|  |  |
| --- | --- |
|  |  |

Technical Specification Document



Project Name: GWC\_SalesOrderHandler\_Service

Issue Title: Process Sales Order based on predetermined process type.

Date: May 2023

# Prepared By

|  |  |
| --- | --- |
| **Document Owner(s)** | **Project/Organization Role** |
| Justin Pope | Software Developer |

# Modification Version Control

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Issue. Version** | **Date** | **Author** | **Tag ID** | **Change Description** |
| 1 | 05/19/2022 | Justin Pope | SDM 28959 | Initial Creation |
| 2 | 05/08/2023 | Justin Pope | SDM 32322 | SCT / PO Automation |

Contents

[Prepared By 2](#_Toc127266457)

[Modification Version Control 2](#_Toc127266458)

[2 Design Overview 4](#_Toc127266459)

[1.1 Business Case/Scope 4](#_Toc127266460)

[1.2 Audience 5](#_Toc127266461)

[1.3 Reference 5](#_Toc127266462)

[1.4 Terms and Definitions 5](#_Toc127266463)

[1.5 Technical Design Diagram 6](#_Toc127266464)

[1.5.1 Application Architecture Diagram 6](#_Toc127266465)

[1.5.2 Integration Architecture Diagram 8](#_Toc127266466)

[1.6 Assumptions/Dependencies 11](#_Toc127266467)

[1.7 Outstanding Questions /Open issues 11](#_Toc127266468)

[3 Data Design 11](#_Toc127266469)

[2.1 Database Management System Files 11](#_Toc127266470)

[3.2 Non-Database Management System Files 32](#_Toc127266471)

[4 Detailed Design 32](#_Toc127266472)

[3.1 Hardware Detailed Design 32](#_Toc127266473)

[3.2 Software Detailed Design 32](#_Toc127266474)

[3.2.1 Application Code Organization 32](#_Toc127266475)

[5 External Interface Design 37](#_Toc127266476)

[4.1 Interface Architecture 37](#_Toc127266477)

[4.2 Interface detailed design 37](#_Toc127266478)

[6 Human-Machine Interface 37](#_Toc127266479)

[5.1 Interface Design Rules 37](#_Toc127266480)

[5.1.1 Inputs 37](#_Toc127266481)

[7 Appendices 38](#_Toc127266482)

[Appendix A: Other supporting documentation 38](#_Toc127266483)

# Design Overview

This application is a windows service to run on 7Syspro. This service will be set to run continuously when Syspro is not in Black-out. Upon start, the service will set up the necessary settings to access the SQL database, Syspro Business Objects, email objects and configure the application timer to five minutes. When the timer has elapsed, the application will query staged orders and perform the type of process that has been indicated for them. If the order was processed successfully, the application will update the current record as processed. If the process resulted in an error, the application would update the record that it errored as well as send emails out based on what kind of error had happened.

The following types of records can be processed:

* Dispatch Processing:   
  The application can process Orders and corresponding Dispatch Tickets to dispatch charges (freight and/or Surcharges) on the order. The Application will query for items on the Sales Order and items on the Dispatch Ticket. If the whole order has been dispatched, all charges attached to the Order will be placed on the ticket. If there are items that remain on the Sales Order, adjust the charges based on what is being dispatched and add new charges to the sales order for what remains.
* Back Order Processing:  
  The application can process Orders with items in a backordered status. The application will determine for the order and each item whether a Supply Chain Transfer (SCT) or Purchase Order (PO) will be needed to fulfill them and create each one. If successful, the application will notify the correct users as for the creation of each SCT and PO.

The application will be built on the .Net Framework as a Service application. The service will utilize the “SYSPROWCFServicesClientLibrary40” as reference. The following business objects will be used for api calls to Syspro to create or maintain Sales Orders, Supply Chain Transfers and Purchase Orders: SORTOI, POROI, SORTTR. The application will utilize SQL stored procedures on the SQL08 server SysproDocument database.

## 1.1 Business Case/Scope

The application is built to be able to handle multiple processes. The following issues have occurred:

**SDM 28959 – Split Ship for Surcharge/Freight**

Issue

Gabriella White is increasing shipping split orders to resolve orders that are not completely ready to ship. Our current process places entire amount of Surcharge and Freight onto the first dispatch which our customers are unsatisfied with paying up front.

Solution

The application will be able to process Sales Orders and corresponding Dispatch Notes by adding appropriate amount onto a Dispatch Note while Backordering the remaining charge to the Sales Order.

**SDM 32322 – SCT / PO Automation for Retail**

Issue

Gabriella White is looking to automate the Back Order review process. The current process takes up a lot of the user’s time between ordering SCTs, creating POs, and updating the original order.

Solution

The application will be modified with a new module to automatically create the SCTs and POs needed for the order and update the order correctly.

## 1.2 Audience

This is a technical document targeted for technical readers that will maintain the designed application. It is assumed the reader has some technical background related to the technologies mentioned in the above description.

## 1.3 Reference

List all references that are included in the document. Also include applicable policies and procedures associated with the document and its contents.

| Ref# | Document Name | URL |
| --- | --- | --- |
| 1 | Application diagram |  |
| 2 | Syspro API xmls |  |

## 1.4 Terms and Definitions

| Term or Acronym | Definition |
| --- | --- |
| SCT | Supply Chain Transfer |
| PO | Purchase Order |

## 1.5 Technical Design Diagram

### 1.5.1 Application Architecture Diagram

**Staging Process**

A screenshot of a computer

Description automatically generated with low confidence

The staging process depends a trigger and procedure set up in the SQL Server SQL08. The processes operate as follows:

*Dispatch*

trg\_AdmSignature\_Log\_625100\_AfterUpdate inserts records for Dispatch Notes and Sales Orders to be processed for dispatching charges on the order. The record that is inserted is sets the SalesOrder column to the SalesOrder from the MdnMaster record, the ProcessType to 0 for the Dispatch process within the program, and the OptionalParm1 to the DispatchNoteNumber from the MdnMaster record.

*Backorder*

Usp\_Stage\_SalesOrders\_For\_BackOrder inserts a staged record for Sales Orders to process items that are in Back Order. The procedure confirms the following:

1. The sales order has a deposit to cover a percentage of the total that is determined by the credit terms.
2. The sales order is a retail order that is in an Open status and not a SCT.
3. The sales order does not have any line items with temporary stock codes and there are line items that are to be special ordered.

**Main Service Loop**

**A picture containing text, diagram, screenshot, plan

Description automatically generated**

The main service, upon start up, will set the service timers that are determined by the services settings. Upon the service timer elapsing, the main loop will begin. The services will pull down staged records that are ready to be processed, loop through each one and initiating the process defined on the record. Once the process is finished, the staged record will be update based on the outcome of the process. Once the staging timer elapses, the staging procedure is executed. Currently only orders for the Backorder process are staged in this way.

**Dispatch Process**

**Diagram

Description automatically generated**

The dispatch process handles each action that is need for the Sales Order. The procedure that will load the charge will inform the application on what is needed for the Order. The following actions are returned from the procedure:

* Dispatch: Per charge (Freight and Surcharge), there will be a row that will correspond to the insert into the dispatch note. Information on the charge value will be returned to the application for insert.
* SORTOI: The procedure will wrap up all charges that are to be back order to the Sales Order. The application will then call the SysPro API.

**Backorder Process**

A picture containing text, diagram, screenshot, parallel

Description automatically generated

The backorder process handles items that are backordered on a particular Sales Order and creates Supply Chain Transfers or Purchase Orders for them. The process validates all Syspro business objects (SORTOI and PORTOI) that are created. If validation finds issues, the store managers of the retail order will be notified via emails of issues found. If no issues occur, the process will transmit the objects, execute necessary updates that are not capable within the business objects, created acknowledgements for each SCT and PO that is created, and email the store manager with the successful outcome.

### 1.5.2 Integration Architecture Diagram

1. SysPro Business Objects
   1. SORTIO – Sales Order Import
      1. Parameters  
         Text, timeline

         Description automatically generated
      2. SORTIODOC  
         Text

         Description automatically generated with medium confidence
   2. PORTOI – Add, Change a Purchase Order
      1. Parameters
      2. PORTOIDOC
   3. SORTTR – Sales Order Supply Chain Transfer
   4. COMSFM – Custom Form Setup

## 1.6 Assumptions/Dependencies

1. 7Syspro server is available with network connectivity
2. SQL08 server is available with network connectivity

## 1.7 Outstanding Questions /Open issues

# Data Design

## Database Management System Files

* Data Connection String
  + InitialCatalog: SysproDocument
  + IntegeratedSecurity: SSPI
  + PersistSecurityInfo: true
* Tables / Added Records
  + Detailed information on the tables can be seen in the Project Table Structure document that can be found in Appendix C. The following is a list of tables names referenced in this project.
    - [SysproDocument].[dbo]
      * [Application]
      * [Setting]
      * [Log\_Event]
      * [Log\_Setting]
    - [SysproDocument].[SOH].[SorMaster\_Process\_Staged]
    - [SysproCompany100].[dbo]
      * [SorMaster]
      * [SorDetail
      * [MdnMaster]
      * [MdnDetail]
* Stored Procedures to be created
  + [SOH].[SorMaster\_Process\_Staged\_UPDATE]
    - Updates the SorMaster\_Process\_Staged record updating the Processed, ERROR, and LastChangedDateTime columns.
    - Parameters:
      * @ProcessNumber int
      * @Processed bit
      * ERROR bit
    - SQL Statement

Update SPS

set SPS.Processed = @Processed,

SPS.[ERROR] = @ERROR,

SPS.[LastChangedDateTime] = GETDATE()

from [SOH].[SorMaster\_Process\_Staged] SPS

where SPS.ProcessNumber = @ProcessNumber

* + [SOH].[SorMaster\_Process\_Staged\_GET]
    - Queries SorMasterProcess\_Staged records and returns records that have not been processed and have not errored.
    - Parameter: None
    - SQL Statement

Select

ProcessNumber,

[SalesOrder],

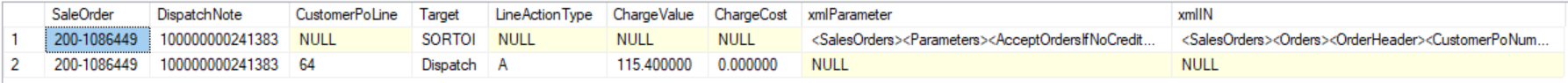
[ProcessType],

[OptionalParm1] as [DispatchNumber]

from [SOH].[SorMaster\_Process\_Staged]

where Processed = 0

and ERROR = 0

* + [SOH].[SalesOrderProcessCharges\_Get]
    - Calculates Sales Order Lines (SorDetail) with LineType in (4, 5) that need to be placed on the correlating Dispatch Note (MdnMaster) and add remaining charges on Back Order in the case of a Split Order. This procedure returns rows that represents each action the main service will need to take. The following is an example output:  
        
      From the above example, row 1 has the information that the application will use to call the Syspro API using the SORTOI Business Object. The following columns are important: Target, xmlParameter, smlIN. Row 2 has the information the application will use to call the Insert Procedure to insert Dispatch Note Lines (MdnDetail). The following columns are important: SaleOrder, DispatchNote, CustomerPoLine, Target, ChargeValue, ChargeCost.
    - Parameter:
      * @ProcessNumber int
    - SQL Statement

declare @LineActionAdd as Varchar(2) = 'A',

@LineActionChange as VArchar(2) = 'C',

@BackOrderPercent as Decimal(18,10),

@DispatchPercent as decimal(18,10),

@Dispatched as decimal(16,2),

@BackOrdered as decimal(16,2),

@SORTOIDOC\_XML as XML,

@SORTOI\_Name as varchar(10) = 'SORTOI',

@SOH\_ApplicationID as integer = (select applicationid from dbo.[Application] where ApplicationCode = 'SOH');

DROP TABLE IF EXISTS #SOH\_Charges

Select

@BackOrdered = [Calculated].[BackOrdered],

@Dispatched = [Calculated].[Dispatch]

from (

Select

SPS.SalesOrder as [SaleOrder],

sum(Total.[Value]) as [Total],

sum(BackOrdered.[Value]) as [BackOrdered],

sum(Dispatched.[Value]) as [Dispatch]

from [SysproDocument].[SOH].[SorMaster\_Process\_Staged] as SPS

left join (

select

SD.SalesOrder,

sum(CASE SD.[MDiscValFlag]

WHEN 'U' THEN ROUND(SD.MOrderQty\*(SD.[MPrice] - SD.[MDiscValue]),2)

WHEN 'V' THEN ROUND(SD.MOrderQty\*(((SD.[MOrderQty] \* SD.[MPrice]) - SD.[MDiscValue])/SD.MOrderQty),2)

ELSE ROUND(SD.MOrderQty\*(SD.[MPrice] \* (1 - SD.[MDiscPct1] / 100) \* (1 - SD.[MDiscPct2] / 100) \* (1 - SD.[MDiscPct3] / 100)),2)

END) as [Value]

from [SysproCompany100].[dbo].[SorDetail] as SD WITH (NOLOCK)

where LineType = 1

group by SalesOrder ) Total on Total.SalesOrder collate Latin1\_General\_Bin = SPS.SalesOrder

left join (

select

SD.SalesOrder,

sum(CASE SD.[MDiscValFlag]

WHEN 'U' THEN ROUND((SD.MShipQty+SD.MBackOrderQty+SD.QtyReserved)\*(SD.[MPrice] - SD.[MDiscValue]),2)

WHEN 'V' THEN ROUND((SD.MShipQty+SD.MBackOrderQty+SD.QtyReserved)\*(((SD.[MOrderQty] \* SD.[MPrice]) - SD.[MDiscValue])/SD.MOrderQty),2)

ELSE ROUND((SD.MShipQty+SD.MBackOrderQty+SD.QtyReserved)\*(SD.[MPrice] \* (1 - SD.[MDiscPct1] / 100) \* (1 - SD.[MDiscPct2] / 100) \* (1 - SD.[MDiscPct3] / 100)),2)

END) as [Value]

from [SysproCompany100].[dbo].[SorDetail] as SD WITH (NOLOCK)

where LineType = 1

group by SalesOrder ) BackOrdered on BackOrdered.SalesOrder collate Latin1\_General\_Bin = SPS.SalesOrder

left join (

Select

DispatchNote,

sum(TotalValue) as [Value]

from [SysproCompany100].dbo.MdnDetail WITH (NOLOCK)

group by DispatchNote ) as Dispatched on Dispatched.DispatchNote collate Latin1\_General\_BIN = SPS.[OptionalParm1]

where SPS.ProcessNumber = @ProcessNumber

group by SPS.SalesOrder) as [Calculated];

if (@BackOrdered + @Dispatched) > 0

begin

set @BackOrderPercent = @BackOrdered / (@BackOrdered + @Dispatched);

set @DispatchPercent = 1-@BackOrderPercent;

end

else

begin

set @BackOrderPercent = 0;

set @DispatchPercent = 0;

end

Select

SPS.SalesOrder,

Charges.[CustomerPoLine],

Charges.[LineAction],

Charges.[LineType],

Charges.[MiscChargeValue],

Charges.[MiscChargeCost]

into #SOH\_Charges

from [SysproDocument].[SOH].[SorMaster\_Process\_Staged] as SPS

left join (

--Charge Lines to Dispatch

select

SD.SalesOrder,

SD.SalesOrderLine as [CustomerPoLine],

@LineActionChange as [LineAction],

SD.LineType as [LineType],

Ceiling(SD.NMscChargeValue\*@DispatchPercent\*100)/100 as [MiscChargeValue],

Ceiling(SD.NMscChargeCost\*@DispatchPercent\*100)/100 as [MiscChargeCost]

from [SysproCompany100].dbo.SorDetail as SD WITH (NOLOCK)

where SD.LineType in (4,5)

and SD.[NMscInvCharge] <> 'I'

and @Dispatched > 0

union

--Charge Lines to Add

select

SalesOrder,

SD.SalesOrderLine as [CustomerPoLine],

@LineActionAdd as [LineAction],

SD.LineType as [LineType],

Floor(SD.NMscChargeValue\*@BackOrderPercent\*100)/100 as [MiscChargeValue],

Floor(SD.NMscChargeCost\*@BackOrderPercent\*100)/100 as [MiscChargeCost]

from [SysproCompany100].dbo.SorDetail as SD WITH (NOLOCK)

where SD.LineType in (4,5)

and SD.[NMscInvCharge] <> 'I'

and @BackOrderPercent > 0 ) as [Charges] on [Charges].[SalesOrder] collate Latin1\_General\_BIN = SPS.[SalesOrder]

left join [SysproCompany100].[dbo].[MdnDetail] as DD WITH (NOLOCK) on DD.SalesOrder collate Latin1\_General\_BIN = SPS.SalesOrder

and DD.SalesOrderLine = Charges.[CustomerPoLine]

where SPS.ProcessNumber = @ProcessNumber

and DD.SalesOrderLine is null;

if (select count(\*) from #SOH\_Charges where LineAction = @LineActionAdd) > 0

begin

SET @SORTOIDOC\_XML = (

Select

case

when Charges.[LineType] = 4

then [SOH].[svf\_Create\_SysPro\_BusObj\_SORTOIDOC\_FreightLine](Charges.SalesOrder,

Charges.LineAction,

Charges.CustomerPoLine,

Charges.MiscChargeValue,

Charges.MiscChargeCost)

when Charges.[LineType] = 5

then [SOH].[svf\_Create\_SysPro\_BusObj\_SORTOIDOC\_MiscChargeLine](Charges.SalesOrder,

Charges.LineAction,

Charges.CustomerPoLine,

Charges.MiscChargeValue,

Charges.MiscChargeCost)

end

From #SOH\_Charges as Charges

FOR XML path(''), ROOT('OrderDetails'));

SET @SORTOIDOC\_XML = (

Select

[SOH].[svf\_Create\_SysPro\_BusObj\_SORTOIDOC\_SalesOrderHeader](SPS.SalesOrder),

@SORTOIDOC\_XML

from [SysproDocument].[SOH].[SorMaster\_Process\_Staged] as SPS

where SPS.ProcessNumber = @ProcessNumber

FOR XML PATH('Orders'), ROOT('SalesOrders')

);

end

-- Sales Order Actions

Select

SPS.SalesOrder as [SaleOrder],

SPS.[OptionalParm1] as [DispatchNote],

null as [CustomerPoLine],

@SORTOI\_Name as [Target],

null as [LineActionType],

null as [ChargeValue],

null as [ChargeCost],

cast(P.ParameterDocument as varchar(max)) as [xmlParameter],

cast(@SORTOIDOC\_XML as varchar(max)) as [xmlIN]

from [SysproDocument].[SOH].[SorMaster\_Process\_Staged] as SPS

left join dbo.Parameter as P on p.ApplicationId = @SOH\_ApplicationID and p.ParameterDocumentName = @SORTOI\_Name

where SPS.ProcessNumber = @ProcessNumber

and @BackOrdered > 0

and @SORTOIDOC\_XML is not null

union

--Dispatch Actions

-- NOTE: will need to modify this return when we upgrade to Syspro V.8 and utilize Dispatch Note Business Object

Select

Charges.SalesOrder as [SaleOrder],

SPS.[OptionalParm1] as [DispatchNote],

Charges.CustomerPoLine as [CustomerPoLine],

'Dispatch' as [Target],

'A' as [LineActionType],

Charges.MiscChargeValue as [ChargeValue],

Charges.MiscChargeCost as [ChargeCost],

null as [xmlParameter],

null as [xmlIN]

from #SOH\_Charges as Charges

left join [SysproDocument].[SOH].[SorMaster\_Process\_Staged] as SPS on SPS.ProcessNumber = @ProcessNumber

where Charges.LineAction = @LineActionChange

and @Dispatched > 0

* + - Notes:
      * This procedure will need to be updated when Syspro7 is upgraded to Syspro8. Access to the Dispatch Note business object will mean that we can use the API instead of manually entering in the Dispatch Note Lines.
      * Any modifications to this procedure will result in the main service application to error when trying to process records.
  + [SOH].[MdnDetail\_INSERT]
    - This is the procedure that will be called to insert charges onto the Dispatch Note. This functionality was copied from [SysproCompany100].[dbo].[trg\_AdmSignatureLog\_625100\_AfterUpdate].
    - Parameters:
      * @DispatchNote NVARCHAR(15)
      * @SalesOrder NVARCHAR(15)
      * @LineNumber NVARCHAR(15)
      * @MiscChargeValue Decimal(14,2)
      * @MiscChargeCost Decimal(14,2)
    - Stored Procedure

DECLARE @Blank AS VARCHAR(1) = '',

@NextMdnDetLine AS DECIMAL(4, 0),

@No AS VARCHAR(1) = 'N',

@Zero AS TINYINT = 0;

SET @NextMdnDetLine = (SELECT MAX([DispatchNoteLine])

FROM SysproCompany100.dbo.MdnDetail WITH (NOLOCK)

WHERE [DispatchNote] = @DispatchNote);

INSERT INTO SysproCompany100.dbo.MdnDetail (

[DispatchNote]

,[DispatchNoteLine]

,[SalesOrder]

,[SalesOrderLine]

,[LineType]

,[DispatchStatus]

,[TotalValue]

,[OverUnderFlag]

,[StockDepleted]

,[ConfirmationDate]

,[ConfirmationLine]

,[OrigShipSoUom]

,[OrigShipStkUom]

,[OrigBoSoUom]

,[MStockCode]

,[MStockDes]

,[MWarehouse]

,[MBin]

,[MOrderQty]

,[MQtyToDispatch]

,[MBackOrderQty]

,[MUnitCost]

,[MBomFlag]

,[MParentKitType]

,[MQtyPer]

,[MScrapPercentage]

,[MPrintComponent]

,[MComponentSeq]

,[MQtyChangesFlag]

,[MOptionalFlag]

,[MDecimals]

,[MOrderUom]

,[MStockQtyToShp]

,[MStockingUom]

,[MConvFactOrdUm]

,[MMulDivPrcFct]

,[MPrice]

,[MPriceUom]

,[MCommissionCode]

,[MDiscPct1]

,[MDiscPct2]

,[MDiscPct3]

,[MDiscValFlag]

,[MDiscValue]

,[MProductClass]

,[MTaxCode]

,[MLineShipDate]

,[MAllocStatSched]

,[MFstTaxCode]

,[MStockUnitMass]

,[MStockUnitVol]

,[MPriceCode]

,[MConvFactAlloc]

,[MMulDivQtyFct]

,[MTraceableType]

,[MMpsFlag]

,[MPickingSlip]

,[MMovementReqd]

,[MSerialMethod]

,[MZeroQtyCrNote]

,[MAbcApplied]

,[MMpsGrossReqd]

,[MContract]

,[MBuyingGroup]

,[MCusSupStkCode]

,[MCusRetailPrice]

,[MTariffCode]

,[MLineReceiptDat]

,[MLeadTime]

,[MTrfCostMult]

,[MSupplementaryUn]

,[MReviewFlag]

,[MReviewStatus]

,[MInvoicePrinted]

,[MDelNotePrinted]

,[MOrdAckPrinted]

,[MHierarchyJob]

,[MCustRequestDat]

,[MLastDelNote]

,[MUserDef]

,[MQtyDispatched]

,[MDiscChanged]

,[MCreditOrderNo]

,[MCreditOrderLine]

,[MUnitQuantity]

,[MConvFactUnitQ]

,[MAltUomUnitQ]

,[MDecimalsUnitQ]

,[MEccFlag]

,[MVersion]

,[MRelease]

,[MCommitDate]

,[NComment]

,[NCommentFromLin]

,[NMscChargeValue]

,[NMscProductCls]

,[NMscChargeCost]

,[NMscInvCharge]

,[NCommentType]

,[NMscTaxCode]

,[NMscFstCode]

,[NCommentTextTyp]

,[NMscChargeQty]

,[NSrvIncTotal]

,[NSrvSummary]

,[NSrvChargeType]

,[NSrvParentLine]

,[NSrvUnitPrice]

,[NSrvUnitCost]

,[NSrvQtyFactor]

,[NSrvApplyFactor]

,[NSrvDecimalRnd]

,[NSrvDecRndFlag]

,[NSrvMinValue]

,[NSrvMaxValue]

,[NSrvMulDiv]

,[NPrtOnInv]

,[NPrtOnDel]

,[NPrtOnAck]

,[NTaxAmtFlag]

,[NDepRetFlagProj]

,[NRetentionJob]

,[NSrvMinQuantity]

,[NChargeCode]

,[TpmUsageFlag]

,[PromotionCode]

,[TpmSequence]

,[SalesOrderInitLine]

,[PreactorPriority]

,[SalesOrderDetStat]

,[JnlYear]

,[JnlMonth]

,[Journal]

,[JournalLine]

,[MaterialAllocLine]

,[ScrapQuantity]

,[FixedQtyPerFlag]

,[FixedQtyPer]

)

SELECT @DispatchNote AS [DispatchNote]

, ROW\_NUMBER() OVER

(ORDER BY [SalesOrder] ASC

,[SalesOrderLine] ASC)

+ @NextMdnDetLine AS [DispatchNoteLine]

,[SalesOrder] AS [SalesOrder]

,[SalesOrderLine] AS [SalesOrderLine]

,[LineType] AS [LineType]

,@Blank AS [DispatchStatus]

,@Zero AS [TotalValue]

,@Blank AS [OverUnderFlag]

,@Blank AS [StockDepleted]

,NULL AS [ConfiramtionDate]

,@Zero AS [ConfirmationLine]

,@Zero AS [OrigShipSoUom]

,@Zero AS [OrigShipStkUom]

,@Zero AS [OrigBoSoUom]

,@Blank AS [MStockCode]

,@Blank AS [MStockDes]

,@Blank AS [MWarehouse]

,@Blank AS [MBin]

,@Zero AS [MOrderQty]

,@Zero AS [MQtyToDispatch]

,@Zero AS [MBAckOrderQty]

,@Zero AS [MUnitCost]

,@Blank AS [MBomFlag]

,@Blank AS [MParentKitType]

,@Zero AS [MQtyPer]

,@Zero AS [MScrapPercentage]

,@Blank AS [MPrintComponent]

,@Blank AS [MComponentSeq]

,@Blank AS [MQtyChangesFlag]

,@Blank AS [MOptionalFlag]

,@Zero AS [Decimals]

,@Blank AS [MOrderUom]

,@Zero AS [MStockQtyToShp]

,@Blank AS [MStockingUom]

,@Zero AS [MConvFactOrdUm]

,@Blank AS [MMulDicPrcFct]

,@Zero AS [MPrice]

,@Blank AS [MPriceUom]

,@Blank AS [MCommissionCode]

,@Zero AS [MDiscPct1]

,@Zero AS [MDiscPct2]

,@Zero AS [MDicsPct3]

,@Blank AS [MDicsValFlag]

,@Zero AS [MDicsValue]

,@Blank AS [MProductClass]

,@Blank AS [MTaxCode]

,NULL AS [MLineShipDate]

,@Blank AS [MAllocStatSched]

,@Blank AS [MFstTaxCode]

,@Zero AS [MStockUnitMass]

,@Zero AS [MStockUnitVol]

,@Blank AS [MPriceCode]

,@Zero AS [MConvFactAlloc]

,@Blank AS [MMulDivQtyFct]

,@Blank AS [MTraceableType]

,@Blank AS [NMpsFlag]

,@Blank AS [MPickingSlip]

,@Blank AS [MMovementReq]

,@Blank AS [MSerialMethod]

,@Blank AS [MZeroQtyCrNote]

,@Blank AS [MAbcApplied]

,@Blank AS [MMpsGrossReqd]

,@Blank AS [MContract]

,@Blank AS [MBuyingGroup]

,@Blank AS [MCusSupStkCode]

,@Zero AS [MCusRetailPrice]

,@Zero AS [MTariffCode]

,NULL AS [MLineReceiptDat]

,@Zero AS [MLeadTIme]

,@Zero AS [MTrfCostMult]

,@Blank AS [MSupplimentaryUn]

,@Blank AS [MReviewFlag]

,@Blank AS [MReviewStatus]

,@Blank AS [MInvoicePrinted]

,@Blank AS [MDelNotePrinted]

,@Blank AS [MOrdAckPrinted]

,@Blank AS [MHierarchyJob]

,NULL AS [MCustRequestDat]

,@Blank AS [MLastDelNote]

,@Blank AS [MUserDef]

,@Zero AS [MQtyDispatched]

,@Blank AS [MDiscChanged]

,@Blank AS [MCreditOrderNo]

,@Zero AS [MCrediteOrderLine]

,@Blank AS [MUnitQuantity]

,@Zero AS [MConvFactUnitQ]

,@Blank AS [MAltUomUnitQ]

,@Zero AS [MDecimalsUnitQ]

,@Blank AS [MEccFlag]

,@Blank AS [MVersion]

,@Blank AS [MRelease]

,NULL AS [MCommitDate]

,[NComment] AS [NComment]

,@Zero AS [NCommentFromLin]

,@MiscChargeValue AS [NMscChargeValue]

,[NMscProductCls] AS [NMscProductCls]

,@MiscChargeCost AS [NMscChargeCost]

,@Blank AS [NMscInvCharge]

,@Blank AS [NCommentType]

,[NMscTaxCode] AS [NMscTaxCode]

,[NMscFstCode] AS [NMscFstCode]

,@Blank AS [NCommentTextTyp]

,[NMscChargeQty] AS [NMscChargeQty]

,[NSrvIncTotal] AS [NSrvIncTotal]

,[NSrvSummary] AS [NSrvSummary]

,[NSrvChargeType] AS [NSrvChargeType]

,[NSrvParentLine] AS [NSrvParentLine]

,[NSrvUnitPrice] AS [NSrvUnitPrice]

,[NSrvUnitCost] AS [NSrvUnitCost]

,[NSrvQtyFactor] AS [NSrvQtyFactor]

,[NSrvApplyFactor] AS [NSrvApplyFactor]

,[NSrvDecimalRnd] AS [NSrvDecimalRnd]

,[NSrvDecRndFlag] AS [NSrvDecRndFlag]

,[NSrvMinValue] AS [NSrvMinValue]

,[NSrvMaxValue] AS [NSrvMaxValue]

,[NSrvMulDiv] AS [NSrvMulDiv]

,[NPrtOnInv] AS [NPrtOnInv]

,[NPrtOnDel] AS [NPrtOnDel]

,[NPrtOnAck] AS [NPrtOnAck]

,[NTaxAmountFlag] AS [NTaxAmountFlag]

,[NDepRetFlagProj] AS [NDepRetFlagProj]

,@Blank AS [NRetentionJob]

,[NSrvMinQuantity] AS [NSrvMinQuantity]

,[NChargeCode] AS [NChargeCode]

,@Blank AS [TpmUsageFlag]

,@Blank AS [PromotionCode]

,@Zero AS [TpmSequence]

,[SalesOrderInitLine] AS [SalesOrderInitLine]

,@Zero AS [PreactorPriority]

,@Blank AS [SalesOrderDetStat]

,@Zero AS [JnlYear]

,@Zero AS [JnlMonth]

,@Zero AS [Journal]

,@Zero AS [JournalLine]

,@Blank AS [MaterialAllocLine]

,@Zero AS [ScrapQuantity]

,@Blank AS [FixedQtyPerFlag]

,@Zero AS [FixedQtyPer]

FROM SysproCompany100.dbo.SorDetail

WHERE [SalesOrder] = @SalesOrder

AND [SalesOrderLine] = @LineNumber;

UPDATE SysproCompany100.dbo.MdnMaster

SET [NextDetailLine] = (SELECT MAX([DispatchNoteLine])

FROM SysproCompany100.dbo.MdnDetail WITH (NOLOCK)

WHERE [DispatchNote] = @DispatchNote)

,[ReprintFormat] = 'O'

,[Email] = (SELECT [Email]

FROM SysproCompany100.dbo.SorMaster WITH (NOLOCK)

WHERE [SalesOrder] = @SalesOrder)

WHERE [DispatchNote] = @DispatchNote;

* + [SOH].[usp\_SorMaster\_Process\_Staged\_Get]
    - Get the SorMaster\_Process\_Staged records that have not yet been processed.
    - Parameter: none
    - SQL Statement

SET NOCOUNT ON;

SELECT

[ProcessNumber],

[SalesOrder],

[ProcessType],

[OptionalParm1]

FROM [SOH].[SorMaster\_Process\_Staged]

WHERE [Processed] = 0

and [ERROR] = 0

ORDER BY [CreateDateTime];

* + - Notes
* Trigger that was Altered
  + [dbo].[trg\_AdmSignatureLog\_625100\_AfterUpdate]
    - This trigger has been updated to start inserting SorMaster\_Process\_Staged records. Snipping from lines 76 – 382.
    - SQL

Graphical user interface, text, application, email

Description automatically generated

* Scalar Value Functions to be created
  + [SOH].[svf\_Create\_SysPro\_BusObj\_SORTOIDOC\_SalesOrderHeader]
    - Returns the SalesOrderHeader XML for the SORTOI SORTOIDOC xml.
    - Parameter
      * @SalesOrder varchar(20)
    - SQL Statement  
        
      return (

Select

SM.CustomerPoNumber as [CustomerPoNumber]

, 'C' as [OrderActionType]

, '' as [NewCustomerPoNumber]

, '' as [Supplier]

, sm.Customer as [Customer]

, CONVERT(varchar, sm.OrderDate, 23) as [OrderDate]

--, sm.ShippingInstrs as [ShippingInstrs]

--, sm.ShippingInstrsCod as [ShippingInstrsCode]

, sm.CustomerName as [CustomerName]

--, sm.ShipAddress1 as [ShipAddress1]

--, sm.ShipAddress2 as [ShipAddress2]

--, sm.ShipAddress3 as [ShipAddress3]

--, sm.ShipAddress3Loc as [ShipAddress3Locality]

--, sm.ShipAddress4 as [ShipAddress4]

--, sm.ShipAddress5 as [ShipAddress5]

--, sm.ShipPostalCode as [ShipPostalCode]

--, sm.ShipToGpsLat as [ShipGpsLat]

--, sm.ShipToGpsLong as [ShipGpsLong]

--, sm.LanguageCode as [LanguageCode]

, iif(sm.Email is null, '', RTRIM(sm.Email)) as [Email]

--, '' as [OrderDiscPercent1]

--, '' as [OrderDiscPercent2]

--, '' as [OrderDiscPercent3]

, RTRIM(sm.Warehouse) as [Warehouse]

, RTRIM(sm.SpecialInstrs) as [SpecialInstrs]

, sm.SalesOrder as [SalesOrder]

, sm.OrderType as [OrderType]

--, '' as [MultiShipCode]

--, '' as [ShipAddressPerLine]

--, '' as [AlternateReference]

--, '' as [Salesperson]

--, '' as [Branch]

--, '' as [Area]

, convert(varchar, sm.ReqShipDate, 23) as [RequestedShipDate]

--, '' as [InvoiceNumberEntered]

--, '' as [InvoiceDateEntered]

--, '' as [OrderComments]

--, '' as [Nationality]

--, '' as [DeliveryTerms]

--, '' as [TransactionNature]

--, '' as [TransportMode]

--, '' as [ProcessFlag]

--,patindex(sm.TaxExemptNumber,@RegularExpr) as [TaxExemptNumber]

--,patindex(sm.TaxExemptFlag,@RegularExpr) as [TaxExemptionStatus]

--,patindex(sm.GstExemptNum,@RegularExpr) as [GstExemptNumber]

--,patindex(sm.GstExemptFlag ,@RegularExpr) as [GstExemptionStatus]

--, sm.CompanyTaxNo as [CompanyTaxNumber]

--, '' as [ShipAddressPerLineTax]

--, '' as [CancelReasonCode]

--, sm.DocumentFormat as [DocumentFormat]

--, sm.[State] as [State]

--, sm.CountyZip as [CountyZip]

--, '' as [City]

--, '' as [DeliveryRouteAction]

--, '' as [DeliveryRoute]

--, '' as [InvoiceWholeOrderOnly]

--, '' as [SalesOrderPromoQualifyAction]

--, '' as [SalesOrderPromoSelectAction]

--, '' as [GlobalTradePromotionCodes]

--, '' as [POSSalesOrder]

, '' as [eSignature]

from [SysproCompany100].[dbo].[SorMaster] as SM

where SM.SalesOrder = @SalesOrder

for xml path('OrderHeader'))

* + - Notes:
      * This Scalar value function is set up with tags from the library Syspro.  
        Apendix C: Syspro Business Objects
      * Commented columns in select statement should remain to keep the Business Object structure. It is not being implemented in its current application.
      * The regular expression is used to parse out non-alphanumeric characters within columns for the API Post.
  + [SOH].[svf\_Create\_SysPro\_BusObj\_SORTOIDOC\_MiscChargeLine]
    - Return the MiscChargeLine XML for the SORTOI SORTOIDOC xml.
    - Parameters
      * @SalesOrder varchar(20)
      * @LineAction varchar(2)
      * @LineNumber integer
      * @Charge decimal(14,2)
      * @Cost decimal(14,2)
    - SQL Statement

RETURN (

select

SD.SalesOrderLine as [CustomerPoLine],

@LineAction as [LineActionType],

--'' as [LineCancelCode],

@Charge as [MiscChargeValue],

@Cost as [MiscChargeCost],

sd.NMscChargeQty as [MiscQuantity],

--IIF(SD.NMscProductCls is null, '', PatIndex(SD.NMscProductCls, @RegularExpr)) as [MiscProductClass],

IIF(SD.NMscTaxCode is null, '',RTRIM(SD.NMscTaxCode)) as [MiscTaxCode],

--'' as [MiscNotTaxable],

IIF(SD.NMscFstCode is null, '',RTRIM(SD.NMscFstCode)) as [MiscFstCode],

--'' as [MiscNotFstTaxable],

IIF(SD.NComment is null, '',RTRIM(SD.NComment)) as [MiscDescription],

IIF(SD.NChargeCode is null, '',RTRIM(SD.NChargeCode)) as [MiscChargeCode],

IIF(SD.MTariffCode is null, '',RTRIM(SD.MTariffCode)) as [MiscTariffCode]--,

--'' as [ConfigPrintInv],

--'' as [ConfigPrintDel],

--'' as [ConfigPrintAck]

from [SysproCompany100].[dbo].[SorDetail] as SD

where SD.SalesOrder = @SalesOrder

and SD.SalesOrderLine = @LineNumber

* + - FOR XML PATH('MiscChargeLine'));
    - Notes:
      * This Scalar value function is set up with tags from the library Syspro.  
        Apendix C: Syspro Business Objects
      * Commented columns in select statement should remain to keep the Business Object structure. It is not being implemented in its current application.
      * The regular expression is used to parse out non-alphanumeric characters within columns for the API Post.
  + [SOH].[svf\_Create\_SysPro\_BusObj\_SOROIDOC\_FreightLine]
    - Return the FreightLine xml for the SORTOI SORTOIDOC xml.
    - Parameters
      * @SalesOrder varchar(20)
      * @LineAction varchar(2)
      * @LineNumber integer
      * @Charge decimal(14,2)
      * @Cost decimal(14,2)
    - SQL Statement  
        
       RETURN (

select

SD.SalesOrderLine as [CustomerPoLine],

@LineAction as [LineActionType],

--'' as [LineCancelCode],

@Charge as [FreightValue],

@Cost as [FreightCost],

iif(SD.NMscTaxCode is null, '', RTRIM(SD.NMscTaxCode)) as [FreightTaxCode],

--'' as [FreightNotTaxable],

iif(SD.NMscFstCode is null, '', RTRIM(SD.NMscFstCode)) as [FreightFstCode]--,

--'' as [FreightNotFstTaxable]

from [SysproCompany100].[dbo].[SorDetail] as SD

where SD.SalesOrder = @SalesOrder

and SD.SalesOrderLine = @LineNumber

FOR XML PATH('FreightLine'));

* + - Notes:
      * This Scalar value function is set up with tags from the library Syspro.  
        Apendix C: Syspro Business Objects
      * Commented columns in select statement should remain to keep the Business Object structure. It is not being implemented in its current application.
      * The regular expression is used to parse out non-alphanumeric characters within columns for the API Post.

## Non-Database Management System Files

# Detailed Design

## 3.1 Hardware Detailed Design

## 3.2 Software Detailed Design

### 3.2.1 Application Code Organization

1. SOH\_Service.vb  
   This is the main service module of the application. It will process the main process and regulate how often to start a process.
   1. Variables / Objects
      1. tmrPrimary as Timer
   2. **OnStart** Subroutine
      1. Sets up need parameters and object by calling modMain.GetAppSettings.
      2. Write to log that the service has started.
      3. Sets the interval and activates tmrPrimary. Timer is set to every 5 minutes.
   3. **OnStop** Subroutine
      1. Writes to log that the service has stopped.
      2. Send error email by calling modMain.SendErrorEmail
   4. **tmrPrimary\_Elasped** subroutine
      1. Handles the event from frmPrimary.Elapsed.
      2. Checks if Syspro is in Black Out by calling InBlackOutTime.
      3. Deactivates timer to process staged records.
      4. Calls ProcessOrders
      5. Reactivates timer before the end of the subroutine.
   5. **ProcessOrders** subroutine
      1. Set up local variables
      2. Get orders in the form of a DataTabe by calling DataContext.GetStagedOrders
      3. Begin For Each DataRow from the returned DataTable
         1. Select a Case for the type of the record. This is from the column name “ProcessType” on the DataRow.
            1. Case modMain.OrderProcesses.Dispatch

Process record using ProcessDispatch

* + - * 1. Case Else

Throw an error that this record has and incorrect typing

* + - 1. Catch Exception
         1. Log that there is an error
      2. Update the staged record by calling DataContext.UpdateStagedOrder
         1. This will update the record if it has been processed or that there is an error
      3. If an error was cased call modMain.SendFailureEmail with exception message
  1. **InBlackOutTime** Function
     1. Returns a Boolean
     2. Calls and returns output from SysProHandler.CheckIfCompanyLocked

1. **DataContext.vb**
   1. **GetApplicationSettings** Function
      1. Calls the GEN.usp\_Settings\_Get procedure for the application’s settings.
   2. **GetStagedOrders** Function
   3. **UpdateStagedOrder** Subroutine
   4. **GetProcessedCharges** Function
   5. **InsertMdnDetails** Subroutine
2. **modMain.vb**
   1. **GetAppSettings** Function
      1. **Returns**
         1. **String**
      2. This Function sets up all parameters for the service.
         1. SQL Connection
         2. WebServices
         3. SysPro Logon credentials
         4. Timer Interval
         5. Sets up the following Email Objects
            1. Success
            2. Failure
            3. Error
         6. Returns “”
         7. Catches all exceptions and returns the error message from the exception
   2. **SetUpEmailObject** Subroutine
      1. **Parameters**
         1. **pEMail** as GWC.Mail.Message
         2. **pData** as XElement
         3. **pType** as String
      2. The subroutine knows how to set up the pEMail parameters by parsing the pData.
   3. **SendFailureEmail** Subroutine
      1. **Parameters**
         1. **drProcessRow** as DataRow
         2. **ErrorMessage** as string
      2. This subroutine will create a failure email message with the parameters provided and add it to the GWC email queue.
   4. **SendErrorEmail** Subroutine
      1. **Parameters**
         1. **drProcessRow** as DataRow
         2. **ErrorMessage** as string
      2. This subroutine will create an error email message with the parameters provided and add it to the GWC email queue.
   5. **HTMLEmail** Function
      1. Returns the html header that is used for all emails.
   6. **GetOrderProcessEnum** Function
      1. Will convert an integer to the OrderProcesses Enum. So far there is only one process. The enum will need to be expanded per process added.
   7. **GetORderPRocessENUMDescription** Function
      1. Will return a string description for each OrderProcesses Enum.
   8. **ConfigureDynamicLog** Subroutine
      1. Sets up the DynamicLogger object for the application.
3. **ProcessDispatch.vb**
   1. **Variables**
      1. **ProcessNumber** as integer
      2. **SalesOrder** as string
      3. **DispatchNumber** as string
      4. **dtCharges** as DataTable
   2. **New** Subroutine
      1. **Parameter**
         1. **pProcessRow** as Datatable
      2. For the parameter pProcessRow, the subroutine will set the variables ProcessNumber, SalesOrder, and DispatchNumber.
   3. **Process** Subroutine
      1. This is the only public subroutine for the Dispatch Process. Will call LoadCharges and then ProcessCharges.
   4. **LoadCharges** Subroutine
      1. **Parameter**
         1. **pProcessNumber** as Integer
      2. Sets dtCharges to the output of DataContext.GetProcessedCharges with the parameter pProcessNumber.
   5. **ProcessCharges** Subroutine
      1. For each row in dtCharges, the subroutine will do the following:
         1. For drCharge(“Target”) = “Dispatch”, call CreateDispatchItem with the datarow
         2. For drCharge(“Target”) = “SORTOI”, call CallSORTIO with the datarow
   6. **CreateDispatchItem** Subroutine
      1. **Parameter**
         1. **drCharge** as DataRow
      2. Calls DataContext.InsertMdnDetail with data from drCharge
      3. Catches exceptions thrown and will log them with the logger before throwing them up to the service.
   7. **CallSORTIO** Subroutine
      1. **Parameter**
         1. **drCharge** as DataRow
      2. Using an instance of SysProHandler, the subroutine will LogOn and TransactionPost with information from drCharge.
      3. Catches all exceptions to log and throw up to the service.
      4. Finally will call LogOff
4. **SysProHandler.vb**The SysProHandler will handle all interactions with the SYSPROWCFServicesClientLibrary40.dll.
   1. **Variables:**
      1. strXMLuser as string
      2. scfService as SYSPROWCFServicesPrimitiveClient
   2. **New** Subroutine
      1. Instantiates scfService with the application settings Base Aderss.
   3. **LogOn** Subroutine
      1. Using the applications logon credentials, the subroutine will logon to Syspro using scfService.Logon and storing the returned user data xml off into strXMLuser.
   4. **LogOff** Subroutine
      1. Logs off the user calling wcfService.Logoff with strXMLuser.
   5. **TransactionPost** Function
      1. **Parameters:**
         1. pBusObject as string
         2. pParameters as string
         3. pXMLIN as string
      2. **Returns**
         1. rtnXML as string
      3. Calls wcfServcie.TransactionPost using the parameters strXMLuser, pBusObject, pParamters, pXMLIN
      4. Will store returned xml into a local variable rtnXML to be returned
      5. Process rtnXML with CheckXmlOutForError. If an error is found, with create and throw new exception with message generated in the CheckXmlOutForError function.
      6. All exceptions are caught to be logged within SysProHandler before being thrown again.
   6. **CheckIfCompanyLocked** Function
      1. **Returns**
         1. rtnBool as boolean
      2. Calls wcfService.Logon and then scfService.Logoff
      3. Exception is caught and processed for if the operator is Disallowed setting rtnBool as True
      4. If no exception is encountered, returns false
   7. **CheckXmlOutForError** Function
      1. **Parameters**
         1. xmlOut as string
         2. errMsg as string
      2. **Return**
         1. rtnBool as Boolean = False
      3. Sets up an XmlDocument object with xmlOut parameter.
      4. Calls XmlErrorNode with the XmlDocument.DocumentElement
   8. **XmlErrorNode** Function
      1. **Parameters**
         1. Node as XmlNode
         2. errMsg as string
      2. **Return**
         1. rtnBool as Boolean = false
      3. For Each XmlNode in parameter node.ChildNodes it will see if it has ChildNodes and will call itself XmlErrorNode with the node. Else, it will check the ParentNode to see if it is “ErrorDescription”. If so, sets the errMsg to the value of the tag.

# External Interface Design

## 4.1 Interface Architecture

## 4.2 Interface detailed design

# Human-Machine Interface

## 5.1 Interface Design Rules

### 5.1.1 Inputs

The input to the service starts at the insert of records into the SorMaster\_Process\_Staged table. The following processes and the functions that insert the records are as follows:

1. **Dispatch** | trg\_AdmSignatureLog\_625100\_AfterUpdate  
   The processing for dispatching begins with the trigger trg\_AdmSignatureLog\_625100\_AfterUpdate. The trigger will insert the corresponding SalesOrder, ProcessType of 0, and DispatchNote in OptionalParms1.

# Appendices

## Appendix A: Other supporting documentation

1. SOH Project Table Structures   
   This document holds details of the tables referenced in the project and details related to there structure.
2. Syspro Business Objects
   1. SORTOI.xml
   2. SORTOI.xsd
   3. SORTOIDOC.xml
   4. SORTOIDOC.xsd
   5. SORTOIOUT.xml
   6. SORTOIOUT.xsd