IDocs: A Guide for New Developers

ebook author: Tony Cecchini

The SAP IDoc Technology

The SAP IDoc Technology, is used in ALE, EDI, and 3rd Party Systems Integration scenarios.

For some of my readers this may be trip down memory lane, but for some, the New Developers, I hope to give you the tools you need to understand and demystify the IDoc concept. So.. for you old dogs, think of it as a remedial and chime in with comments and suggestions. let's get started!

What are IDocs?

You have probably heard the term IDoc many times. This blog will help you understand exactly what an IDoc is and what it does.

Let's look at some important facts about **IDocs**.

- The term IDoc stands for **_intermediate document_**. It is simply a data container used to exchange information between any two processes that can understand the semantics of the data.

- An IDoc is created as a result of executing an Outbound <u>ALE</u> or <u>EDI</u> process whereas with an inbound ALE or EDI process, an IDoc serves as input to create an application object in SAP, like a Sales Order or PO.

- IDocs in the SAP system, are stored in database tables. We can use transactions to view, edit, and process them. When an IDoc is generated in the system, a unique number is assigned to it via a Number Range Object. This number is unique within a client. *IDocs*

- IDocs are independent of the direction of data exchange. An inbound and an outbound process can use an IDoc. For example, the ORDERS01 IDoc is used by the Purchasing module to send a purchase order, and is also used by the Sales and Distribution module to accept a sales order. Using this technique avoids creating redundant IDoc types.

The IDoc Interface

How are IDocs used? What is EDI? ALE?   I might be giving my age away here.... but the IDoc interface has been around since release 2.2, when IDocs were initially used in the EDI process. So this is a **PROVEN**, Scalable technology that is used in a wide variety of interfacing requirements.

OK, let me define some of the concepts we will touch on...

EDI Integration (Electronic Data Interchange)

EDI is the electronic exchange of business documents between trading partners in a common industry–standard format, such as ANSI X12 or EDIFACT. Several applications (purchasing, sales, or shipping) in SAP are enabled for EDI. To use EDI, an application first creates an application document, such as a purchase order. Then the EDI interface layer converts the application document (the purchase order) into an IDoc, which is transferred to an EDI subsystem, or PI with an EDI Plug-in. The EDI middleware translates the IDoc into an industry–standard format and then transfers it to a business partner over a network.

EDI Architecture
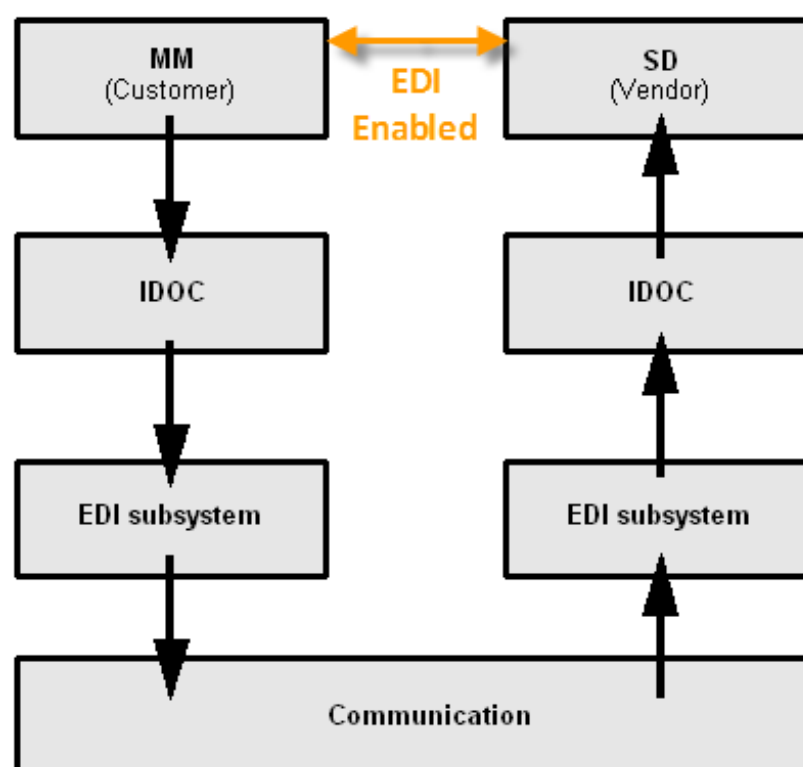
Advantages of EDI process

- Reduced data Entry Errors
– Reduced Processing cycle time
– Availability of data in electronic form
– Reduced Paper Work
– Reduced Cost
– Reduced Inventories and Better Planning
– Standard Means of Communicating

ALE Integration (Application Link Enabling)

ALE enables the exchange of data between two SAP systems. This allows SAP business processes and applications to be distributed across multiple SAP systems. ALE ensures integration in a distributed SAP environment. The IDoc acts as the **data container** . SAP introduced ALE as its initiative to support a distributed yet integrated environment. ALE allows for efficient and reliable communication between distributed processes across physically separate SAP systems to achieve a distributed, yet integrated, logical SAP system. Because ALE architecture is independent of the participating systems, this enabled SAP to use this technique for SAP to non–SAP systems as well. This was huge in the early years of ALE and led to it being widely adopted as a "Best Practice" for communicating with SAP and Non-SAP systems.
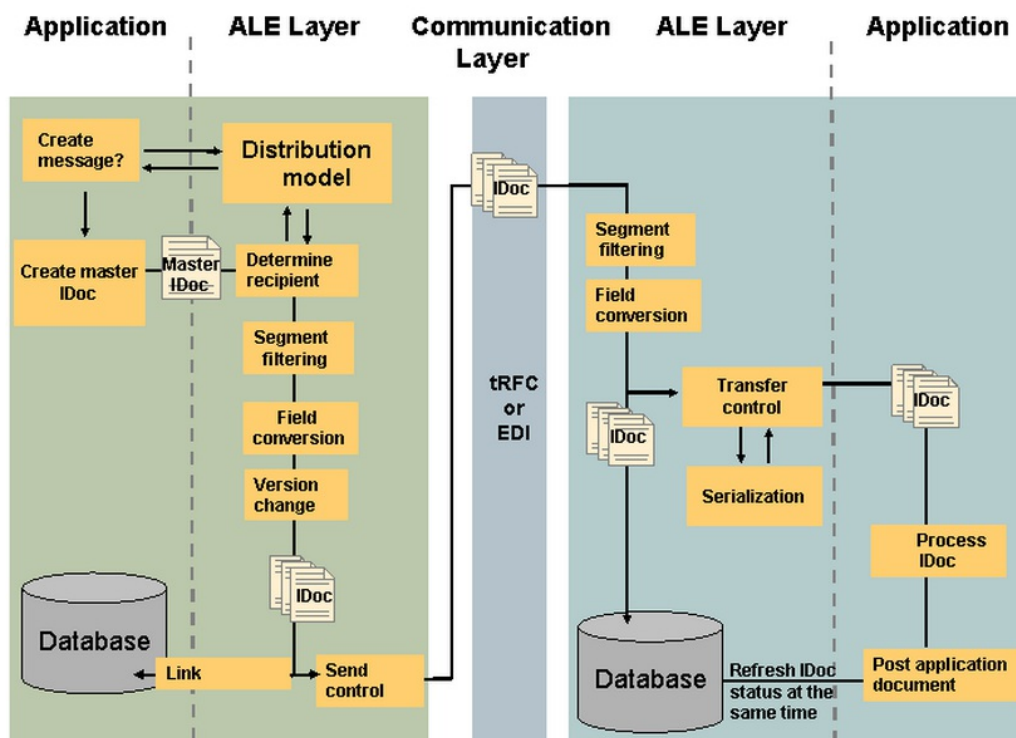
ALE supports

- Distribution of applications between different releases of R/3 Systems
– Continued data exchange after a release upgrade without requiring special maintenance
– Customer-specific extensions.
– Communication interfaces that allow connections to non-SAP systems.

Let's wrap this up by examining some of over-arching benefits of using the IDoc Technology...

Independence from Applications

The biggest advantage of using the IDoc interface is that it's an open interface. It's independent of the internal structure used by SAP to store data and independent of the sending and receiving applications. Any application that can understand the syntax and semantics of the data can use the IDoc interface.

Exception Handling via Workflow

Handling exceptions is always very important and usually a second thought in most designs. If you have designed sophisticated applications in the past, you can, no doubt, relate to the agony of designing a consistent means of logging errors and exceptions across the board and then developing tools to display that information.

IDoc Monitoring

Well with IDocs, you get comprehensive information about the processing. Several standard tools are available to display the logged information. In particular, SAP uses the workflow technology to route an error intelligently to the right person so they can see what happened, why, and how to proceed. By using the IDoc interface, you automatically take advantage of this exception-handling process. This is both for standard and for your custom IDocs. No extra code required.

IDoc Modification and Enhancement

Using standard tools we will discuss in later blogs, (the IDoc editor and Segment editor), you can either enhance standard SAP IDocs or create new IDocs in the system to support custom interfaces. Your newly developed IDocs integrate seamlessly into the standard EDI interface because they are developed using standard tools provided by the system. IDocs developed in this manner become available in the standard list of SAP IDocs and can take advantage of all the tools designed for standard IDocs, such as IDoc monitoring, error handling, and archiving. We will talk at length about each one of these in turn in later.

Summary

So in summary, IDocs act as containers for data exchanged between two applications. The IDoc interface is functionally rich and provides a robust environment for interfacing SAP with SAP, as well as with external applications. Using the IDoc interface for integrating external applications with the SAP system offers several benefits, such as a thoroughly documented interface, independence of the application product, numerous testing and troubleshooting tools, and a sophisticated means of error handling via workflow.

The SAP IDoc Technology

Let us continue our look at SAP **IDocs** and the IDoc Technology by exploring the architecture of an IDoc. The architecture can be best explained by looking at an IDoc's definition and run–time components.
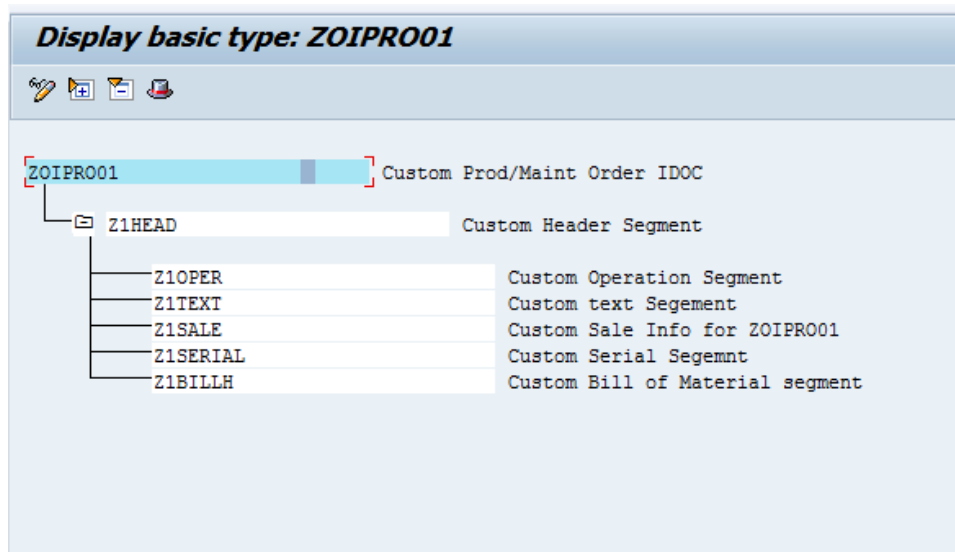
IDoc Definition Components

Each of the following sections begins with a formal definition of the component.

Basic IDoc Type

Basic IDoc type defines the structure and format of the business document that is to be exchanged between two systems. Lets look at an example below for Basic IDoc Type ZOIPRO01.

A basic IDoc type has the following attributes – (You can display a Basic IDoc Type in transaction WE30).

IDocs: A Guide for New Developers

The first attribute we will look at is **NAME.** This can be up to a thirty–character name. Custom IDoc types always start with a Z. The last two characters are the version number. After a basic IDoc type is released and you move to a newer version of the SAP system, any changes to the structure of the basic IDoc type will create a new basic IDoc type. In general, the version number is incremented by one. So in the example above we have a custom IDoc type named ZOIPRO01. If we changed this after it was released, it would be named ZOIPRO02...etc..etc. You can see SAP doing the same thing if you look at the delivered Basic IDoc Types ORDERS01, ORDERS02, ORDERS03, …. ORDERS06…etc

Next you will see a list of **SEGMENTS**. These segments make up the IDoc structure.   We have Z1HEAD, Z1OPER, Z1TEXT, Z1SALE, Z1SERIAL and Z1BILLH. These segments may have a defined **HIERARCHY**. The hierarchy of segments specifies the physical sequence and any parent–child relationship in the segments. A parent–child relationship signifies that the child segment cannot exist without the parent. In our example above the segment Z1OPER is a child of the parent segment Z1HEAD. Therefore an instance of the CHILD cannot occur unless it follows a related PARENT segment.

Segments

A segment defines the format and structure of a data record. Segments are reusable components, which means they can be used in more than one IDoc type. A segment consists of various fields that represent data in a data record. Data elements can be of two types: positional or based on qualifiers.
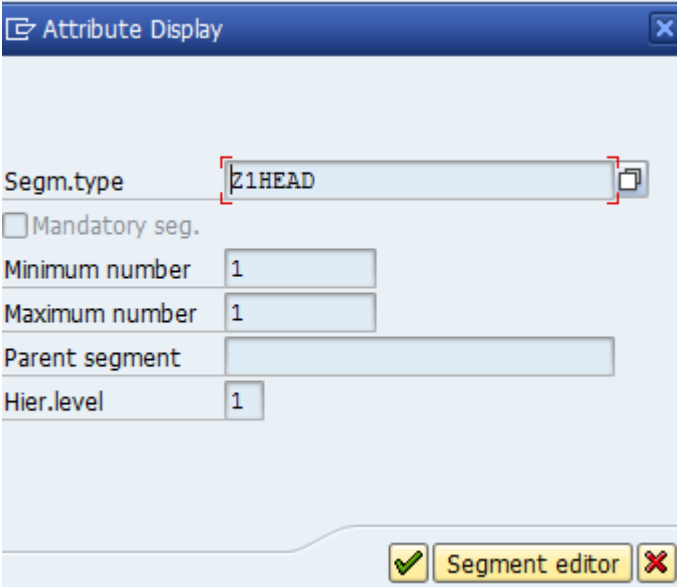
A positional data field occupies a fixed position in an IDoc. These fields always occur in the same position shown in the segment.

For example, assume a segment has a date field for three dates: the delivery date, the goods issue date, and the order creation date. Instead of creating three separate fields and assigning a fixed position to each one, the three fields can be represented using two fields a qualifier field and a date field. The qualifier field identifies the type of date, and the date field contains the date. You will usually see this when an IDoc represents an EDI Message. This is so we can match the IDoc fields to the EDI Message Implementation Guide from the partner we are trading with.

A field can also be based on a qualifier, in which case the value represented in a field is tied to the qualifier.

OK, lets look at the attributes of an IDoc Segment. If you "double-click" on a segment ; a popup will be displayed showing you specific attributes about that segment. Lets "double-click" on Z1HEAD first.

Data Field

I'd like you to notice is each segment has an attribute that defines whether the segment is **OPTIONAL or MANDATORY**. In the example, Z1HEAD is NOT a mandatory segment. If the CHECK-BOX was checked, the of course it would be. Each segment also has an attribute that defines the **MINIMUM** and the **MAXIMUM** number of times a data record corresponding to a segment can exist in an IDoc.

A data field represents a single data item that is used in a segment. All data field values must be alphanumeric values. The valid data types for a field are CHAR, CLNT, CUKY, DATS, LANG, and NUMC. If you press the segment editor button, which is transaction WE31, you will see a list of the data fields defined for a segment.

P...	Field Name	Data element	ISO...	E...	
1	ORDNUM	CHAR12	☐	12	
2	ORDTYP	CHAR4	☐	4	
3	ORDCAT	NUMC2	☐	2	
4	PLANT	CHAR4	☐	4	
5	PRODNO	CHAR18	☐	18	
6	LOTNO	CHAR10	☐	10	
7	TOTQTY	CHAR14	☐	14	
8	TOTUOM	CHAR14	☐	14	
9	BOM	CHAR8	☐	8	
10	SPORDNO	CHAR18	☐	18	
11	SPRODFLAG	CHAR1	☐	1	
12	ARELDATE	DATS	☐	8	
13	SSTRDATE	DATS	☐	8	
14	SSTRTIME	TIMS	☐	6	
15	SFINDATE	DATS	☐	8	

IDoc RUN Time Components

An IDoc has a record–oriented structure, which is very much like the record structure in a file . At run time the following events occur.

An IDoc is an instance of an IDoc type. What does this mean? Well, above we created or viewed an existing DEFINITION of a Basic IDoc Type. When and IDoc is actually created in the system, it is assigned an IDoc Number from a Number Range Object and viola it is instantiated or externalized as an object we can work with in ECC.

- A unique IDoc number is allocated. (via a Number Range Object)
– One control record is attached to the IDoc.
– Status records are attached.
– Syntax rules are checked.

Lest look at the graphic below. It represents the 3 types of records that make up an IDoc. We will go into each one individually in a moment.
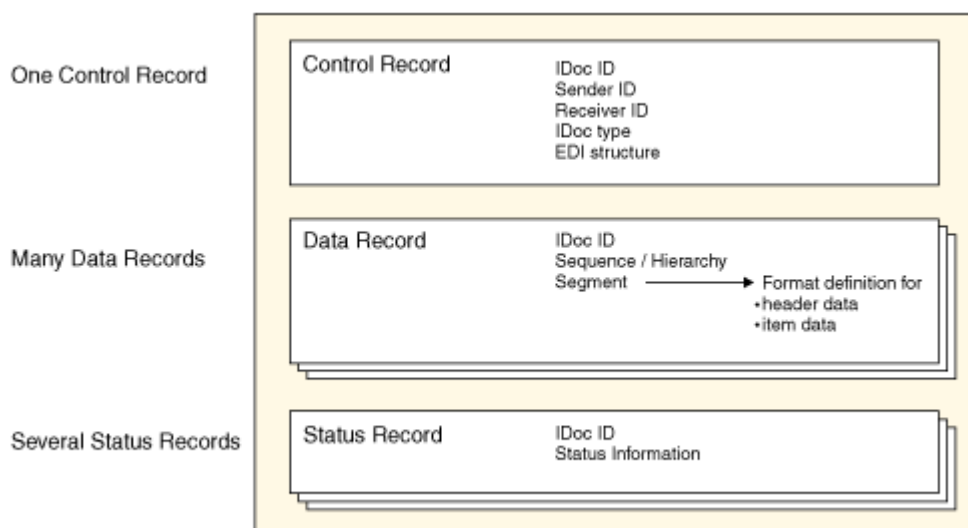
Technical Description of the IDoc Format

IDoc CONTROL RECORD

An IDoc consists of a header record, any number of lines of application data (application records), and any number of status records per IDoc.

IDoc DATA RECORDS

The header contains general information about which data is supposed to be transferred, who is the sender, and who is the receiver. This information basically includes the IDoc number, sender and receiver information, and information such as the message type it represents and the IDoc type. The control record data is stored in the EDIDC table. The key to this table is the IDoc number.

The data records contain business-related information. To make sure the technical format is independent of the business object and can also be understood by non-SAP systems, the content of each data record is stored as a string of 1000 characters. This character string is preceded by a control area containing information about how to interpret the 1000 characters. Data records for IDocs from version 4.0 on are stored in the EDID4 table.

IDoc STATUS RECORDS

Status records contain information about the previous statuses of the IDoc, such as "successfully created" or "successfully posted." Status records aren't transferred between partner systems; that is, both the sender and receiver keep their own status records in their respective systems. The format of the status record is supplied by SAP, and the formats are stored in the EDIDS table. The key for this table is the IDoc number, date and time a message was logged, and a status counter.

Finally lets close this month's Blog by looking at what the IDoc SYNTAX Check does.

When any IDoc is created, it goes through a syntax check to ensure its integrity. The syntax of an IDoc is governed by the definition of its IDoc type. Remember we can define or view this in WE30 as we did at the beginning of this writing. The syntax rules checked for an IDoc are as follows.

Syntax Rules for an IDoc

- Only valid segments as defined in the IDoc type are allowed.
– Segments specified as mandatory must exist.
– A data record cannot exceed the maximum number of repetitions defined for the segment type.
– Segments must occur in the same physical sequence defined in the IDoc structure. For example, a child segment cannot exist without its parent segment. A parent segment, however, can exist without a child segment.

Summary

So in summary, an IDoc type represents the definition component of an IDoc. An IDoc type is a version–controlled object that defines a list of permitted segments for an IDoc and the hierarchical arrangement of those segments. The IDoc type effectively defines the syntax of an IDoc.

An IDoc is the run–time instance of an IDoc type. An IDoc consists of a control record, several data records,and a list of status records. The control record defines control information such as sender and receiver information. The data records contain the application data that is to be transferred via IDocs. The status records contain status information (success or failure) recorded at each point in the process.

The SAP IDoc Technology

In this month we will continue our look at SAP **IDocs** and the IDoc Technology by exploring IDoc extensions and enhancements.

Why do We Need an Extended IDoc?

Standard SAP sends out or receives in data through IDocs using standard delivered Segments, Message Types and fields. But sometimes, these fields are not sufficient for a specific end-to-end business scenario as far as data transfer is concerned. So in such scenarios, we can add new segments with completely new structures to the standard IDoc as an extension. We create a brand new structure and insert it into existing delivered IDoc structure creating a whole new IDoc satisfying the requirement. This new IDoc is called an Extended IDoc.

As an example, lets take a scenario in billing, where we already have a predefined **IDoc type** 'INVOIC02'. But the requirement is to transfer an additional structure containing the fields VBRK-KTGRD (Account assignment group for this customer) and VBRK-MANSP (Dunning block). In order to fulfill this requirement, we need to create a new segment structure, add two additional fields to it, then add it as an extension to the existing IDoc Type 'INVOIC02. Sound good? OK lets take each step in turn .....

Create a new IDoc Segment using transaction WE31

Our first task is to create a segment with the two new fields VBRK-KTGRD (Account assignment group for this customer) and VBRK-MANSP (Dunning block). In this transaction we create a segment type. This segment type has two fields KTGRD and MANSP as specified from VBRK table. This segment will be used in the final extended IDoc.

Lest take a look at the standard IDoc INVOIC02 using transaction WE30. We want to add our new segment under the first header segment E1EDK01 as a CHILD segment.
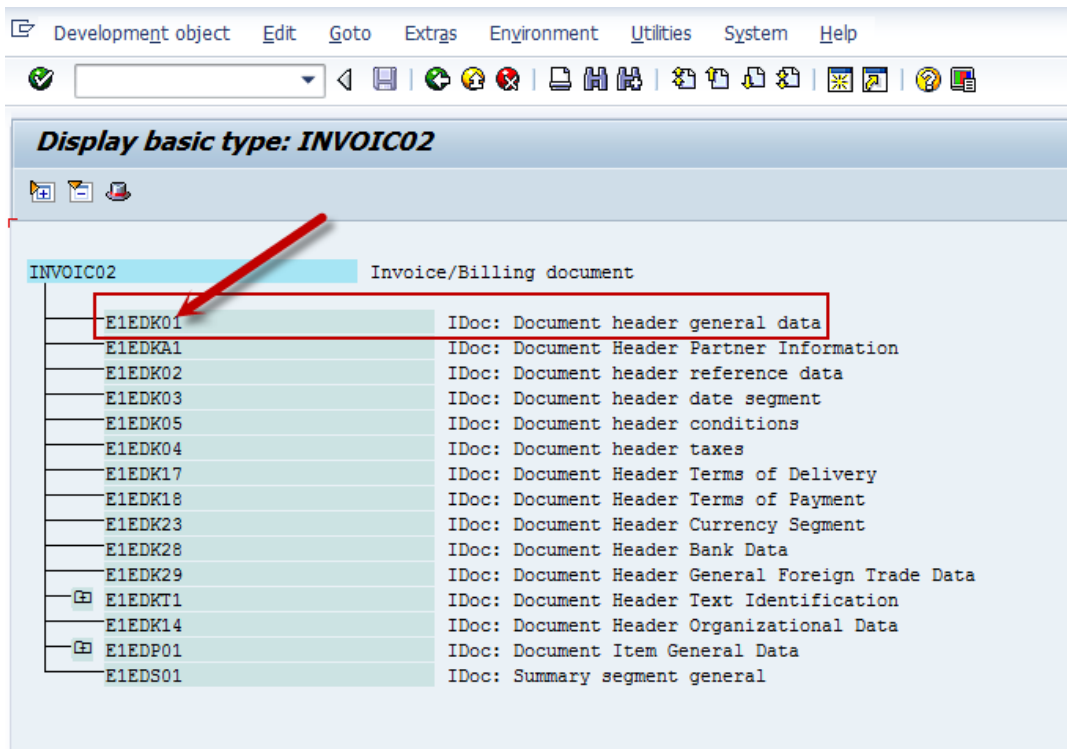
OK, now lets use WE31 to create a new Segment ZE1EDK01.

IDocs: A Guide for New Developers

Next we enter a description, add our two new fields and hit save.

When you hit SAVE, you will get a popup as below. Just put in your User-id. The next popup will be for the transport system, for now, lets just make this a LOCAL OBJECT.

IDocs: A Guide for New Developers

At this point we can RELEASE the segment by following the menu path below. If for some reason you need to change the segment after it has been released, go back and cancel the release and make your changes.
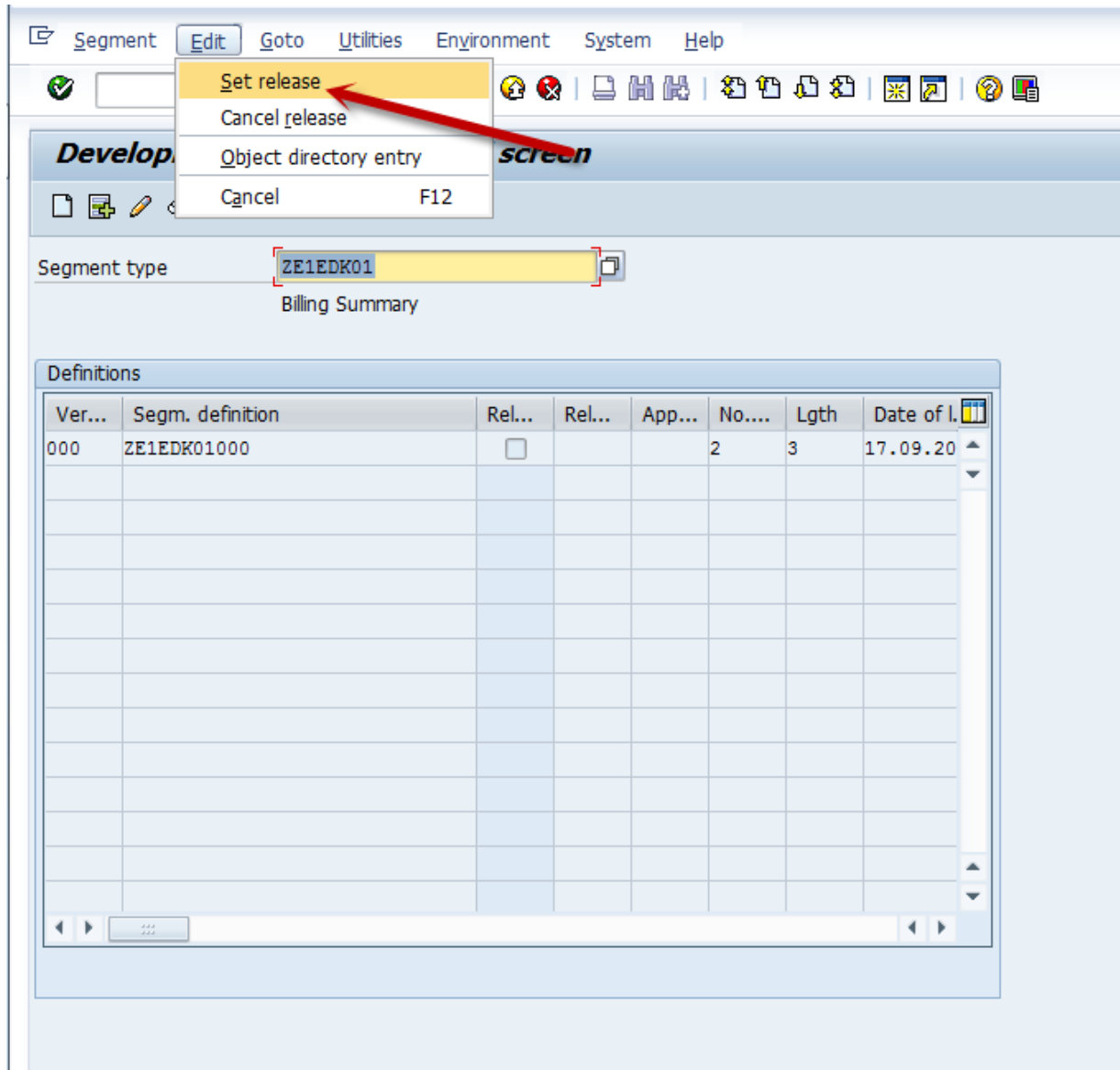
Create a new IDoc Type (Extension) using transaction WE30

Now we can use transaction WE30 to create the new Extension Type ZINVOIC02. Fill in the Obj. Name and hit CREATE.

Please be sure to mark the new IDoc type as an extension!
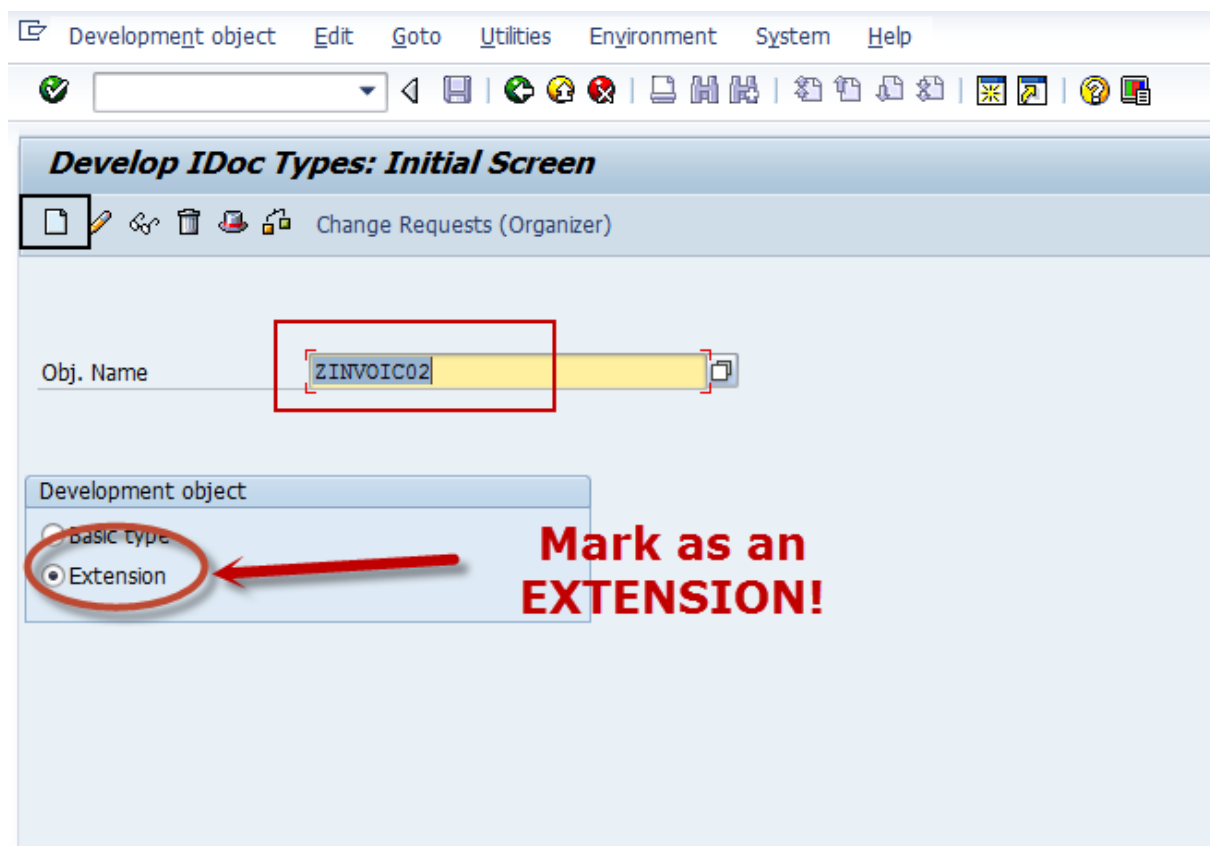
When you hit the CREATE button, a new popup screen will appear. Fill in the description and enter the IDoc Type we want to extend and hit the GREEN Check.

For us this will be INVOIC02.

You will get a screen showing our extension ZINVOIC02 and a set of segments belonging to the standard IDoc INVOIC02. Since the extension has VBRK-KTGRD and VBRK-MANSP and they belong to "HEADER" tab in SAP Billing transaction VF02 the extension is done for the relevant segment type E1EDK01 related to "Header General Data". Click on the E1EDK01 Segment to place focus on that segment and click the create button.
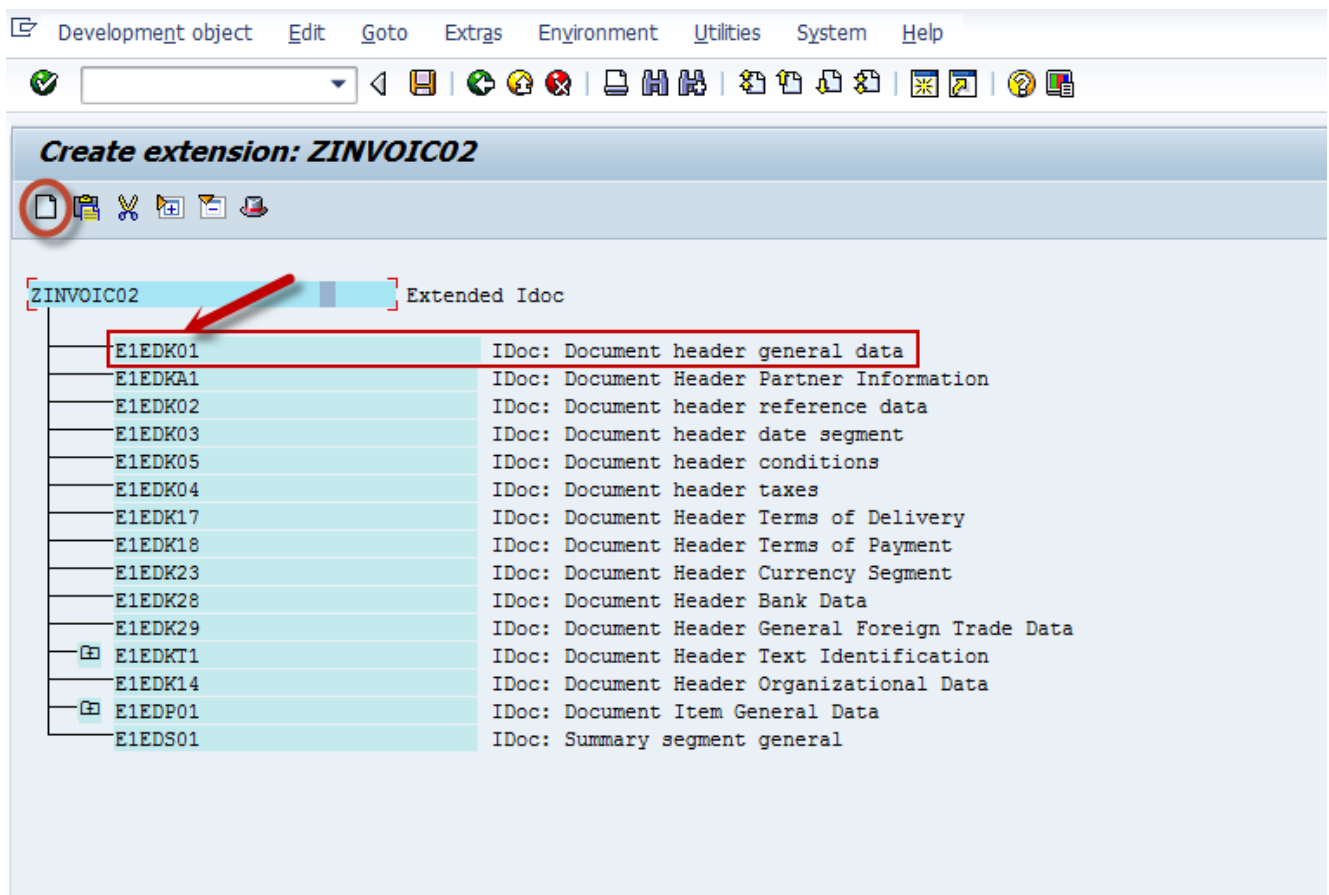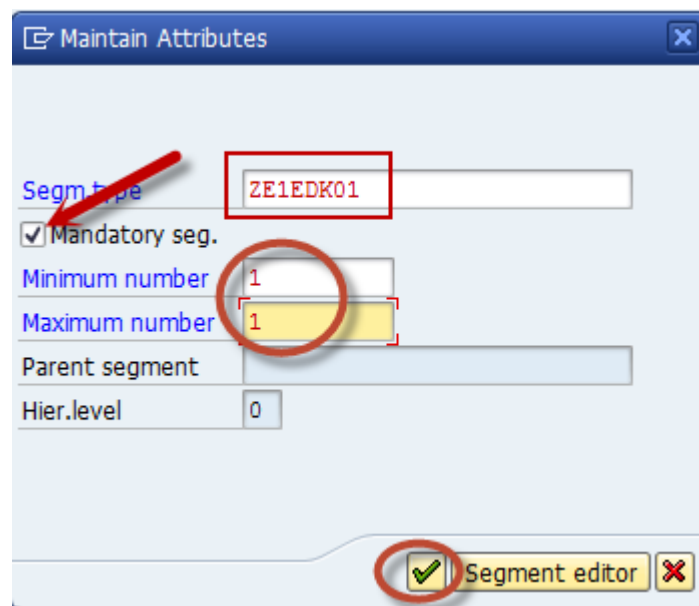
When you hit the CREATE button, yet another new set screens will appear. The first Pop-Up is information only and you can ENTER past this. You will see our new segment will be a CHILD segment to the PARENT segment E1EDK01.
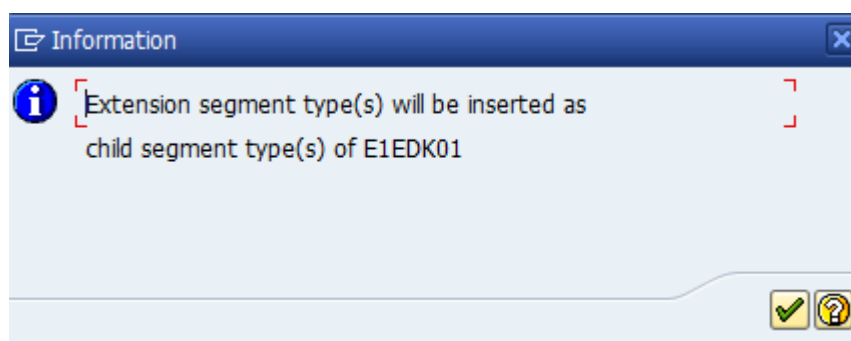
The next screen is for identifying our new segment we created way back using transaction WE31, and, maintaining the attributes for the new segment. The attributes include whether it is a mandatory Segment. The Maximum and minimum numbers specify the number of times the segment can be repeated in sequence. The Hierarchy level suggests the parent/child relationship. Segments which have a parent segment (like ours) have a hierarchy level which is one higher than that of their parent.

We will use our new custom Segment ZE1EDK01, set it to mandatory, and use a minimum and maximum of 1. Enter these values and click the GREEN check button.

Almost there! Now after you hit the GREEN check, you will have the distinct privilege of seeing the result of your hard work. The actual Extension Type with our new segment shown. All we need to do now is hit the SAVE button. Again, for our purposes we will make this a LOCAL object so no transports are necessary.
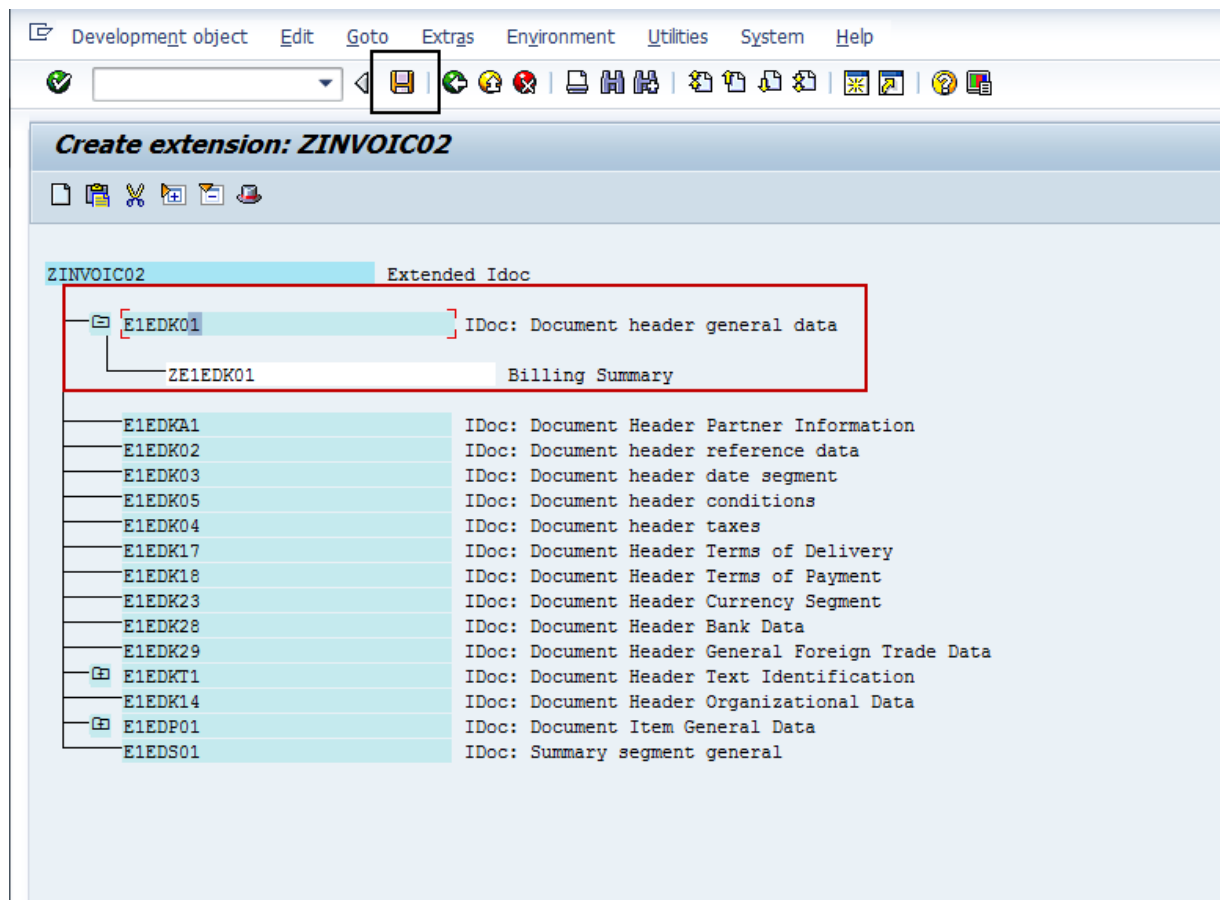
The last and final step is set the RELEASE flag on this extension. Once you have done this the **IDoc Extension** is ready to use!
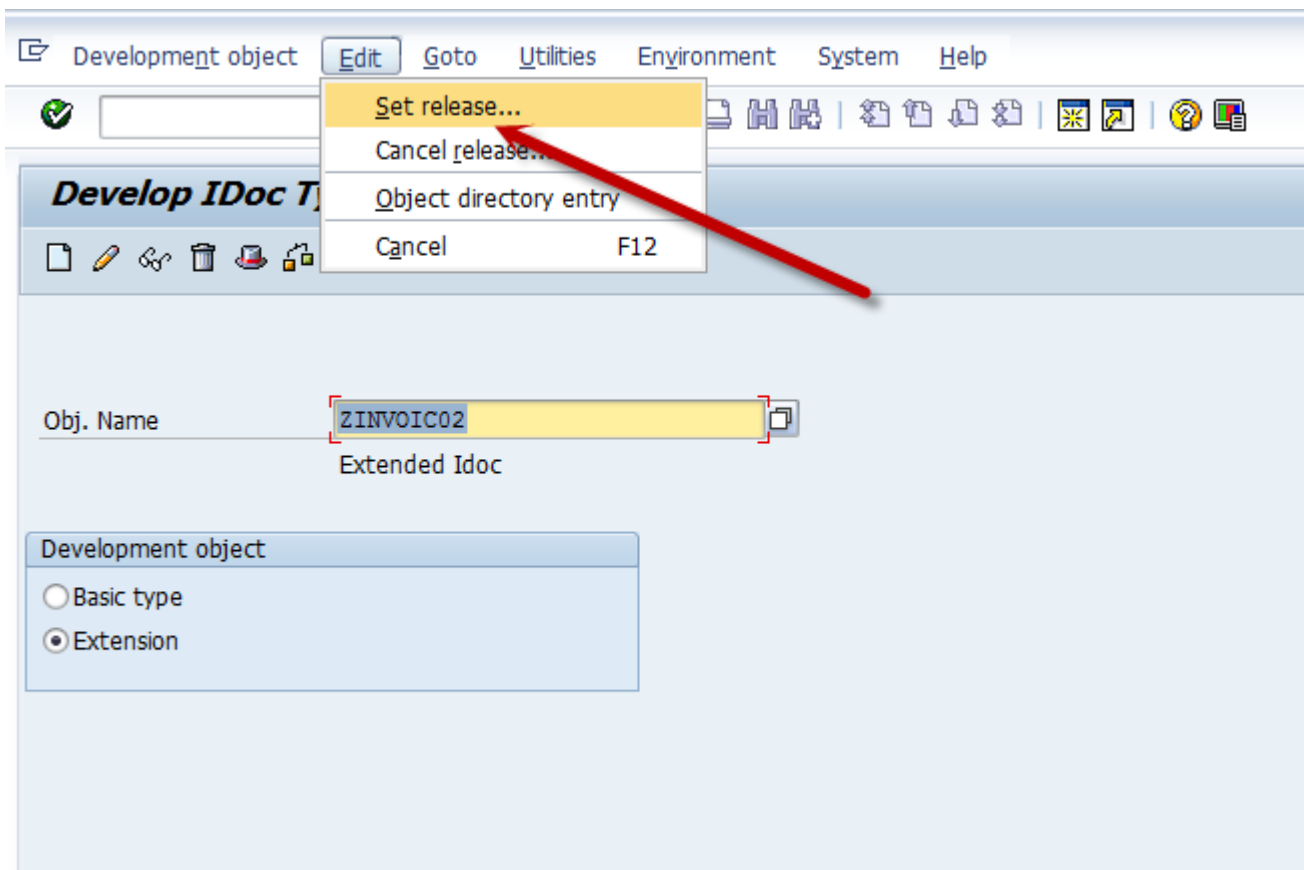
Summary

So in summary, we needed a IDoc Type that was different than the delivered IDoc Type of INVOIC02. We needed to add new fields. We achieved this by creating a custom segment, adding our fields to it, then inserting into the delivered IDoc Type as a NEW IDoc Extension.

The SAP IDoc Technology

We will continue our look at SAP **IDocs** and the IDoc Technology by exploring how we can use the new custom **IDoc extension** we created in the last Chapter.

Create an IDoc Message Type

The message type determines the technical structure of the message, along with the data contained. Through configuration it will also determine the process flow involved in a "Distributed environment". The Message Type controls Process Code, which in turn drives a Function Module to derive the content of the message in an OUTBOUND scenario or execute the desired SAP process in an INBOUND scenario. Finally, it also controls how the IDocs will be processed (batch, immediately etc). The former is accomplished in conjunction with Partner Profile configuration which we will discuss later in the blog.

For starters – there exist many delivered standard _IDoc message_ types predefined by SAP. Here are a few of them as an example……

SAP Object	IDoc Message Type
CUSTOMER	DEBMAS
VENDOR	CREMAS
MATERIAL	MATMAS
SALES ORDER	ORDRSP
PURCHASE ORDER	ORDERS
INVOICE	INVOIC

We can also create a customized logical message type according to our specifc requirements. Since we are using the invoice here we could use the existing message type INVOIC, but I want to show you how to create your own logical message type, so we will create a custom one. We will use ZINVOIC as our custom message type.

Create a new Custom IDoc Message Type using transaction WE81

Execute transaction WE81. Enter into Change mode by clicking the button. Hit New Entries as shown in the screen below.
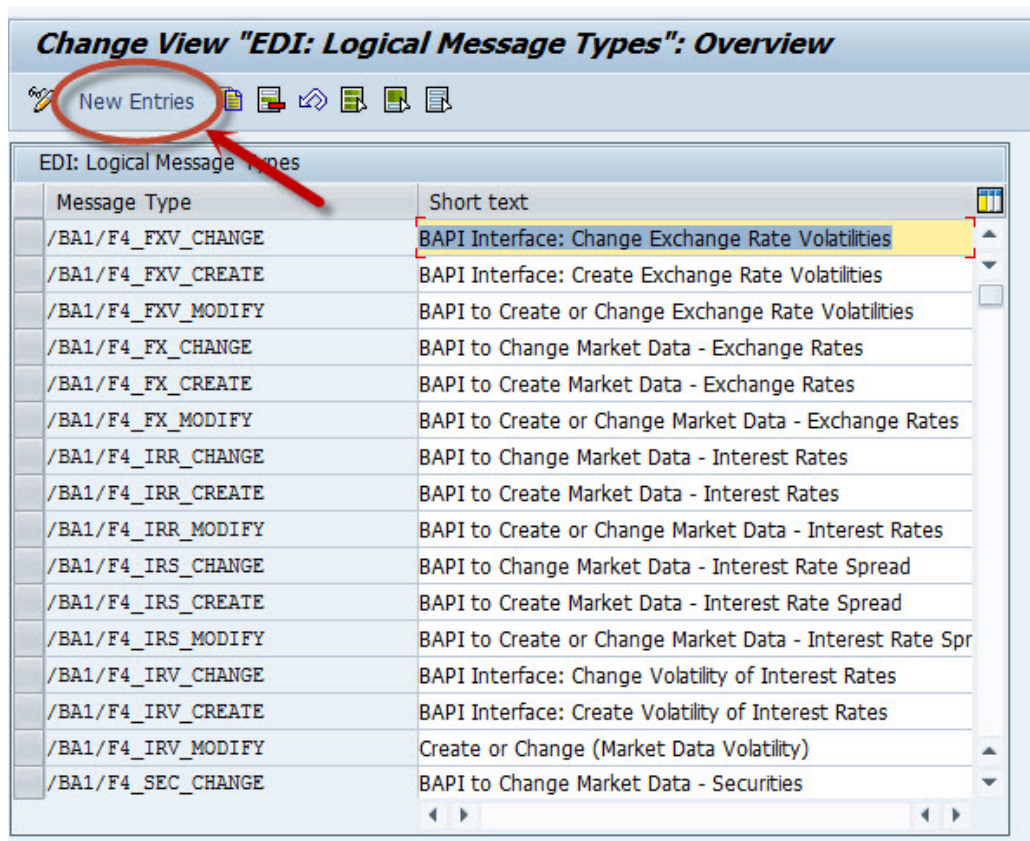
You will then be able to enter our NEW Custom Message Type along with a description and hit SAVE . See below...
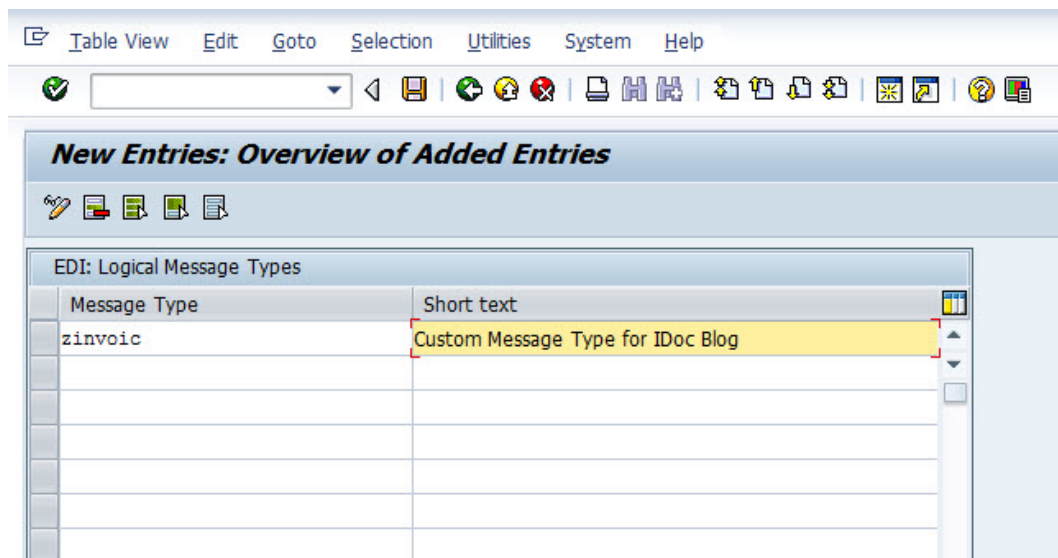
When you hit SAVE, you will be prompted by SAP to create a transport. For our purposes I am creating a transport with the text "DO NOT TRANSPORT" as this is for education purposes and there is no real requirement behind this.
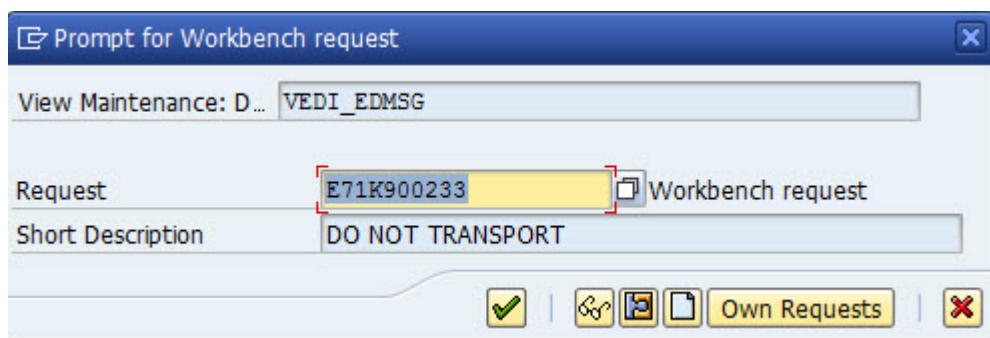
That is it! We have created a custom IDoc Message type. Now we have to link this new message type to our IDoc Extension we created earlier.

Create a New IDoc Assignment using transaction WE82

In order to use the new IDoc extension we created, we need to assign or link the Custom Message type to it. In order to this we use transaction WE82. Lets look at the screen below. Enter into Change mode by clicking the CHANGE   button. Hit New Entries.
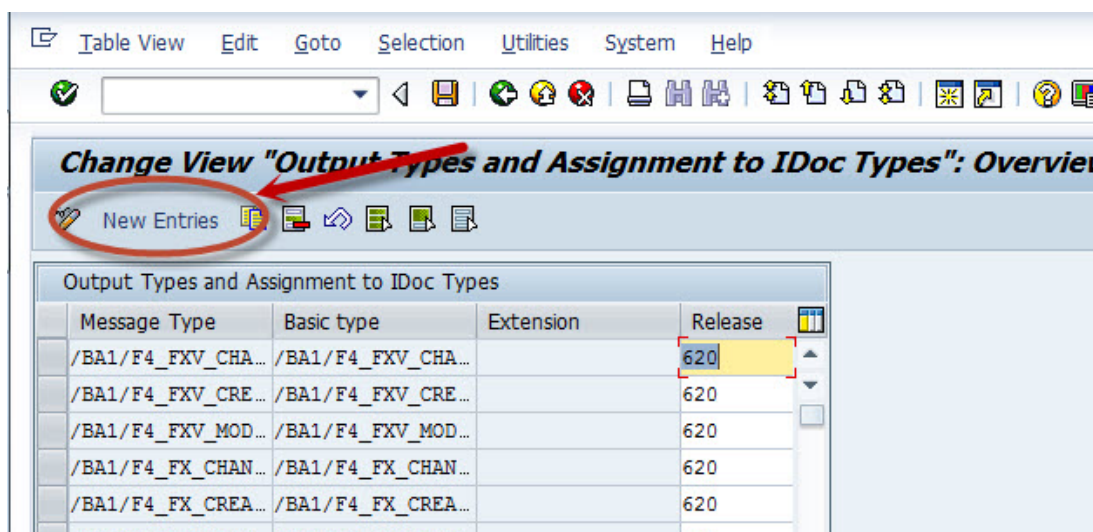
Next we enter in the information. So we enter our custom Message type ZINVOIC, the Basic IDoc (delivered or IDoc Type we extended) INVOIC02 and our Extension ZINVOIC02. For release I put 700, but for you, please enter your ABAP AS release.
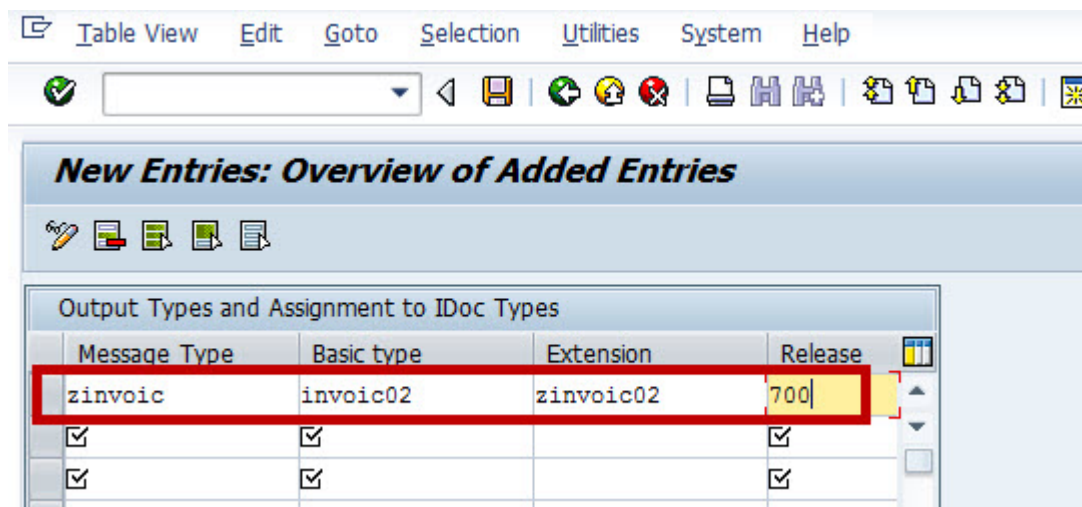
Hit the Save Button and you'll be prompted again to use the Transport we created previously. Go ahead an use it.
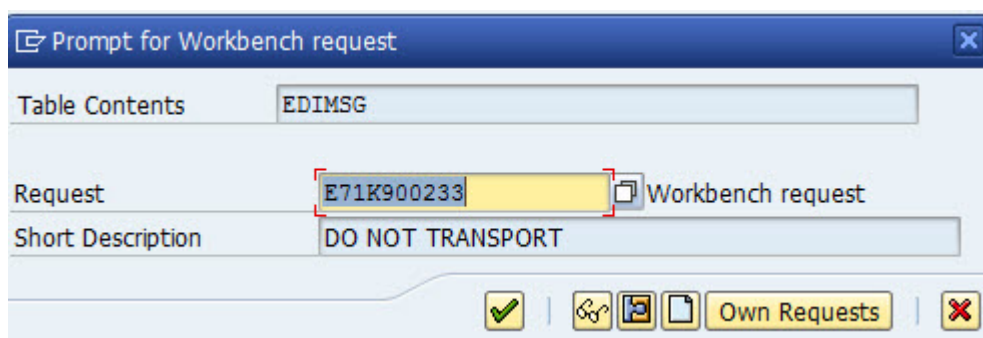
Create an IDoc Entry for the Partner Profile using transaction WE20

The next step in configuring this IDoc Interface would be to create an entry in the Partner Profile using transaction WE20. Lets take a look at the transaction. Please note this will be an Outbound Invoice, so our partner could be a Customer, but will likely be the Middleware your company is using such as PI (Process Integration, formerly know as XI). We will choose an XI Partner. Expand the tree for Customer (see bellow)
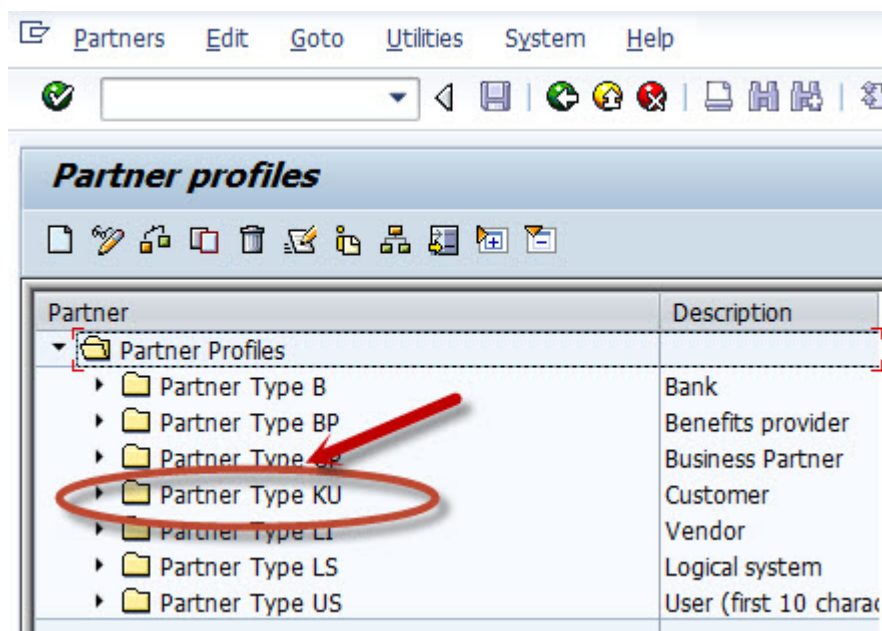
We will Choose the XI_EDI Customer.
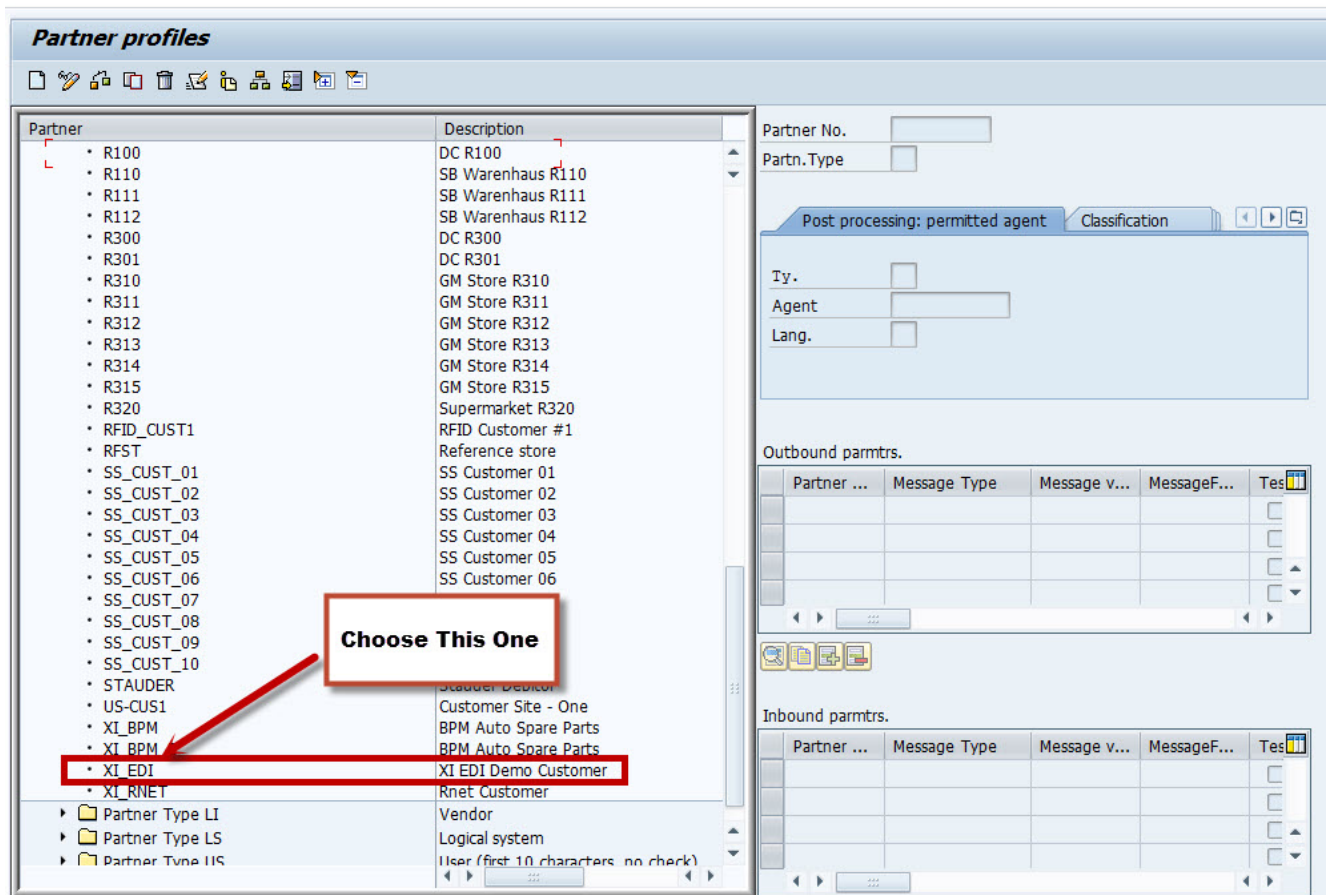
Next, we need to click on the CREATE button to create a new entry. Again,since this is an outbound Interface, we need to be working in the "OUTBOUND PARAMETERS" section shown below. Take note there is already an entry for ORDRSP (Order Response), this is one of the delivered Message Types we discussed at the beginning of this Blog.

Lets add our Entry… and take a look at each of the areas noted in the screen shot in turn.

First, take a look at #1. You will notice we used a Partner Type of BP. This is the Bill-To Pary and makes sense as this is an Invoice.

Next, #2 is the Custom Message type we created. Note the description to the right of the entry.

Next, #3 is the Receiver Port and Package size. The Port Specifies how the IDocs are transferred. There are various technical possibilities for this communication known as port types. While the Package size describes the number of IDocs to send in one package or call to the Middleware.

Next #4 is used to either send the IDoc immediately (Real -Time) or to collect them and use a batch job (RSEOUT00) to send at a defined interval (Near Real – Time).

Finally #5 shows the delivered Basic IDoc and our Extension that has the new Fields.

IDoc Processing and Message Control

To complete this outbound Entry it will be necessary to fill in the Message Control Tab. While it is not my intent to teach this subject in this blog, a rudimentary understanding will be necessary.

What is Message Control ?

Message control is a mechanism in which you can trigger the outputs based on certain conditions . As SAP puts it is is "The output or follow up processing of partner-dependent messages is automated via Message Control. The application then calls Message Control via the specified interfaces." There can be different forms of output like Workflow, print output, IDoc dispatch or even a fax. This is normally configured for the transactional data being entered and requirement being addressed.

SD and MM applications use message control for the message output. Message control is also referred as **Output control.**

Follow the link Message Control Explained for a detailed information on Message Control including how to configure it.

OK, lets end this month's blog by adding the details of the Message Control Tab and save our entry. Take a look at the screen below

These three key fields – Application, Message type and Process Code assigned in the message control tab together uniquely identify a message type which uniquely identifies an IDoc type.

Application: The Application specified in message control determines the output type and uniquely identifies a message type which can be assigned uniquely to an IDoc type. For Example: 'EA' is used for 'Purchasing RFQ' in Materials Management (MM) and V3' is used for 'Billing' in Sales and Distribution (SD).

Message type: Message type along with the application uniquely identifies a message type which can be assigned uniquely to an IDoc type.
For Example: 'LAVA' = 'Shipping notification' in dispatch (application 'V2').

Process code: The Process code is used by an IDoc Interface to determine the application Function Module which converts the SAP document into an IDoc.For Example: ME10: Purchase order (MM)

Here are the values we have chosen:

The application is V3: Billing
The Message type RD00: Invoice
The process code SD09: INVOIC: Invoice

Lets double click on the Process Code in this screen to see the configured Function Module we will be using.

Partner profiles: Outbound parameters

Partner No.	XI_EDI	XI EDI Demo Customer
Partn.Type	KU	Customer
Partner Role	BP	Bill-to party

Message Type: ZINVOIC — Custom Message Type for IDoc Blog
Message code:
Message function: ☐ Test

Outbound Options | **Message Control** | **Post Processing: Permitted Agent**

Application: V3 : Billing
Message Type: RD00 : Invoice
Process Code: SD09 : INVOIC: Invoice

Application	Message type	Process		Chang...	
V3	RD00	SD09	*Double Click!*	☐	

We would then be taken to Transaction WE41.

The function module which is embedded in each process code follows a naming convention "IDOC_<OUTPUT / INPUT >_NAME OF MESSAGE TYPE". For our example the delivered Message Type for an Invoice is INVOIC and this is an Outbound scenario, so the function module will be "IDOC_OUTPUT_INVOIC". (If this was a completely "Brand New" interface and not an Extension, we would need to create and configure our on function module.)

Summary

So in summary, we needed to create a New Message Type ZINVOIC. We then assigned our IDoc Extension to the new Message Type. We configured the Partner Profile to use our new IDoc Extension. We also touched on Message Control for Outbound IDoc scenarios.

The SAP IDoc Technology

In this month we will continue our look at SAP **IDocs** and the *IDoc* Technology by exploring how we can use the new custom IDoc extension we created in the last blog. If you need a refresher on how to extend an IDoc CLICK HERE

Finding and Updating an IDoc Customer Exit Using SMOD

If you recall from last month's blog, the Function Module we are using follows a naming convention "IDOC_<OUTPUT / INPUT >_NAME OF MESSAGE TYPE". For our example the delivered Message Type for an Invoice is INVOIC and this is an Outbound scenario, so the function module will be "IDOC_OUTPUT_INVOIC".

The **User Exit** for the transaction can be found using the transaction SMOD. Here we need to give the package to find the exact enhancement and the respective function module which will serve our purpose. The steps are as follows...
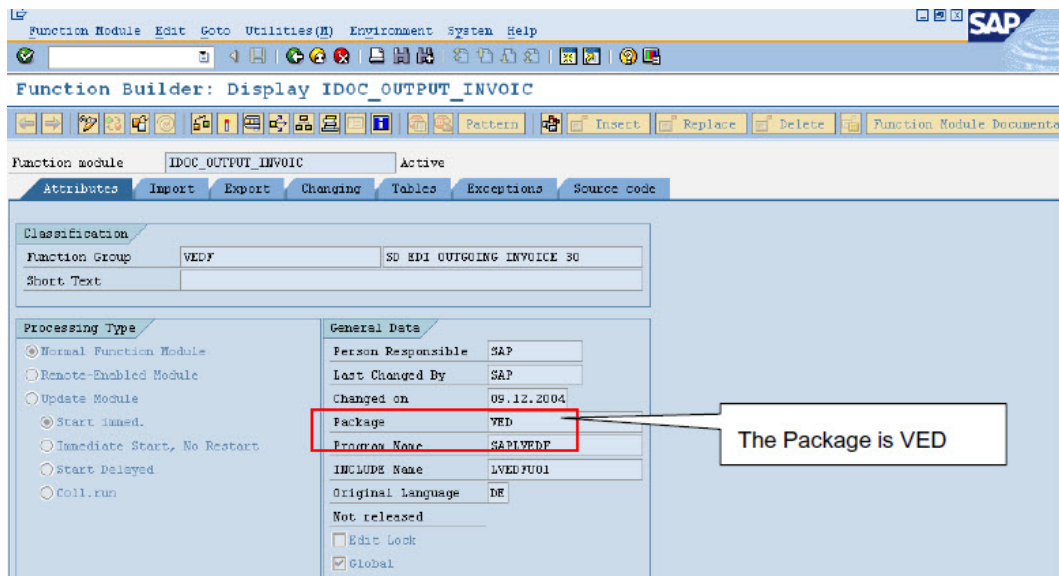
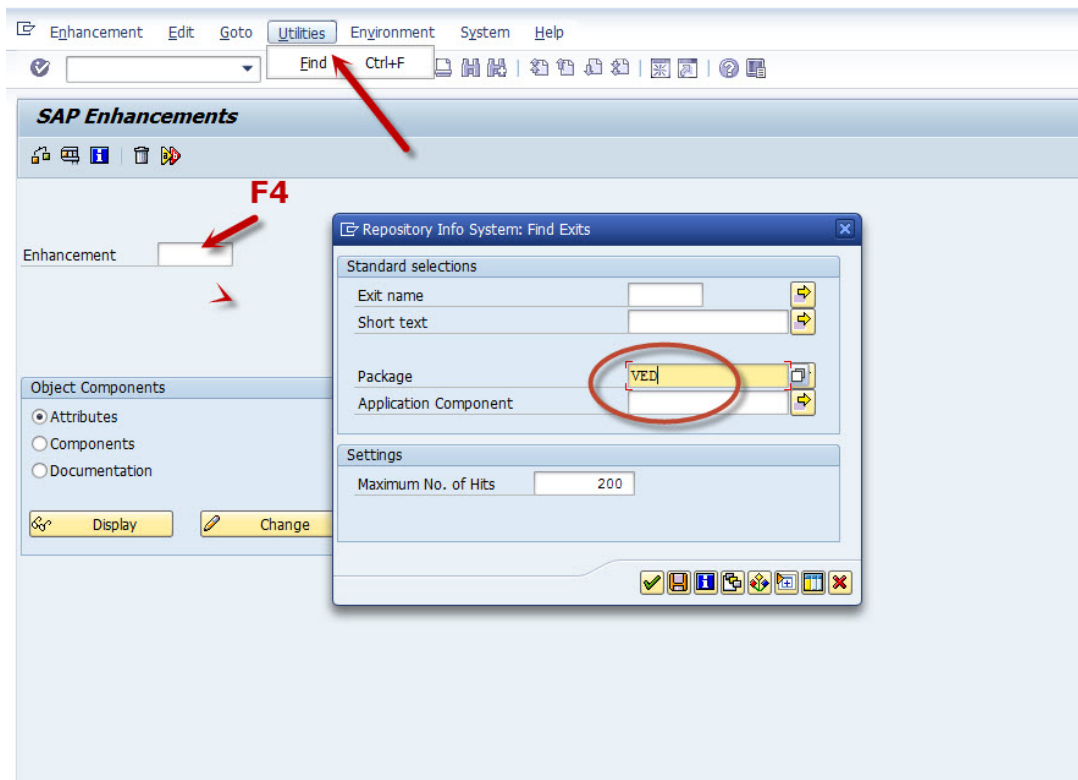Go to SE37 to find the package of "IDOC_OUTPUT_INVOIC" and click on Display.
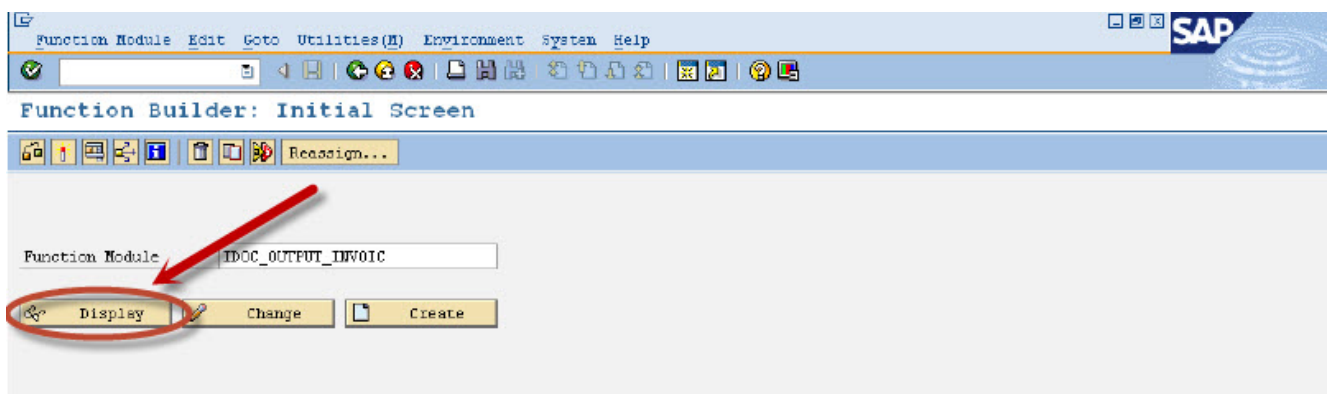
After clicking on display. Go to the "Attributes" tab and look for the package name.

Here we get the Package as "VED", Use this Package name in SMOD to find the respective User Exit function module.

Go to Transaction SMOD and give the value of Package we got (here "VED") in F4 help or utilities-> Find.

Hit the green check and we get the set of <u>User Exit</u> Names. One way to find the most suitable exit could be by the short text description. Sometimes you have to read the documentation for each exit, and sometimes it's plain old trial and error using the debugger and breakpoints. You can scroll through the Exits using the up and down arrows.
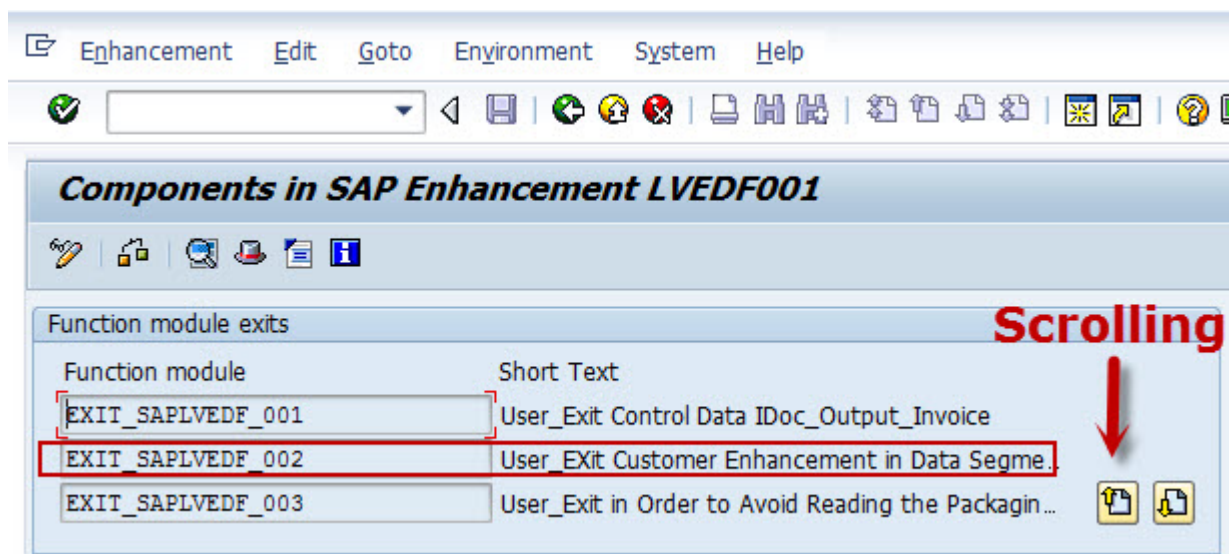

The short text for EXIT_SAPLVEDF_002 is "User_Exit Customer Enhancement in Data Segments for billing Docu". This sounds like us. Lets talk through the business requirement.

"We have to add a custom segment ZE1EDK01 as a child of standard segment E1EDK01 when creating the OUTBOUND IDoc for a billing document."

OK, this looks promising. I usually double click into the User Exit Function and examine the Import/Export and tables parameters so I know I can get at the data I need, and update the structure or tables I need.
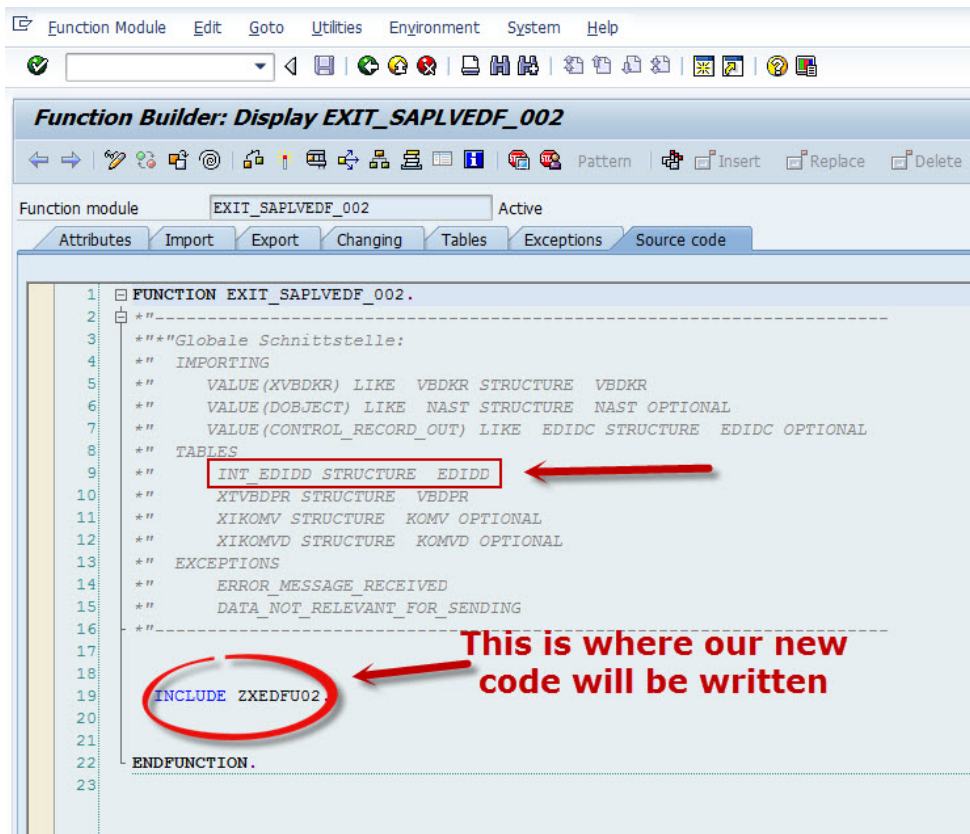

I can see in tables section of the "Formal" Parameters that I have access to INT_EDIDD and is typed to EDIDD. This is good as this is the *IDocs* segments. All I need to do is read this table looking for the segment E1EDK01, USE OPEN SQL to get the data I need and build and APPEND my custom Segment ZE1EDK01. This is a perfect place to do this.

Using Transaction CMOD to Build a Project and User Exit

First we need to create a project that will hold our enhancement. Execute transaction *CMOD* (**C**ustomer **MOD**) and fill in a project name and click create. I will use the name ZBILL as my project name.
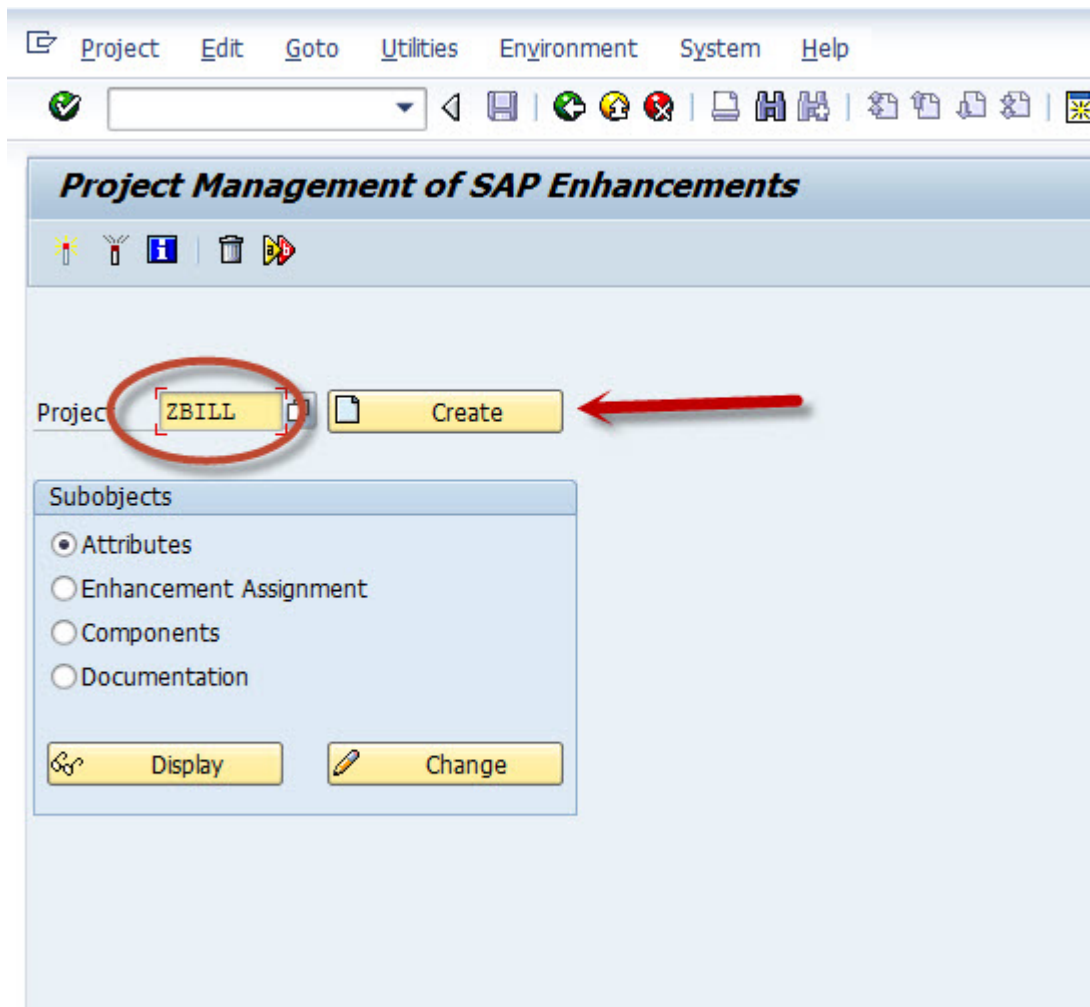

Fill in some descriptive text and click on Enhancement Assignments.

You can add the enhancement LVEDF001 that we found using SMOD and hit enter. You will most likely get the following error.

You can add the enhancement LVEDF001 that we found using SMOD and hit enter. You will most likely get the following error.
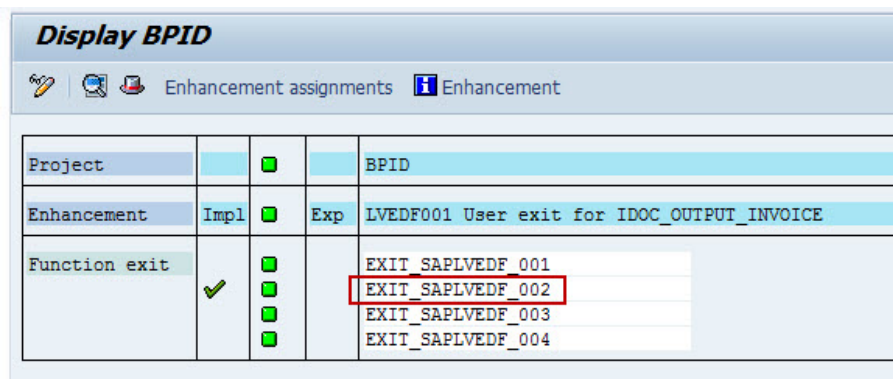
This means the enhancement is ALREADY in a project and can't be added to this one. The project is BPID. If we look at project BPID we will see it is already there.
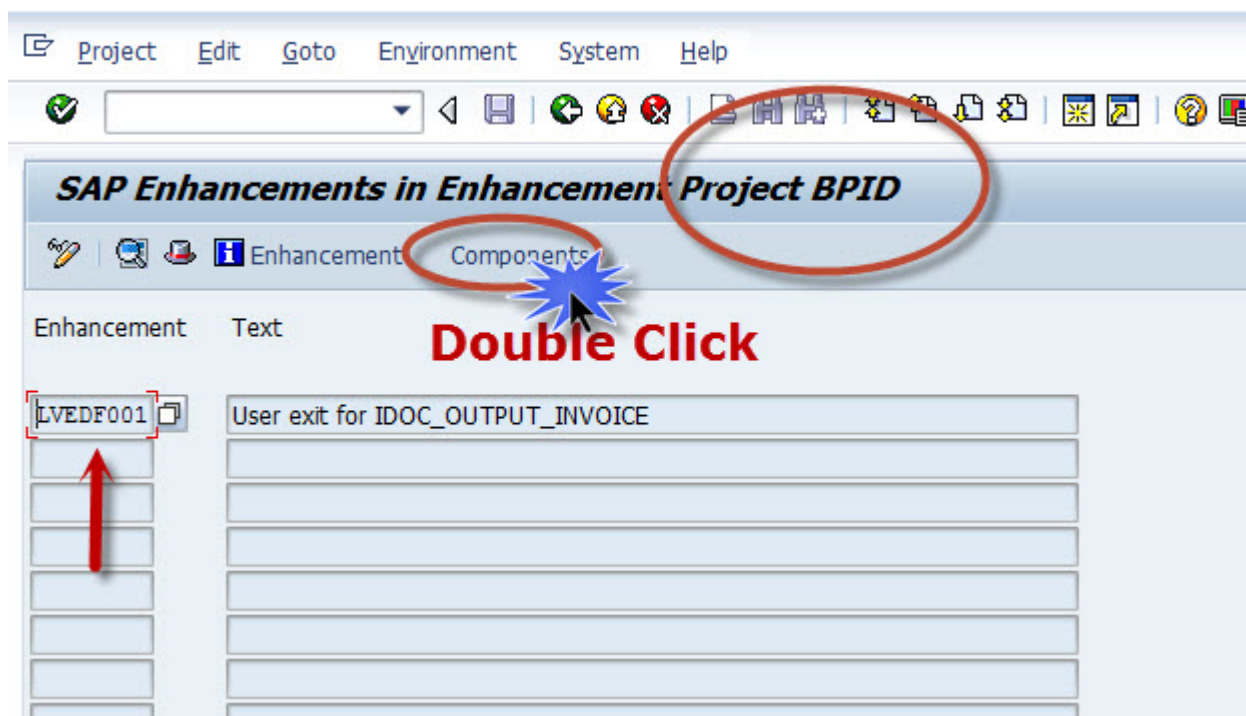
Adding your Custom Code to the CMOD Function Exit

Double Click on Components and you will see the exit EXIT_SAPLVEDF_002 we found using SMOD. Now all we have to do is add the code and make sure we activate the project and our code will be used when the *IDocs* are generated for outbound processing.

Double click into the exit and add the following code...

```
TYPES : BEGIN OF ty_vbrk,
          ktgrd TYPE ktgrd,
          mansp TYPE mansp,
        END OF ty_vbrk.

DATA : wa_edidd    TYPE edidd,
       v_lines     TYPE i,
       wa_e1edk01  TYPE e1edk01,
       wa_ze1edk01 TYPE ze1edk01,
       wa_vbrk     TYPE ty_vbrk.


READ TABLE int_edidd INTO wa_edidd WITH KEY segnam = 'ZE1EDK01
IF sy-subrc NE 0.
  READ TABLE int_edidd INTO wa_edidd WITH KEY segnam = 'E1EDK(
  IF sy-subrc = 0.
    wa_e1edk01 = wa_edidd-sdata.
    SELECT SINGLE ktgrd mansp FROM vbrk INTO wa_vbrk
        WHERE vbeln = wa_e1edk01-belnr.
    IF sy-subrc = 0.
      wa_ze1edk01-ktgrd = wa_vbrk-ktgrd.
      wa_ze1edk01-mansp = wa_vbrk-mansp.
      int_edidd-segnam = 'ZE1EDK01'.
      MOVE wa_ze1edk01 TO int_edidd-sdata.
      APPEND int_edidd.
      CLEAR int_edidd.
    ENDIF.
```

We have now populated our two new fields KTGRD and MANSP of our new custom segment ZE1EDK01.

Summary

So in summary, We used transaction SMOD to view the enhancements available for a package. We chose an enhancement suitable for our requirement and created or modified a PROJECT to include this enhancement. We then chose the correct component and added our custom code and activated the Project.

This was a very rudimentary example of how to enhance the supplied solution using Customer Exits (*CMOD*). I chose this method over BAdi's and other enhancement techniques as it lends it self nicely as an introduction for beginners. For a detailed and in-depth study of all the possible Enhancement techniques available in SAP, I would ask you to read our blogs on "The New Enhancement Framework".

I hope this has helped you begin your own journey into IDoc development in SAP. If you found value in this information, then click the button below to learn more.

Learn more