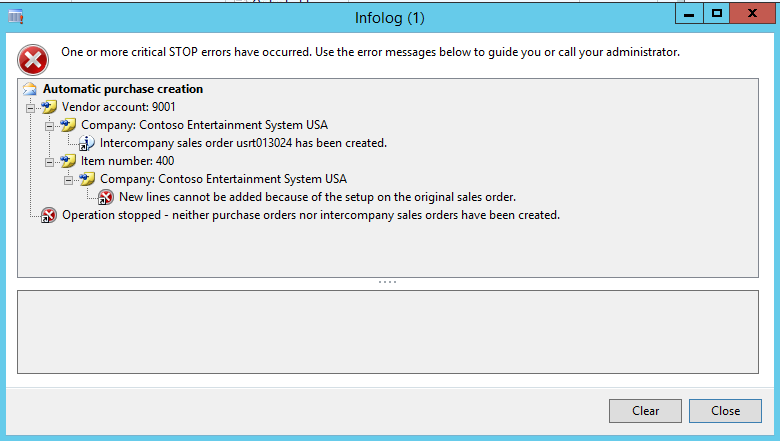
The problem have been solved with a Fix in Class SalesFormletterParmDataPackingslip.createParmLine() See appendix 1 for specific codes that have been added.

## Scenario 2 – Transfer kits through the intercompany chain

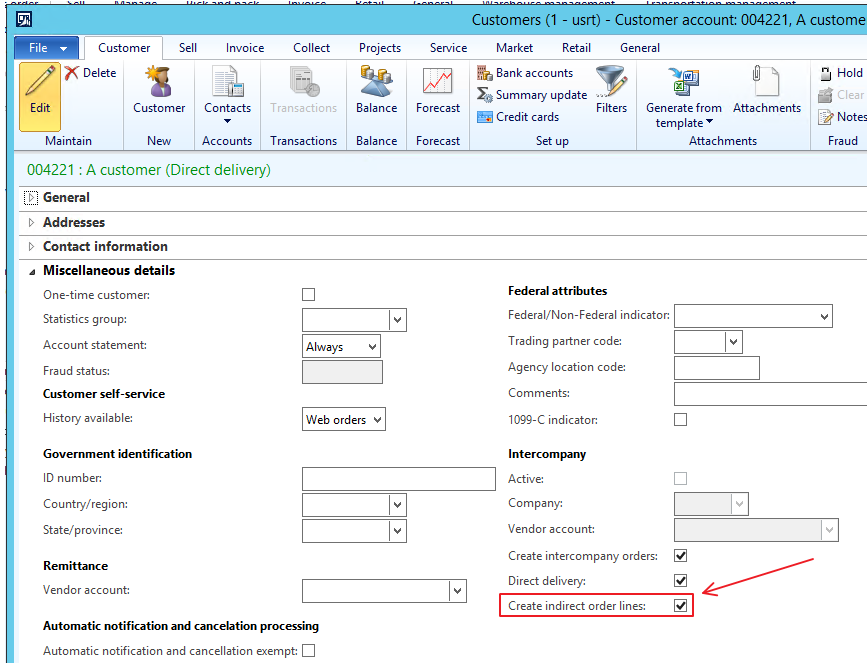
If we try to add all kit-lines like this ;



The AX solution will generate the following error ;



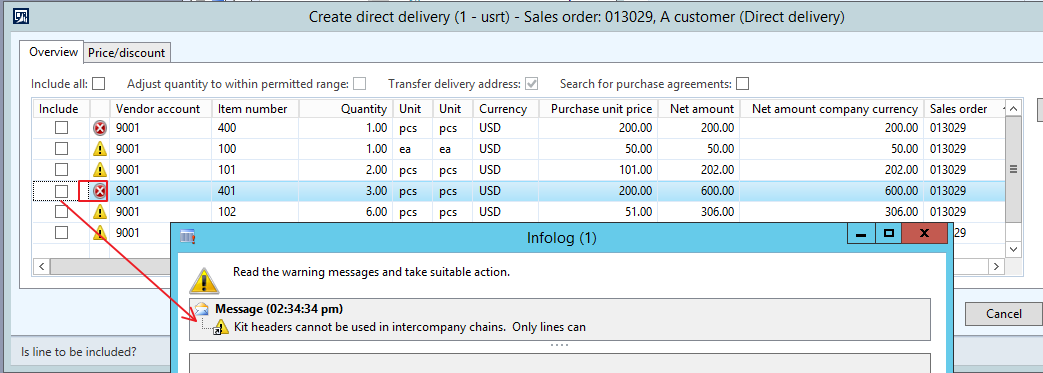
This error is related to allowing intercompany allow indirect creation of lines, and means that it is not allowed to add saleslines in the intercompany salesorder (in USFM), but only on the original sales company (USFT). This is a parameter in dynamics AX, and this parameter must be set on the customer record;



See Appendix 2 for the spesific code controlling this.

But kit headers are not transferred in a intercompany chain is, and will be quite a large change to make this happen. The reason for this is that it would imply that kits are supported on purchase orders also.

A possible solution is therefore to prevent that kit-headers can be transferred through the IC chain. Some symbols and messages have therefore been added to make this easier for the end user;



The code for this is attached in appendix 3

Appendix 1

**protected** **boolean** createParmLine(Common \_salesLineOrig,

TradeLineRefId \_tableRefId,

InventOwnerId\_RU \_inventOwnerId = '',

**boolean** \_mcrShipAlone = **false**)

{

**boolean** lineCreated = **false**;

SalesParmLine newSalesParmLine;

SalesQty shippingQty = -**1**;

InventQty inventShippingQty = -**1**;

***//>Kit***

**#KIT**

**#if.KitEnabled(true)**

**SalesLine salesLineOrig = \_salesLineOrig;**

**SalesLine salesLineParrent;**

**KitSalesLine kitSalesLine;**

**VendDocumentLineMap parmLine;**

**SalesQty salesQty ;**

**InventQty inventQty;**

**PdsCWInventQty pdsCWQty;**

**boolean isCwItem = pdsIsCWItem(salesLineOrig.ItemId);;**

**#endif**

***//<Kit***

**if**(\_mcrShipAlone)

{

**if** (\_salesLineOrig.RecId)

{

newSalesParmLine.clear();

[newSalesParmLine.DeliverNow, newSalesParmLine.RemainBefore , newSalesParmLine.RemainAfter ] = this.qtySales (\_salesLineOrig, naReal());

[newSalesParmLine.InventNow , newSalesParmLine.RemainBeforeInvent, newSalesParmLine.RemainAfterInvent] = this.qtyInvent (\_salesLineOrig, naReal());

**if** (newSalesParmLine.DeliverNow)

{

newSalesParmLine.ParmId = salesParmUpdate.ParmId;

newSalesParmLine.initFromSalesLine(\_salesLineOrig);

newSalesParmLine.setLineAmount();

newSalesParmLine.calcPackingUnitQty();

newSalesParmLine.TableRefId = \_tableRefId;

*//Set ship alone fields and insert immediately*

this.mcrSetShipAloneQuantities(newSalesParmLine, \_salesLineOrig);

newSalesParmLine.setLineAmount();

this.insertParmLine(newSalesParmLine);

lineCreated = **true**;

this.allowCreateParmTable(**true**);

}

}

}

**else**

{

lineCreated = **super**(\_salesLineOrig, \_tableRefId, \_inventOwnerId, \_mcrShipAlone);

}

***//>Kit***

**#KIT**

**#if.KitEnabled(true)**

**if (!kittingParrentTransIDSet)**

**kittingParrentTransIDSet = new Set(Types::String);**

**kitSalesLine = KitSalesLine::find(salesLineOrig.InventTransId);**

**if (kitSalesLine.KitParrentInventTransId && !kittingParrentTransIDSet.in(kitSalesLine.KitParrentInventTransId))**

**{**

**salesLineParrent = SalesLine::findInventTransId(kitSalesLine.KitParrentInventTransId);**

**if (this.parmInterCompanyParmId()**

**&& !this.parmInterCompanyParmSelectFromJournal())**

**{**

**parmLine = this.interCompanyParmLine(salesLineOrig);**

**salesQty = this.interCompanyParmLineQty(\_salesLineOrig);**

**inventQty = this.interCompanyParmLineQty(\_salesLineOrig, true);**

**if (isCwItem)**

**{**

**pdsCWQty = this.pdsCWInterCompanyParmLineQty(\_salesLineOrig);**

**}**

**}**

**else if ((salesLineOrig.DeliveryType == TradeLineDlvType::DropShip) && this.parmParmIdSynched())**

**{**

**parmLine = this.findParmLine(this.parmParmIdSynched(), salesLineOrig.InventRefTransId);**

**salesQty = parmLine.ReceiveNow;**

**inventQty = parmLine.InventNow;**

**pdsCWQty = parmLine.PdsCWReceiveNow;**

**}**

**if (inventQty)**

**{**

**if (salesLineOrig.SalesQty == 0)**

**inventQty = 0;**

**else**

**inventQty = inventQty \* salesLineParrent.SalesQty/salesLineOrig.SalesQty;**

**if (salesLineOrig.PdsCWQty == 0)**

**pdsCWQty = 0;**

**else**

**pdsCWQty = pdsCWQty \* salesLineParrent.PdsCWQty/salesLineOrig.PdsCWQty;**

**newSalesParmLine.clear();**

**salesLineParrent.InventDeliverNow = inventQty;**

**salesLineParrent.PdsCWInventDeliverNow = pdsCWQty;**

**salesLineParrent.SalesDeliverNow = UnitOfMeasureConverter::convert(inventQty,**

**UnitOfMeasure::unitOfMeasureIdBySymbol(salesLineParrent.inventTable().inventUnitId()),**

**UnitOfMeasure::unitOfMeasureIdBySymbol(salesLineParrent.SalesUnit),**

**NoYes::Yes,**

**InventTable::itemProduct(salesLineParrent.ItemId));**

**[newSalesParmLine.DeliverNow, newSalesParmLine.RemainBefore , newSalesParmLine.RemainAfter ] = this.qtySales (salesLineParrent, naReal());**

**[newSalesParmLine.InventNow , newSalesParmLine.RemainBeforeInvent, newSalesParmLine.RemainAfterInvent] = this.qtyInvent (salesLineParrent, naReal());**

**if (newSalesParmLine.DeliverNow)**

**{**

**newSalesParmLine.ParmId = salesParmUpdate.ParmId;**

**newSalesParmLine.initFromSalesLine(salesLineParrent);**

**newSalesParmLine.setLineAmount();**

**newSalesParmLine.calcPackingUnitQty();**

**newSalesParmLine.TableRefId = \_tableRefId;**

**newSalesParmLine.setLineAmount();**

**this.insertParmLine(newSalesParmLine);**

**lineCreated = true;**

**this.allowCreateParmTable(true);**

**}**

**}**

**kittingParrentTransIDSet.add(kitSalesLine.KitParrentInventTransId);**

**}**

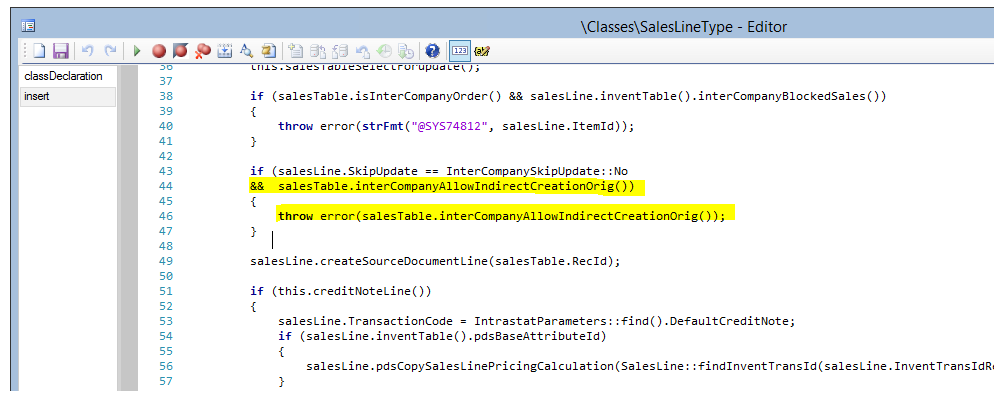
**#endif**

***//<Kit***

**return** lineCreated;

}

Appendix 2



Appendix 3

