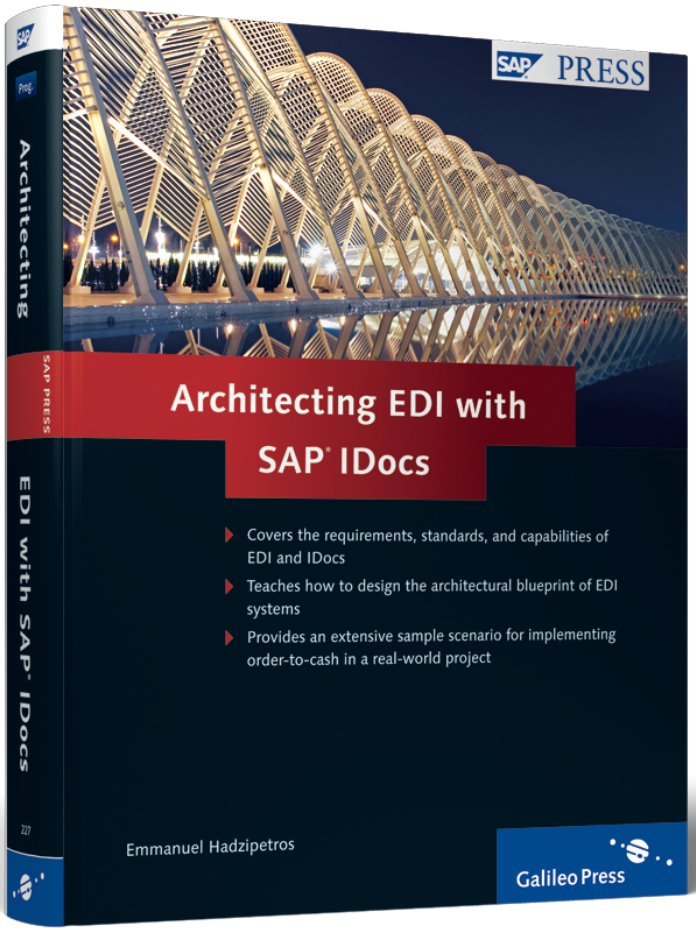
Emmanuel Hadzipetros

Architecting EDI with SAP® IDocs



Galileo Press®

Bonn • Boston

Contents at a Glance

Contents

Ordering product is the heart of the business, and Darryl Q would have understood the need for an effective process. It's time then to step through the 850-ORDERS interface build. We'll provide the specs and go over the business process, mapping requirements, and custom development and configuration in SAP.

8 Building the 850–ORDERS Inbound

We've been through the blueprint phase and looked at Acme's business and legacy systems. We have a design for our new SAP EDI system and have learned a little bit about EDI and the IDoc interface.

The time has come to build some interfaces.

The seven chapters of Act 3 are written as functional and technical specs for the key interfaces in the order-to-cash cycle between Acme Pictures and its most important customer Gordy's Galaxy of Games & B Flix.

They're not formal specifications. They present requirements for building our interfaces, including EDI to IDoc mapping and a discussion of any custom code or configuration that may need to be developed.

We'll begin with the inbound EDI X12 850 PO to ORDERS IDoc interface, the foundational process in the order-to-cash cycle.

The wrinkle is that Acme and Gordy are VMI (vendor managed inventory) partners, so they exchange two types of orders, both of which post to sales orders in SAP against a customer PO.

These are straight EDI for new release: X12 850 PO to ORDERS IDoc and VMI orders for replenishment and catalog. VMI is a two-step process:

- An X12 852 feed to Acme's VMI system used to support calculation of suggested customer orders
- VMI flat file with suggested customer PO to ORDERS IDoc

We'll look at both processes in this chapter. The end result is the same: Sales orders are created in SAP by an ORDERS IDoc.

8.1 Functional Specifications

Inbound EDI 850 POs or VMI orders make up the sole means of creating sales orders for Gordy's Galaxy in Acme's new SAP system. Data to create SAP sales orders are mapped from the 850 or the VMI order file to an IDoc using the ORDERS message with ORDERS05 basic type.

8.1.1 Process Overview

The process begins with an EDI transmission, either an 852 POS to VMI or an 850 PO directly to SAP after being translated to an ORDERS IDoc.

The 852 is used in the VMI system to support calculations based on a variety of data feeds that generate suggested POs for Gordy's Galaxy. The VMI orders are then mapped to an ORDERS IDoc and sent into SAP to create sales orders.

Both EDI and VMI sales orders generate an acknowledgement in an ORDRSP IDoc that is sent to Gordy's Galaxy.

8.1.2 Requirements

Sales orders for Gordy's Galaxy are posted in SAP from customer POs sent either as 850 X12 EDI transmissions or as VMI orders in flat files from the VMI system. SAP sales orders are created for one sold-to and one ship-to partner and are identified by order type:

- ZEDI for EDI orders with no SDQ
- ZEDS for EDI orders with SDQ
- ZVMI for VMI orders

Other requirements include

- There is no duplicate posting of customer POs.
- The order type comes into the sales order from the IDoc.

- All dates sent by Gordy in the EDI 850 are posted to the sales order in SAP. Those that can't be accommodated in a standard date field are stored in a text element.
- Delivery dates for VMI orders are sent from the VMI system.
- Delivery plants for VMI orders are sent from the VMI system. They can be changed after the sales order is created but before delivery documents are generated.
- An order acknowledgement is created when the sales order is completed and is sent to the EDI system as an ORDRSP IDoc.
- Conditions that can lead to an incomplete order include
  - Customer credit check fails.
  - ATP (item availability) check for the ordered product fails to find sufficient inventory to fulfill the order.

8.1.3 Dependencies

The 850-ORDERS interface is dependent on master data, configuration, and development objects in SAP and the EDI RIM. This includes

- Master data objects required to create sales orders, including:
  - GL chart of accounts: Assigned to the company code to record dollar values for costs and revenues for the accounting system.
  - Customers: For sold-to and ship-to partners, payment terms, shipping conditions, and credit checks, assigned to Acme sales organization, distribution channel, and division.
  - Delivery plants: For assignment of vendor plants for shipping.
  - Materials: For finished movies on DVD ordered at the item level and for component materials in sales BOMs.
  - Bills of materials: Identifying components in ordered finished goods.
  - Customer material info records (table KNMT) : Not required for Gordy's Galaxy because it sends Acme's item numbers. But this is required for all customers who send their own internal item numbers.
  - Pricing conditions: For header-level and item-level standard prices, taxes, discounts, credits, promotions, freight charges, and so on.

  - EDPAR: Partner mapping from external to internal customer numbers ensuring identification of SAP sold-to and ship-to partners.
  - EDSDC: Sales organization data will be mapped for the SAP sold-to partner and the customer's vendor number for Acme.
  - Partner profiles: To identify the sold-to partner for the incoming IDoc order. Partner profiles will be at the sold-to partner level; there will be only one partner profile per EDI customer.
- IDoc configuration completed in SAP to support inbound ORDERS for Gordy's Galaxy.
- EDI maps built for 852 VMI and 850 order translations.
- Business process models (BPMs) built in the EDI RIM to route ORDERS IDocs to SAP.

8.1.4 Assumptions

Sales orders are created from ORDERS IDocs that have been processed by a scheduled job in SAP. There is only one sold-to and ship-to partner for each sales order, although a customer PO can generate multiple sales orders.

Other key assumptions include

- Supporting master data is loaded into SAP and a process defined for adding new sold-to and ship-to partners, materials, and BOMs.
- Gordy's Galaxy sends GLNs for its sold-to and ship-to partners.
- Gordy sends UPC numbers and Acme's internal SAP material numbers for all goods ordered.
- VMI order pricing is determined by the pricing conditions called when the sales order is created.
- Pricing for EDI orders is determined when the sales order is created in SAP by comparing the prices sent in Gordy's PO to the price proposed by the pricing conditions set for the material ordered.
  - If the two match or are within a tolerance limit, Gordy's price posts to the sales order.
  - If the difference between the two prices exceeds tolerance, the reason for the difference is identified, and the correct price is used.
- The base unit of measure for items ordered is EA (eaches).

- All data that must be returned to the customer in the invoice must post to the sales order from the IDoc. Data that can't be accommodated in a standard field in the order goes into a text element.
- A customer credit check is run when the sales order is created. A new credit check is run only if pricing or quantities are changed in the order.
- Business users are responsible for addressing application errors in the ORDERS IDocs.
- Incomplete sales orders are saved for manual editing.
- All errors in the EDI system are handled by the EDI team.
- EDI and VMI orders are sent into SAP immediately, and the IDocs are posted to sales orders within no more than an hour.
- EDI errors or issues that may affect the timeliness of order creation are communicated to the business users immediately.

8.1.5 Data Required to Create a Sales Order

SAP sales orders are created with Transaction VA01. At a minimum, the fields in Table 8.1 must be populated to create a sales order.

Table	Field	Description	Sample value
VBAK	AUART	Order type	OR
VBAK	VKORG	Sales organization	0010
VBAK	VTWEG	Distribution channel	10
VBAK	KUNNR	Sold-to partner	GRDY01
VBPA	PARVW	Partner qualifier — ship-to	WE
VBPA	KUNNR	Ship-to partner	GRDY01001
VBKD	BSTKD	Customer PO number	9997895
VBKD	BSTDK	Customer PO date	20081202
VBAK	VDATU	Requested delivery date	20081204
VBAP	MATNR	SAP material number	999284
VBAP	XXXX	Customer item number (UPC)	9998989989121
VBAP	KWMENG	Order quantity	230

Table 8.1 Fields That Are Populated When a Sales Order Is Created

8.1.6 Reconciliation Procedure

Successful import of the ORDERS IDoc is confirmed through any of the IDoc monitoring tools such as BD87 or WE05.

IDoc status should be 64 — *IDoc ready to be transferred to application* —before the scheduled processing job is kicked off and 53 — *Application document posted* — after.

The EDI team confirms the data in the IDoc against the data in the X12 850 transaction set sent from the customer, and the users validate that the sales order was created against the data sent in the IDoc.

8.1.7 Enhancements to the Process

An enhancement is required during sales order creation to ensure that a customer PO posts only once. This may be a little like squaring the circle. Each PO can contain product-ordering information for multiple store locations at the line-item level, whereas each SAP sales order only carries ordering information for one store.

This means that we must be able to create multiple sales orders for each PO while ensuring that the same customer PO doesn't post to a second batch of sales orders, which is a common issue on SAP EDI sites.

Double-posting results in double-ordering and duplicate shipments to the customer leading to unnecessary costs, returns, and customer dissatisfaction with Acme's service.

8.1.8 Enhancement Details

The duplicate PO check occurs in code during IDoc processing and is transparent to the user. It checks sold-to, ship-to, PO number, and PO date. If it finds existing sales orders against the PO, it checks if an ORDRSP IDoc was generated from it.

If an IDoc has been generated, the PO has already posted, its sales orders have completed, and an acknowledgement has been sent back to the customer. The incoming order is a dupe PO. It should trigger an error in the IDoc and stop it from posting.

8.1.9 Errors and Error Handling

Errors that may occur during processing of the inbound 850-VMI-ORDERS interface include

- The IDoc will fail if the sold-to or ship-to partners don't exist in SAP, or if the sales organization can't be determined. If these errors occur, the customer or sales org data are entered, and the IDoc reruns.
- The IDoc will fail if SAP can't identify the material number from the item number sent in the EDI transaction. The customer is asked to resend the PO, or the IDoc is edited and reprocessed.
- A customer PO that has already posted will trigger an error in the IDoc if it tries to post again. If the PO needs to be reposted, the sales orders that posted in the initial run are deleted.
- If there isn't enough inventory to fulfill an order when the sales order is created, it will be put on hold and deliveries won't be generated until inventory is entered and the order is released.
- If a customer credit check fails during sales order creation, the order will be put on hold until the credit department releases it.

Sales order or IDoc errors will be communicated to the responsible business user immediately. There is a service level agreement with the partner mandating how quickly shipments need to be sent after orders are received.

Standard IDoc monitoring programs such as WE05 or even B87 will be used to track and monitor IDocs.

8.2 End-to-End Process Flow

An overview of the end-to-end process flow for creating SAP sales orders from EDI transmissions is outlined in Figure 8.1.

Two processes are at work here for Gordy's Galaxy: new release with EDI orders and replenishment, and catalog with VMI orders. The two processing flows merge with the creation of one ORDERS IDoc instance for each sales order that will be created in SAP.

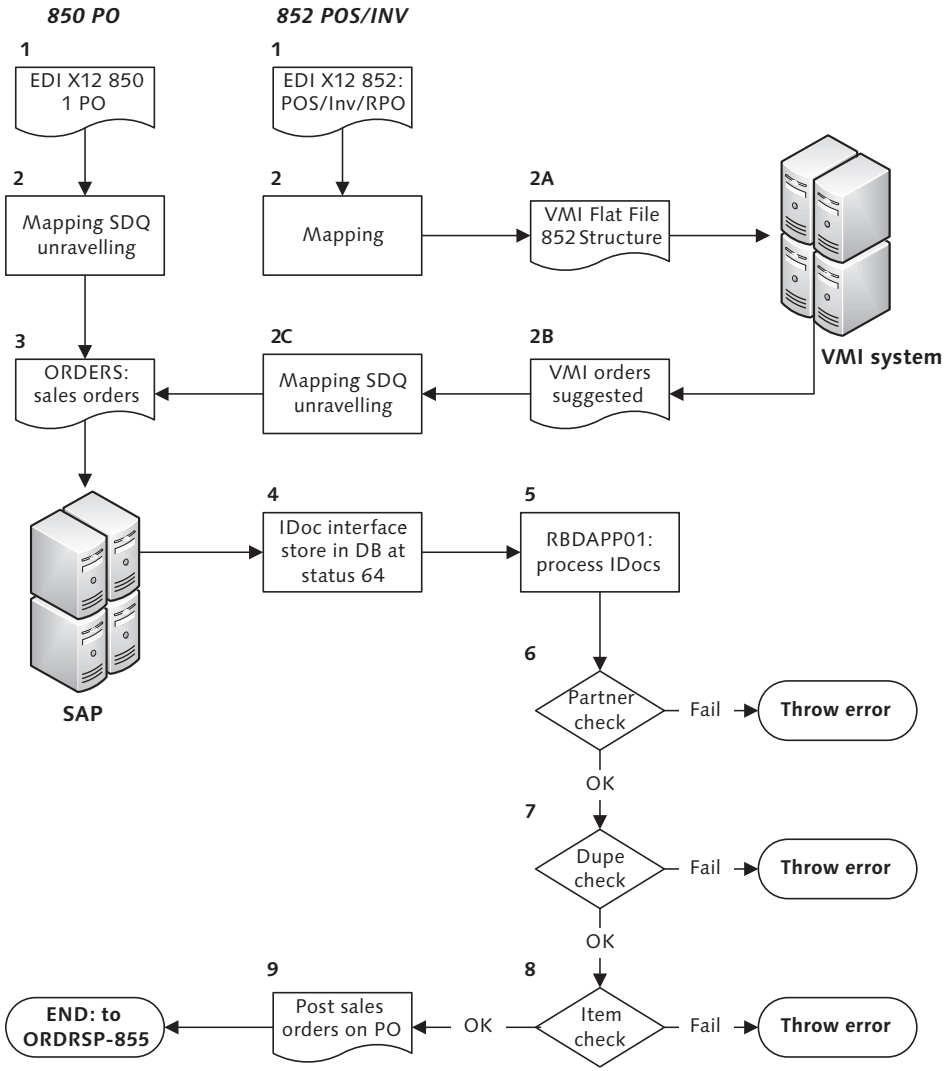


Figure 8.1 The Inbound EDI/VMI Order Creation Process

This is easier said than done: The 850 PO and the VMI order file include all ordering data for each of Gordy's 2,000 store locations at the item level. Ordering quantity for each location is stored in the 850 in one or many SDQ segments that occur as children to item-level segment PO1, which contains item numbers for the product being ordered.

The SDQ segment can hold order quantities for up to 10 stores. The VMI order file is structured in a similar manner. But an SAP sales order can only be created for one sold-to and one ship-to location. So we need to build one ORDERS IDoc for each store that orders product. And we must include every item that it's ordering in the IDoc.

This can be handled in an ABAP program if we build an ORDERS05 basic type with an SDQ segment to bring the PO data into SAP.

We're going to look at the logic for doing this, whether in a map through a Java exit, a script, or in ABAP. The logical problem is the same, but the specifics of doing it vary from tool to tool. Many mapping tools have robust programming or rules languages that allow conditional processing, looping, indexing, and also support arrays and even Java objects.

As long as the mapping tool has access to all of the source structures and data, it should be able to unravel the SDQ into multiple orders with the help of a little creative coding.

VMI Processing

The VMI process flow begins with an 852 transmission from Gordy's Galaxy by AS2 into the EDI RIM. The 852 carries three types of data:

- Store-level point of sales (POS) data transmitted every night. The 852 POS is a consolidation of check-out scans of items sold in each store throughout the day.
- Store inventory levels sent once a week.
- Open reserved PO numbers (RPOs) sent intermittently, before Acme runs out of PO numbers for VMI orders.

Regardless of the data it holds, the 852 is mapped to an internal flat file that has a structure similar to the 852 and is sent into the VMI system.

In VMI, POS and inventory data support calculations that generate POs for replenishment of goods and catalog product for Gordy's stores. The RPO file is used to assign PO numbers to the calculated orders.

The POS and inventory feeds are also sent to StoreData for use in store-level replenishment calculations.

The final orders calculation is also based on a number daily feeds from SAP, including but not restricted to

- Customer store locations
- Finished goods master data
- BOMs and BOM changes since the last feed
- Inventory levels at the vendor's warehouse
- Open and changed sales orders
- Open deliveries
- Returns
- Open vendor POs for manufacture of finished goods

There are also daily extracts from StoreData and other backend systems, including shelf location and dimensions in all of Gordy's stores, titles ordered by store and item number, and minimum and maximum order levels per store.

VMI calculates a PO for each of Gordy's stores. A PO number is pulled from a table populated by the RPO feed. RPO is then marked as consumed and is no longer available for use.

The VMI order is extracted to an ASCII file by VMI. The file has a flatter structure than an 850 PO but includes an SDQ-like record with order item quantity for up to six stores in each segment, that is, a child to an item header that identifies the product being ordered.

The following key values are mapped to the IDoc:

- The SAP sold-to partner number for Gordy to the send partner field in the control record EDIDC-SNDPRN.
- Order type OR for standard orders to field E1EDK14-ORGID with qualifier 012 in field QUALF.
- The RPO number to field E1EDK02-BELNR with qualifier 001 in field QUALF.
- Gordy's store location GLN to E1EDKA1-LIFNR with qualifier WE in field PARVW.
- The quantity to be ordered for each item in field E1EDP01-MENGE.
- The SAP material number for each movie ordered in field E1EDP19-IDTNR with qualifier 002 in field QUALF.

- The item's UPC code in field E1EDP19-IDTNR with qualifier 003 in field QUALF.
- Gordy's item number in field E1EDP19-IDTNR with qualifier 001 in field QUALF.

The VMI order file is exported to the EDI RIM, where it's identified as a VMI order for Gordy's Galaxy. RIM calls a map that unravels the store-level data and builds one ORDERS IDoc for each store and each product being ordered by that store.

The IDocs are batched together into a file and, at this point, the VMI process ends, and the IDoc is sent into SAP through the IDoc adapter.

EDI 850 Processing

The EDI processing flow begins with receipt of an 850 PO transmission from Gordy's Galaxy by AS2 into the EDI RIM. RIM identifies the 850 from Gordy and calls the map to translate it.

Gordy uses the SDQ segment at the item level to identify each store and the quantity of product being ordered. As with the VMI file, the map unravels order data from the item level and builds one ORDERS IDoc for each store and each product being ordered by that store.

The same key values are mapped to the IDoc as for the VMI order.

The IDocs are batched into a file and sent by RIM into SAP through the IDoc adapter by calling function `EDI_DATA_INCOMING`.

At this point, the VMI and EDI processes merge. The SAP IDoc interface kicks in, confirms that the file contains IDocs, checks that there are matching partner profiles, and writes the IDocs to the database at status 64.

The IDocs are processed by program RBDAPP01, which is scheduled to pick up Gordy's orders every hour. RBDAPP01 reads the IDoc database and identifies all ORDERS IDocs at status 64 where EDIDC-SNDPRN equals Gordy's SAP sold-to partner.

It then identifies the IDoc function — `IDOC_INPUT_ORDERS` — from the process code ORDE in the inbound partner profile for Gordy's Galaxy message type ORDERS and calls it to post the IDocs to sales orders. `IDOC_INPUT_ORDERS` takes over and loops through the IDoc.

When it hits segment E1EDKA1, it reads EDPAR to convert Gordy's GLN to the SAP ship-to partner.

If the EDPAR read fails, an error is thrown, IDoc processing stops, and a status 51 application error is added to the IDoc status record. In most cases, this error occurs because either the location has not been added as a ship-to partner in SAP, or it doesn't yet exist for the sales organization posting the order.

The user creates the ship-to record or extends it to the sales org in the customer master and updates EDPAR to create the link among Gordy's sold-to, ship-to, and GLN. The IDoc is then reprocessed.

After the successful EDPAR conversion of Gordy's GLN to the SAP ship-to partner, a duplicate order check is run to ensure that the customer PO hasn't already posted a sales order for the current sold-to and ship-to partners.

Table VBAK is read for the sales org, order type, SAP sold-to partner, customer PO number, and date. If there's no hit, there's no dupe, and IDoc processing continues to the next check.

If there is a hit, table VBPA is read with the sales order number and sold-to partner to identify the ship-to. VBPA stores complete partner data for all sales documents. If there's no hit, PO duplicate check processing exits.

If there is a hit, we have a possible dupe. We need to check if the sales order generated an ORDRSP IDoc, proving that it was completed and generated an acknowledgement to the customer.

This is done by calling function `NREL_GET_NEIGHBOURHOOD`, which links the sales order number — the object key — and its object type — BUS2032 — to all IDocs that either created or were generated from it.

If it returns an ORDRSP IDoc number, then the PO is a dupe, and an error is thrown. IDoc processing stops and a status 51 application error is added to the IDoc status record.

The responsible user then checks the sales order and confirms. If it's a dupe, either the IDoc or the posted sales order is marked for deletion. If the sales order is deleted, the IDoc is reprocessed.

If the PO number is incorrect, it's corrected in the IDoc and reprocessed.

The next check is on the item with the SAP material number checked first. Gordy sends Acme's SAP number, so this check should pass. Gordy gets a daily finished goods synchronization extract from SAP and should always have Acme's most up-to-date catalog.

If it fails, the customer info record is checked in table KNMT, which will fail because we're not maintaining CMIR for Gordy.

The final check is on the UPC number, which Gordy also sends. If the item check fails, an error is thrown, IDoc processing stops, and a status 51 application error is added to the IDoc status record.

If the material exists in SAP but has not been extended to the sales org posting the order, the check fails. Extend the material and reprocess the IDoc. If the material doesn't exist in SAP, the error is in the EDI file.

The business contacts Gordy and confirms the product ordered and its quantity. The responsible user then changes the material in the IDoc using Transaction BD87 or one of the other standard IDoc list programs discussed in Chapter 17, Troubleshooting and Recovery. The IDoc is then reprocessed.

In some cases, Gordy is asked to resend the order.

The IDoc function also checks table EDSDC for sales org data. Because this is a one-time data entry job, there should be no error unless someone deleted the record from EDSDC.

If all checks are passed, or errors are corrected, and the IDoc reprocessed, the function passes data from the IDoc to an internal table referencing Data Dictionary structure BDCDATA. This data is then used to create the sales order through a call to Transaction VA01. As the order is saved, the system does its ATP and credit checks.

The process ends when output control kicks in and generates an ORDRSP IDoc to send to Gordy to acknowledge posting of its PO or VMI order.

8.3 Technical Specifications

This technical specification describes interface configuration and custom program support in the EDI RIM and SAP for the delivery, translation, and creation of EDI and VMI sales orders in SAP.

8.3.1 Interface Summary

The summary specifications for this interface are outlined in Table 1.2.

Item	Description
Type of Interface	X12 EDI or VMI to IDoc
Logical Message Type	ORDERS
Basic Type	ORDERS05
Custom ABAP 1	User exit in enchancement VEDA0001 in modification project ZEDISOO1
Short Description	Dupe PO check on ship-to and PO number
Interface Direction	Inbound
Source File	850 (PO), 852 (VMI), VMI proposed orders flat file
Trading Partner	Gordy's Galaxy
Map	X12 850 vers. 5010 — ORDERS.ORDERS05 VMI orders FF — ORDERS.ORDERS05
Custom Map Logic	One-to-many mapping; unbundle store order quantity per material in SDQ segments into one IDoc per store
Source System	Gordy's Galaxy EDI via AS2
Target System	Acme SAP via EDI RIM
997 Acknowledgment	Outbound function code PO at transaction detail level
Frequency	Daily, on demand
Schedule	RBDAPP01: Every hour, posts IDocs to sales orders

Table 8.2 Base Specifications for Inbound EDI Orders

8.3.2 Technical Requirements

One ORDERS IDoc is generated by the translation map for each store location and all items ordered by that location for standard order type OR.

User exit coding blocks duplicate posting of customer POs to SAP sales orders.

Configuration in the EDI RIM and SAP support inbound orders and outbound 997 acknowledgements.

8.3.3 Dependencies

The 850-VMI-ORDERS interface is dependent on a number of development objects in SAP and the EDI RIM:

- Inbound envelopes set up in RIM for Gordy's 850 and 852 version 5010 EDI transactions
- Outbound envelopes set up in RIM for 997 FA to be generated for Gordy's Galaxy during deenveloping of the inbound 850 and 852
- Custom cross-reference table ZEDIXREF populated in SAP to read the SAP send and receive partners for the inbound 850 from Gordy
- Job set up in the SAP Job Scheduler (SM36) to post ORDERS IDocs with program RBDAPP01 with variants to select for Gordy's Galaxy

8.3.4 Assumptions

POs from Gordy's Galaxy post from 850 EDI transactions and VMI orders. Gordy sends both SDQ and non-SDQ POs.

The map, a script, or a custom external or ABAP program extract store-level ordering data in the SDQ segment into an indexed array and build one ORDERS IDoc for each store location.

RIM maps the EDI send and receive trading partner IDs to the IDoc control record fields SNDLAD and RCVLAD. These fields are read by an exit in the IDoc interface to identify the SAP sold-to partner for field EDIDC-SNDPRN.

During the EDPAR check on partner segment E1EDKA1, the SAP sold-to partner is read from EDIDC-SNDPRN.

EDI errors are tracked and addressed in the EDI system. Technical errors in the IDoc interface, such as syntax or partner profile errors, are tracked and corrected by the EDI team.

8.3.5 SDQ Processing in the ORDERS05 Translation

SDQ records at the item level of the VMI order and the EDI 850 PO contain order quantities for each store by product. The map, with a little custom coding, extracts item and quantity data for each location, identified by GLN in the SDQ record, and build one ORDERS IDOC for each store.

GLN for the store is inserted into field E1EDKA1-LIFNR with qualifier WE at the header level of the ORDERS IDoc. The basic principle is that each SAP sales order includes all DVD movies ordered by one sold-to partner — Gordy's Galaxy — for one ship-to — Gordy's store location.

The map has a one-to-many relationship between the input and the output. In addition, it moves the store's GLN from the item level of the input to the header level of the output.

This is a common issue in EDI implementations. SDQ is widely used in the 850 PO, 852 POS, and 855 confirmation. Most mapping tools that handle this do so with custom code. We'll look at a logical process that uses an indexed array and some looping that can be used to build one IDoc for each store in an SDQ segment.

To better understand where we're coming from and where we want to go, we need to look at our mapping spec for the 850 SDQ PO to the ORDERS IDoc, as outlined in Table 8.3. This spec is focused on the application data. We won't look at the IDoc control record or at any of the key fields of the control area of the data records.

The spec has been simplified for clarity, including the use of abbreviations for IDoc segment names. Header-level segments begin with K, and item-level segments with P. This is pretty consistent in SAP. EDI data elements are identified with their segment name and position number.

Before we begin, note that common usage is for the EDI transaction to be on the left, regardless of whether it's the source or the target structure. We'll follow this practice for consistency.

Source	Target	Value	Comments
Unmapped IDoc Constants			
	K01-BSART	NB	Purchasing doc type.
	K14-QUALF	12	Order type.
	K14-ORGID	TA	Standard sales order.
BEG – Begin Segment for PO Create – Mandatory – Max 1			
	K02-QUALF	001	Customer PO.
BEG03	K02-BELNR		PO number.

Table 8.3 Simplified Mapping Spec for 850 SDQ PO to ORDERS IDoc

Source	Target	Value	Comments
BEG05	KO2-DATUM		PO date.
DTM – Date Segment – Mandatory – Delivery Date – Instance 1 of 2			
DTM01	K03-IDDAT	002	Identifies requested delivery date where DTM01 = 010.
DTM02	K03-DATUM		Customer delivery date.
DTM – Date Segment – Mandatory – Shipping Date – Instance 2 of 2			
DTM01	KT1-TDID	ZSHD	Identifies ship date where DTM01 = 010.
DTM02	KT2-TDLINE		Ship date posts to orders as text.
N1 Looping Group Begin – Partner Identification – Mandatory – 1 to 200			
N1 – Partner ID Segment – Mandatory – Max 1 per loop – Loop 1			
N101	KA1-PARVW	LF	Customer vendor number, where N101 = SU.
N103		UL	Identifies GLN for partner.
N104	KA1-LIFNR		Acme GLN.
N1 Loop 2			
N101	KA1-PARVW	WE	Customer ship-to number where N101 = ST. Non-SDQ only. Won't be present in SDQ.
N103		UL	Identifies GLN number for partner.
N104	KA1-LIFNR		Gordy store location GLN. Non-SDQ.
N1 Looping Group End			
PO1 Looping Group Begin – Item Detail – Mandatory – 1 to N			
PO1 – Baseline Item Data – Mandatory – Max 1 per PO1 loop			
PO101	PO1-POSEX		Item no.
PO102	PO1-MENGE		Quantity ordered. Non-SDQ only.
PO103	PO1-MENEE	EA	Base unit of measure eaches. Non-SDQ only.
PO104	PO1-VPREI		Unit price.
PO106	P19-QUALF	001	Customer material number where P0106 = IN.
PO107	P19-IDTNR		Customer's product number.
PO108	P19-QUALF	003	UPC for material where PO108 = UP.
PO109	P19-IDTNR		UPC for material.

Table 8.3 Simplified Mapping Spec for 850 SDQ PO to ORDERS IDoc (cont.)

Source	Target	Value	Comments
PO110	P19-QUALF	002	Acme SAP material number where PO110 = VN.
PO111	P19-IDTNR		SAP material number.
SDQ – Store Location Order Quantity Data– Option – 1 to N per PO1 loop			
SDQ01	PO1-MENEE	EA	Base unit of measure = eaches.
SDQ02	K01-PARVW	WE	SAP ship-to partner function by GLN = UL. Inserted in header E1EDKA1 segment of each ORDERS IDoc. For all stores in SDQ.
SDQ03	K01-LIFNR		Store location GLN for ORDERS IDoc 1.
SDQ04	P01-MENGE		Quantity to ORDERS IDoc 1.
SDQ05	K01-LIFNR		If exists, store location GLN to IDoc 2.
SDQ06	P01-MENGE		Quantity ordered to IDoc 2.
SDQ07	K01-LIFNR		If exists, store location GLN to IDoc 3.
SDQ08	P01-MENGE		Quantity ordered to IDoc 3.
SDQ09	K01-LIFNR		If exists, store location GLN to IDoc 4.
SDQ10	P01-MENGE		Quantity ordered to IDoc 4.
SDQ11	K01-LIFNR		If exists, store location GLN to IDoc 5.
SDQ12	P01-MENGE		Quantity ordered to IDoc 5.
SDQ13	K01-LIFNR		If exists, store location GLN to IDoc 6.
SDQ14	P01-MENGE		Quantity ordered IDoc 6.
SDQ15	K01-LIFNR		If exists, store location GLN to IDoc 7.
SDQ16	P01-MENGE		Quantity ordered to IDoc 7.
SDQ17	K01-LIFNR		If exists, store location GLN to IDoc 8.
SDQ18	P01-MENGE		Quantity ordered to IDoc 8.
SDQ19	K01-LIFNR		If exists, store location GLN to IDoc 9.
SDQ20	P01-MENGE		Quantity ordered to IDoc 9.
SDQ21	K01-LIFNR		If exists, store location GLN to IDoc 10.
SDQ22	P01-MENGE		Quantity ordered to IDoc 10.
PO1 Looping Group End			

Table 8.3 Simplified Mapping Spec for 850 SDQ PO to ORDERS IDoc (cont.)

Mapping Structures

The mapping specs help us visualize our data. We need to see the data to make this work. So let's visualize our 850 file.

We'll take a two-step approach. First we'll map the 850 input data to a flat structure in a temporary looping group with one header and multiple item records that we'll use to build one IDoc for each SDQ store location.

The temporary group will mimic and simplify the ORDERS05 basic type structure. This is where our custom code will collect and assemble the IDoc data before passing it to the relevant fields in the target IDoc.

The code will unravel the SDQ segments by store and quantity pair and build in the temporary group one IDoc per location with the store's GLN in the header-level KA1-LIFNR field. We'll then map each field in the temporary looping group to the IDoc.

The temporary looping group will contain everything we need to build an IDoc, including control segment fields in the header and control key values for the data records at both header and item levels.

Assuming a mapping tool that supports this, we'll append the temporary looping group to the end of our input at the same level as the transaction. It will contain a header section with only one record and another looping group for the item-level data.

The target IDoc is also set up as a looping group with the same hierarchy as our temporary flattened IDoc, allowing generation of multiple instances of the IDoc for each X12 850 transaction. This ensures that the looping levels in the temporary input structure and the IDoc output are compatible. In other words, parent-child relationships and the number of looping iterations for each group will be identical on both sides.

Repeating IDoc segments, such as E1EDKA1, will be copied into single instances of the segment specific to one qualifier for one-to-one mapping. For example, the IDoc will have two instances of E1EDKA1: for the vendor (PARVW = LF) and the ship-to partner (PARVW = WE).

We'll also need to put logic in the SEGNUM field of the control key area of each data record to increment a counter that will provide a sequential number for each

IDoc segment, regardless of hierarchy. We declare a global variable and increment it each time SEGNUM is processed:

```
SEGNUM = seg_cnt + 1.
```

Mapping specifications for the temporary looping group and the IDoc are detailed in Table 8.4. We'll use the same approach for the VMI order, which also has an SDQ record so we don't need to repeat the details here.

Input	IDoc Target	Value	Comments
TMP_ORDERS Looping Group Begin –IDoc Mapping Structure – 1 to N			
IDOC_HDR – Header Level Data – Mandatory – Max 1			
DC40_MANDT	EDI_DC40-MANDT	100	SAP target client
DC40_DOCREL	EDI_DC40-DOCREL	620	SAP version
DC40_DIRECT	EDI_DC40-DIRECT	2	Inbound
DC40_IDOCTYP	EDI_DC40-IDOCTYP	ORDERS05	Basic type
DC40_MESTYP	EDI_DC40-MESTYP	ORDERS	Message type
DC40_STD	EDI_DC40-STD	X	EDI standard
DC40_STDVRS	EDI_DC40-STDVRS	005010	EDI version
DC40_STDMES	EDI_DC40-STDMES	850	EDI transaction
DC40_SNDPOR	EDI_DC40-SNDPOR	EDI_IDOC	Sender file port
DC40_SNDPRT	EDI_DC40-SNDPRT	KU	Customer
DC40_SNDPFC	EDI_DC40-SNDPFC	AG	Sold-to partner
DC40_SNDPRN	EDI_DC40-SNDPRN	GRDY01	Gordy's sold-to
DC40_SNDLAD	EDI_DC40-SNDLAD		Gordy's TP ID
DC40_RCVPOR	EDI_DC40-RCVPOR	SAPDEV	Receiver port
DC40_RCVPRT	EDI_DC40-SNDPRT	LS	Logical system
DC40_RCVPRN	EDI_DC40-SNDPRN	SAPDEV100	Logical client
DC40_RCVLAD	EDI_DC40-RCVLAD		Acme's TP ID
DC40_REFINT	EDI_DC40- REFINT		ISA Cntrl number
DC40_REFGRP	EDI_DC40- REFGRP		GS Grp Cntrl number

Table 8.4 Mapping the Temporary ORDERS Flat IDoc Structure to the Target ORDERS IDoc for SDQ Processing

Input	IDoc Target	Value	Comments
DC40_REFMES	EDI_DC40- REFMES		ST Txn ID
K01_HLEVEL	K01-HLEVEL	1	Hierarchy level
K01_BSART	K01-BSART	NB	Purchase doc type
K14_HLEVEL	K01-HLEVEL	2	Hierarchy level
K14_QUALF	K14-QUALF	12	Order type
K14_ORGID	K14-ORGID	TA	Standard order
K03_HLEVEL	K03-HLEVEL	2	Hierarchy level
K03_IDDAT	K03-IDDAT	002	Req. delivery date
K03_DATUM	K03-DATUM		Date
KA1_LF_HLEVEL	KA1-HLEVEL	2	Hierarchy level
KA1_LF	KA1-PARVW	LF	Cust vendor number
KA1_LF_LIFNR	KA1-LIFNR		Acme GLN
KA1_WE_HLEVEL	KA1-HLEVEL	2	Hierarchy level
KA1_WE	KA1-PARVW	WE	Cust. ship-to number
KA1_WE_LIFNR	KA1-LIFNR		Gordy ship-to from SDQ segment
K02_HLEVEL	K02-HLEVEL	2	Hierarchy level
K02_QUALF	K02-QUALF	001	Customer PO
K02_BELNR	K02-BELNR		PO no
K02_DATUM	K02-DATUM		PO date
KT1_TDID	KT1-TDID		Ship date text ID
KT2_TDLINE	KT2-TDLINE		Ship date to PO text
IDOC_ITEM_Grp Looping Group Begin – Item Detail – Mandatory – 1 to N			
IDOC_ITEM – Item Details Max 1 per IDOC_ITEM loop			
P01_HLEVEL	P01-HLEVEL	2	Hierarchy level
PO1_POSEX	PO1-POSEX		Item number
PO1_MENGE	PO1-MENGE		SDQ qty
PO1_MENEE	PO1-MENEE	EA	UOM

Table 8.4 Mapping the Temporary ORDERS Flat IDoc Structure to the Target ORDERS IDoc for SDQ Processing (cont.)

Input	IDoc Target	Value	Comments
PO1_VPREI	PO1-VPREI		Unit price
P19_HLEVEL_002	P19-HLEVEL	3	Hierarchy level
P19_QUALF_002	P19-QUALF	002	Acme SAP item
P19_IDTNR	P19-IDTNR		Material number
P19_HLEVEL_001	P19-HLEVEL	3	Hierarchy level
P19_QUALF_001	P19-QUALF	001	Customer item
P19_IDTNR	P19-IDTNR		Material number
P19_HLEVEL_003	P19-HLEVEL	3	Hierarchy level
P19_QUALF_003	P19-QUALF	003	UPC
P19_IDTNR	P19-IDTNR		UPC product number
IDOC_ITEM Looping Group End			
TMP_ORDERS Looping Group End			

Table 8.4 Mapping the Temporary ORDERS Flat IDoc Structure to the Target ORDERS IDoc for SDQ Processing (cont.)

As you can see, everything we need to build an IDoc is in this temporary structure in two segments. All we need to do is unravel store and order quantity pairs from the SDQ segment associated with each item, move the store GLN to the header of the IDoc, and move the order UPC and quantity to the item level of our structure. Then we map each field to the target IDoc.

We're building an IDoc in memory through code that we'll then map to the target IDoc. To do this, we have to write code, whether it's inside the map or outside the map. The capabilities of your mapping tool have to be considered. But this logic can be applied in any scripting or programming language that can read and process an input to build an output, including ABAP.

So let's give it a try.

Program Logic

Our basic working assumption is that the entire 850 input file is available for processing before we map our data to the ORDERS IDoc output. This can happen in memory in a user exit or within the mapping program's normal processing flow.

We're also assuming that we can declare an indexed array in our mapping tool and that the code can be written and called from a user exit or external program or class file.

We'll process the 850 file in three loops with the help of a number of key indexes. The IDocs are built in the temporary looping structure during the deepest loop, at the item level, after the store and quantity data in the SDQ segments have been moved into an indexed array.

The logical processing flow for creating one ORDERS IDoc for each store in an SDQ 850 PO is outlined in Figure 8.2. There's a lot of stuff happening here, but it's not as complex as it might seem.

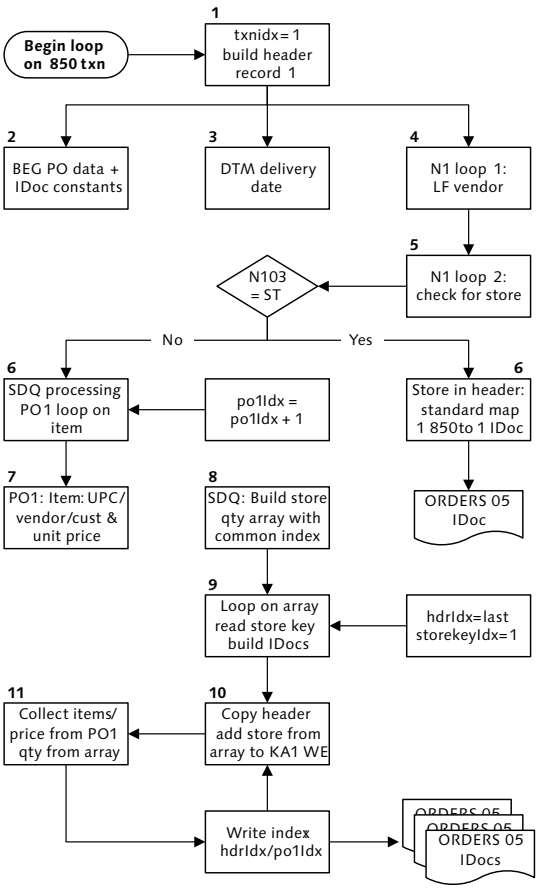


Figure 8.2 Logical Processing Flow for SDQ Orders

The code loops through the input one 850 transaction at a time, setting the transaction-level index — `TXNIDX` — to 1. All other indexes are reset to 0. This index identifies common header data pulled from the current 850 transaction that will be written to the IDOC_HDR record of each IDoc that we build for each store in our SDQ segments.

The BEG segment is read first. Base PO data is moved into the K02 fields of the IDOC_HDR record using index `TXNIDX`:

- BELNR: Customer PO number from BEG03.
- DATUM: PO date from BEG05.
- QUALF: PO qualifier 001.

Constant IDoc header values are also passed to the header records:

- EDI_DC40: All IDoc control record fields.
- K01: HLEVEL 1 and BSART NB.
- K14: HLEVEL 2, QUALF 12, and ORGID TA.
- K02: HLEVEL 2.

The DTM segment is read next. DTM01 is checked for qualifier 010. If it's found, the following values are written to the K03_002 fields with index `TXNIDX`:

- HLEVEL: 2.
- IDDAT: Qualifier 002 identifying the requested delivery date.
- DATUM: Date from DTM02.

A translation error is thrown if qualifier 010 isn't found in DTM01. The delivery date is a mandatory field for the SAP sales order.

The N1 looping group is read next. N103 is checked for qualifier SU. If it's found, the vendor number is passed to the IDOC_HDR KA1_LF fields using index `TXNIDX`:

- HLEVEL: 2.
- PARVW: Qualifier LF.
- LIFNR: Acme's GLN from N104.

During loop read 2 of the N1 Group, N103 is checked for qualifier ST store location. If it's found, there are no SDQ segments, and standard processing proceeds. One IDoc is generated for each 850 transaction using index `TXNIDX` to write the

header segments and P01IDX to write the items. Gordy is very good about this and never sends a store in the N1 loop with an SDQ order. In the real world, things aren't always so clean.

If the ST qualifier isn't found in the N1 loop, an SDQ flag is set, and SDQ processing proceeds.

The PO1 looping group is processed. It's read in a loop with segment PO1 the first to be read. The item index P01IDX is incremented. P01IDX is used with TXNIDX to write item data to the IDOC_ITEM record, linking the PO1 item to the header record of the first IDoc being written.

The following values are passed from the PO1 segment to the P01 fields of the IDOC_ITEM record using index TXNIDX.P01IDX:

- HLEVEL: 2.
- POSEX: Item number from PO101.
- VPREI: Unit price from PO104.

The ordered items are passed next to the P19 fields of the IDOC_ITEM record using index TXNIDX.P01IDX. They are read from the PO1 segment of the 850:

- HLEVEL: 3. In the IDoc, segment E1EDP19 is a child to E1EDP01.
- QUALF_002: Qualifier 002 where PO110 = VN.
- IDTNR: Acme's SAP material number from PO111.
- QUALF_001: Qualifier 001 where PO106 = IN.
- IDTNR: Gordy's material number from PO107.
- QUALF_003: Qualifier 003 where PO108 = UP.
- IDTNR: UPC number for the material from PO109.

Qualifier 002 is passed first because Gordy sends Acme's SAP item number, and this eliminates material determination during IDoc processing.

The values that we've collected into our temporary IDoc so far serve as the template that we'll use to build each IDoc that we'll create for each store and quantity pair in the SDQ segment within the current item loop. These values will be common to all IDocs that we create from this 850 for each store regardless of items and quantity ordered.

Now we come to the fun part. The SDQ segments are read, one at a time. They hold the store locations as GLNs paired with an order quantity for the material in the parent PO1 segment.

Each store location and quantity pair is moved into an indexed array in memory — an internal table in ABAP — that we'll loop through to create one IDoc for each store. Different programming tools handle this task differently but the array could look something like Table 8.5.

Index	GLN	Qty
1	9997495958768	23
2	9997495959876	12
3	9997495960786	6
4	9997495961986	45
5	9997495962686	20

Table 8.5 Indexed Array with Store Order Quantity Pairs

This unravels the SDQ into a tabular structure with one record per store and quantity pair.

We can now loop through this array within our current loop on the 850 PO1 item and match the store and quantity to the material being ordered for it.

All SDQ segments within the PO1 group are at runtime processed one at a time in the order in which they appear in the group. The logic to build the SDQ array would look something like this in pseudo code:

```
IF SDQ0N IS NOT NULL THEN
  MOVE SDQ0N TO SDQ_ARRAY COL2
  MOVE SDQ0NN TO SDQ_ARRAY COL3
  SDQIDX = SDQIDX + 1
  MOVE SDQIDX TO SDQ_ARRAY COL1
  STORECNT = SDQIDX.
ENDIF.
```

Each store location and quantity pair in each SDQ segment is treated in the same way. SDQ0N is the number of the location data element beginning with SDQ03 and ending with SDQ21. SDQ0NN is the number of the quantity data element beginning with SDQ04 and ending with SDQ22.

We also need to get a total count of all store-quantity pairs in the SDQ array. We'll put this into a counter called STORECNT, which equals 5 in our example in Table 8.5.

After all SDQ segments have been read, and the SDQ array built with all store-quantity pairs for the current item, the PO1 loop ends. Before we begin to loop on the next PO1 group, another loop is kicked off on the SDQ array.

This is where we build our IDocs, one for each store, regardless of the number of items ordered by each store.

At the top of each loop of the SDQ array, a Java hash map object or other array or internal table — STORE_MAP — is searched for the store GLN being processed by the current loop pass.

If the store GLN isn't found in STORE_MAP, an IDoc for that store hasn't been created. A new IDoc will be built for store.

First we copy each field of the IDOC_HDR record into our new IDoc and move the store's GLN into the KA1_WE_LIFNR field. We assign a new IDoc header index — STOREIDX — to the IDOC_HDR record being built for the store. All further items ordered by that store will be written to the same IDoc, which will be identified by that STOREIDX.

Next the item data that was collected from the current 850 P01 group are written to a new IDOC_ITEM record using index STOREIDX.P01IDX.

The order quantity for that store is then moved from the SDQ array to the P01_MENGE field in the new IDOC_ITEM record being written using index STOREIDX.P01IDX.

Last, but not least, the STOREIDX index and the store GLN are appended to STORE_MAP. If the store GLN is found in STORE_MAP, an IDoc already exists for that store, and we'll append our item data to it.

STOREIDX is pulled from the STORE_MAP key array. The item data collected from the current 850 P01 group is appended to the IDOC_ITEM record of the existing IDOC_HDR record for the store using the STOREIDX we pulled from STORE_MAP to identify it and the current P01IDX item index to write it.

Then the order quantity for the store is read from the SDQ array and written to the P01_MENGE field in the IDOC_ITEM record being appended to the existing IDoc with write index `STOREIDX.P01IDX`.

Pseudo code for this loop follows in Listing 8.1.

```
loop at sdq_array.
  read store_map for key store_GLN.
  if exists get storeIdx from store_map.
    append current IDOC_ITEM_P01 fields to existing IDoc
      using index storeIdx.p01Idx
    move quantity from SDQ array to
     IDOC_ITEM-P01_MENGE using index storeIdx.p01Idx
    copy current IDOC_ITEM_P19_002 fields to existing IDoc
      using index storeIdx.p01Idx
    copy current IDOC_ITEM_P19_001 fields to existing IDoc
      using index storeIdx.p01Idx
    copy current IDOC_ITEM_P19_003 fields to existing IDoc
      using index storeIdx.p01Idx
  else does not exist create new IDoc.
    move store to store_map.
    storeIdx = last storeIdx + 1.
    move storeIdx to store_map.
    copy current IDOC_HDR_DC40 fields to new IDoc
      using index storeIdx.
    copy current IDOC_HDR_K01 fields to new IDoc
      using index storeIdx
    copy current IDOC_HDR_K14 fields to new IDoc
      using storeIdx
    copy current IDOC_HDR_K03 fields to new IDoc
      using index storeIdx
    copy current IDOC_HDR_KA1_LF fields to new IDoc
      using index storeIdx
    move WE to IDOC_HDR-KA1_WE in new IDoc
      using index storeIdx
    move store GLN from current SDQ array to
      IDOC_HDR-KAI_WE_LIFNR in new IDoc
      using index storeIdx
    copy current IDOC_HDR_K02 fields to new IDoc
      using index hdrIdx.p01Idx
    append current IDOC_ITEM_P01 fields to new IDoc
    using index storeIdx.p01Idx
```

```
    move quantity from current SDQ array to
      IDOC_ITEM-P01_MENGE using index storeIdx.p01Idx
    copy current IDOC_ITEM_P19_002 fields to new IDoc
      using index storeIdx.p01Idx
    copy current IDOC_ITEM_P19_001 fields to new IDoc
      using index storeIdx.p01Idx
    copy current IDOC_ITEM_P19_003 fields to new IDoc
      using index storeIdx.p01Idx
  end if.
endloop
```

Listing 8.1 Pseudo Code to Support Building One IDoc for Each Store and Quantity Pair in Item-Level SDQ Segments

These IDocs are being assembled into a file in memory. The index identifies the current location of the IDoc and its segments.

- The `STOREIDX` index identifies the IDoc and links together all of its segments. The header segments of the IDoc are written only once but copied into each new IDoc for each store.
- The `P01IDX` index identifies the current E1EDP01 group segments being appended to the existing header and items of the IDoc.

This allows us to write only one ORDERS IDoc for each location with the store's GLN in the header-level E1EDKA1_WE segment and each product being ordered for it occupying its own E1EDP01 group.

8.3.6 Duplicate Checking Enhancement

The enchancement for checking duplicates is a straightforward user exit that prevents duplicate custom POs from posting to sales orders in SAP.

The issue here is that if the same PO posts twice for the same store location, double the number of goods that were ordered could be shipped, resulting in higher shipment costs, increased returns, and poor customer service.

There are times when we need to post a customer PO a second time because of errors in an initial transmission that have been fixed. But this scenario will be known in advance, and all sales orders that posted against the initial transmission will be deleted.

This enhancement is aimed at eliminating the accidental reposting of a PO that has already posted. The basic rule is that no customer PO should be allowed to post twice to an SAP sales order for the same ship-to partner.

The code will be written in `CUSTOMER-FUNCTION '011'` in the IDoc processing function `IDOC_INPUT_ORDERS`.

We'll need to create two objects to enable this enhancement:

- Error message flagging the duplicate PO, ship-to partner, and sales order number
- CMOD modification project to code the exit

Create Error Message

We previously created our custom messages in the 900 and above range in standard message class IDOC_ADAPTER. To create our new message, follow these steps:

1. Go to the SAP Repository with Transaction SE80.
2. Click Repository Information System.
3. Open the folder Other Objects.
4. Double-click Message Classes, and enter "IDOC_ADAPTER" in the Standard Selections screen. Execute to open the Repository Info System: Message classes Find screen.
5. Double-click IDOC_ADAPTER to load the Message Maintenance screen, and click the Messages tab.
6. Click Display <-> Change, and scroll down to message 902, which should be blank.
7. Enter the following message into 902:

   Duplicate PO & for ship-to & in sales order &.
8. Save the message, and assign it to a change request.

Create Modification Project

To create the modification project, follow these steps:

1. Go to CMOD, and enter project name "ZEDISOO1". Click Create.

2. Enter a description in the SHORT TEXT field of the ATTRIBUTES screen. Save the project and assign it to a change request.
3. Click ENHANCEMENT ASSIGNMENTS, and get enhancement VEDA0001 (see Figure 8.3).
4. Click COMPONENTS, and select function set `EXIT_SAPLVEDA_011` with function `CUSTOMER-FUNCTION '011'`.

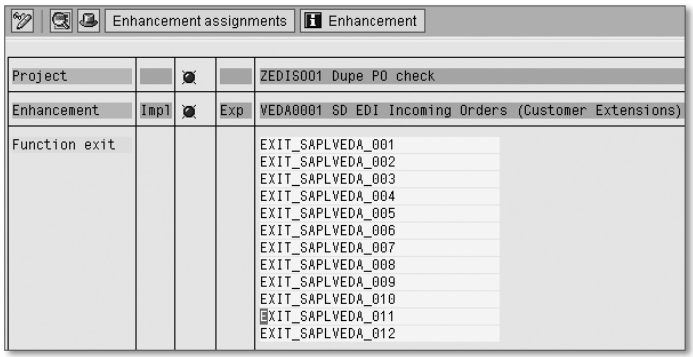


Figure 8.3 Enhancement VEDA0001

5. Double-click the exit name to open the exit in the source code editor of the Function Builder (see Figure 8.4).

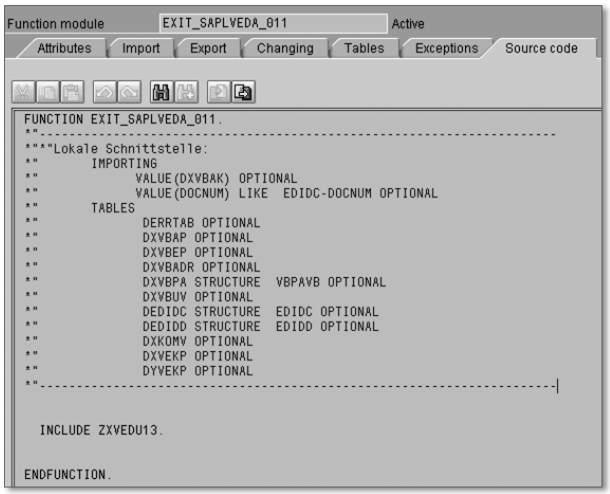


Figure 8.4 EXIT_SAPLVEDA_011 in the Source Code Editor

6. Double-click `ZXVEDU13` to create the include program that we'll use to write our code. The system will throw up the message:

   Program names ZX… are reserved for includes of exit function groups

7. Press ↵ to bypass the message and create the program. You'll be prompted to assign it to a change request.
8. The ABAP Editor opens to a blank screen. The project must be activated after completing the code. Click the activation icon at the top of the screen in the CMOD selection screen, or select menu option PROJECT • ACTIVATE PROJECT.

Program Flow

`CUSTOMER-FUNCTION '011'` is called after the ORDERS IDoc has been processed and confirmed and before the call transaction to Transaction VA01 to create the sales order.

This customer function is the last opportunity to check application data pulled from the IDoc for errors and to pass those errors to the IDoc status record before creating the sales order through the call transaction.

There are 2 import and 11 table parameters available to the exit. For our purposes, we are only interested in

- **Import parameter DXVBAK**
  Brings sales order header data into the exit. Has the structure of VBAK, the sales order header table, with a number of additional fields.
- **Table parameter DERRTAB**
  Collects error messages to pass to the IDoc status record.
- **Table parameter XVBPA**
  Holds SAP partner type and ID data for all partners in the sales order. This internal table is populated by partner determination processing during the EDPAR read.

The sold-to partner and customer PO number are pulled from fields KUNNR and BSTKD in XVBAK. The ship-to partner is pulled from field KUNNR in XVBPA where the qualifier PARVW = WE.

The exit then reads table VBAK in SAP with the PO number and sold-to partner as the key. The SQL will look like Listing 8.2, where `S_VBELN` is a variable to hold the sales order number.

```
select single vbeln into s_vbeln from vbak
    where vkorg = xvbak-vkorg
      and vtweg = xvbak-vtweg
      and spart = xvbak-spart
      and bstnk = xvbak-bstdk
      and kunnr = xvbak-kunnr.
if sy-subrc <> 0.
  exit. * No dupe end processing.
else.
  check for ship-to partner.
endif.
```

Listing 8.2 Selecting Sales Order from VBAK for Dupe PO Exit Check

If there is no hit, the PO has not posted, and exit processing ends.

If there is a hit, table VBPA is read with the sales order number pulled from VBAK and the ship-to number from XVBPA. The code will look something like Listing 8.3.

```
read table xvbpa with key parvw = 'WE'.
if sy-subrc = 0.
  s_kunnr = xvbpa-kunnr
else.
  exit. * No dupe end processing.
endif.
select single kunnr into s_kunnr from vbpa
    where vbeln = s_vbeln
      and parvw = 'WE'
      and posnr = '000000'
      and kunnr = s_kunnr.
if sy-subrc <> 0.
  exit. * No dupe end processing.
else.
  * write error message to derrtab.
endif.
```

Listing 8.3 Determining If the Ship-To Partner Exists for the Sales Order in Table VBPA

If there is no hit, the incoming PO has not yet posted for that ship-to partner. There is no dupe, and exit processing ends.

If there is a hit, the customer PO has already posted for that ship-to partner. Next we check if ORDRSP IDoc has been generated. To do this, we call a function —

NREL_GET_NEIGHBOURHOOD — that checks all objects linked to the sales order we pulled from VBAK. If we find an ORDRSP IDoc, a dupe PO error is raised. The function call is

```
CALL FUNCTION 'NREL_GET_NEIGHBOURHOOD'
  EXPORTING
    IS_OBJECT = s_object
  TABLES
    LINKS     = i_links.
```

s_object has the structure of Data Dictionary type BORIDENT. It needs two values:

- OBJKEY
  Object key: the sales order number with leading 0s.
- OBJTYPE
  Object type: BUS2032, the business object for sales order.

The object links are returned in internal table I_LINKS with the structure of Data Dictionary type RELGRAPHLK. The object type IDOC would be in field OBJTYPE_B, and the number in OBJKEY_B. You then use OBJKEY_B to read table EDIDC and check field MESTYP for message type ORDRSP.

If there's no hit, we have no dupe, and exit processing ends. If we have a hit, the PO is treated as a dupe, and an error is raised. Error message variables are then written to internal table DERRTAB, and exit processing ends. The values in Listing 8.4 are passed.

```
DERRTAB-ARBGB = 'IDOC_ADAPTER'.
DERRTAB-CLASS = 'E'.
DERRTAB-MSGNR = '902'.
DERRTAB-MSGV1 = xvbak-bstdk.
DERRTAB-MSGV2 = s_kunnr.
DERRTAB-MSGV3 = s_vbeln.
append DERRTAB.
```

Listing 8.4 Writing an Error Message to DERRTAB

- ARBGB identifies our message class.
- CLASS identifies the error type.
- MSGNR is our custom message number.
- MSGV1 passes the customer PO number to our message.

- MSGV2 passes the customer SAP ship-to partner from VBPA.
- MSGV3 passes the sales order number.

8.4 EDI Configuration in SAP

Now let's look at IDoc configuration for the inbound ORDERS message type for Gordy's Galaxy.

8.4.1 EDPAR Entries

Go to Transaction VOE4, and enter one record for each of Gordy's stores and distribution centers that will be receiving product from Acme Studios, as in Table 8.6.

Field	Value	Description
KUNNR	GRDY01	Gordy sold-to partner from IDoc
PARVW	WE	Partner function ship-to
EXPNR	0098857055556	External partner for ship-to — Gordy's GLN
INPNR	GRDY010987	Internal SAP ship-to partner number

Table 8.6 We'll Need One EDPAR Entry for Each Stores to Handle Inbound ORDERS from Gordy's Galaxy

This maps Gordy's sold-to partner, from the control segment of the IDoc, to Gordy's store GLN from the N1 or SDQ segments of the 850, to the SAP ship-to partner.

Because there are no checks on its data, and EDPAR isn't linked to any other tables or programs, it can be safely loaded with a custom ABAP that inserts data directly into it. It can also be loaded through an LSMW project or CATT script on Transaction VOE4.

8.4.2 EDSDC Entry

Go to Transaction VOE2, and enter the following record for Gordy, as in Table 8.7.

Field	Value	Description
KUNNR	GRDY01	Gordy sold-to partner from IDoc.
LIFNR	564567	Acme vendor number in Gordy's system.
VKORG	0010	Acme sales organization.
VTWEG	010	Distribution channel.
SPART	00	Division.
AUART		Sales order type: if blank, uses the default OR (TA). Can be used to create different order types for different trading partners.

Table 8.7 EDSDC Entry for Gordy's Galaxy

LIFNR is Gordy's number for Acme Studios, although it doesn't have to be. It does need to be a number that always comes in the ORDERS IDoc translated from Gordy's 850 PO.

This table entry maps Gordy's sold-to partner to the SAP sales organization that will exchange EDI data with them.

8.4.3 ZEDIXREF Entries

We don't need to populate ZEDIXREF for the 852 because it doesn't post to SAP. We do need to add the information from Table 8.8 to custom table ZEDIXREF for the inbound 850 interface from Gordy:

Field	Value	Description
DIRECT	2	Direction inbound
STDMES	850	EDI transaction
MESTYP	ORDERS	IDoc message type
IDOCTP	ORDERS05	IDoc basic type
CIMTYP		IDoc extension
SNDPRN	GRDY01	SAP send partner
RCVPRN	DEVCLNT100	SAP receive partner

Table 8.8 ZEDIXREF Entry for the Inbound 850 from Gordy

Field	Value	Description
SNDLAD	99934567999	EDI send trading partner ID
RCVLAD	99999998889	EDI receive trading partner ID

Table 8.8 ZEDIXREF Entry for the Inbound 850 from Gordy (cont.)

8.4.4 Partner Profile

Go to Transaction WE20. The partner profile for the inbound ORDERS interface for Gordy's Galaxy uses the values in Table 8.9.

Screen	Group	Parameter	Value
Partner Header		Partner number	GRDY01
		Partn.Type	KU
General		Partn.funct.	SP
		Message type	ORDERS
		Message code	
Inbound Options		Process code	ORDE
	Processing	Trigger by background program	X

Table 8.9 Partner Profile Values for Inbound ORDERS from Gordy

Process code ORDE links to function module `IDOC_INPUT_ORDERS` and message type ORDERS in Transaction WE42. Don't forget to save.

See Chapter 6, Section 6.3.6, Partner Profiles, for further details about creating partner profiles.

Once a sales order has successfully posted to SAP, an order confirmation must be generated and sent to Gordy in an 855 EDI transaction. We will now look at the challenges that this interface presents.

Index

A

B

C

D

E

F

G

I

J

K

L

M

O

P

R

S

T

U

V

W

X