ALE Scenario Development Guide

10 February 1998

Written by: Kevin Wilson

TABLE OF CONTENTS

TABLE OF FIGURES

1. INTRODUCTION TO ALE DEVELOPMENT

To develop a new custom ALE scenario, comprises 5 steps:

1. Design and develop the custom IDoc with it' s segments and a new message type
2. Configure the ALE environment with the new IDoc and message type (customer model, partner profiles and linking IDoc to message type)
3. Develop the outbound process which does the following:
    - Populates the custom IDoc with control info and functional data
    - Sends the IDoc to the ALE layer for distribution
    - Updates status and handles errors
4. Configure the ALE inbound side (partner profiles with inbound process code)
5. Develop the inbound process which does the following:
    - Reads the IDoc into a BDC table; selects other data that is required
    - Runs transaction using call transaction or BDC session
    - Updates status and handles errors

Below is a pictorial representation of the flow of a complete ALE scenario from the sending system to the receiving system.

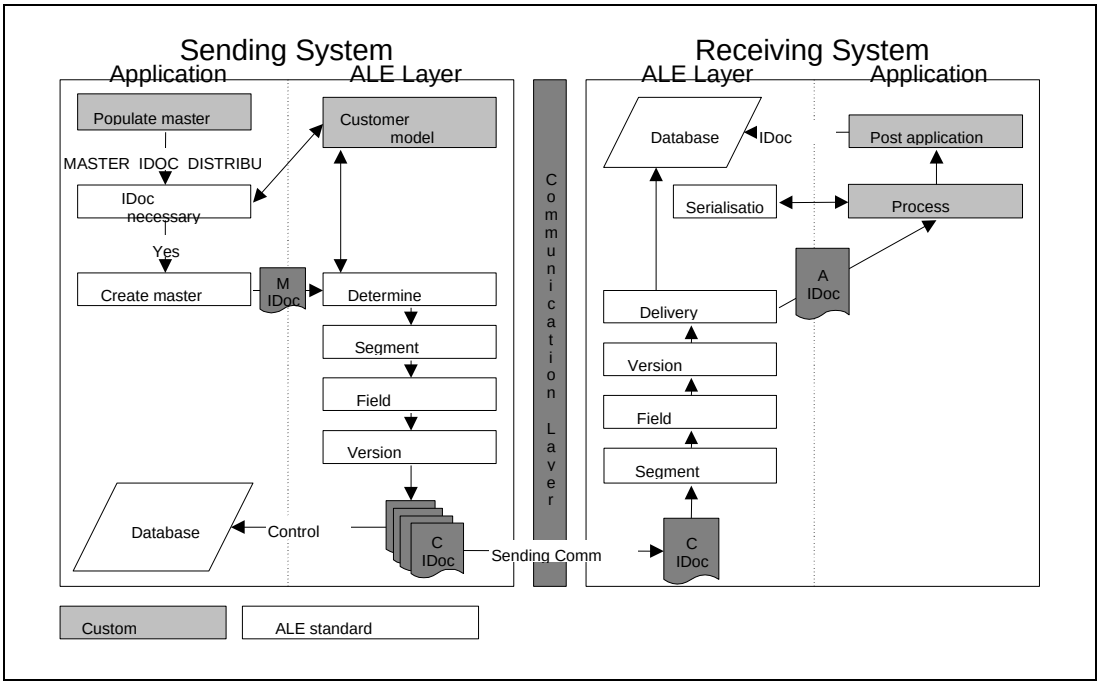

Figure 1: ALE Scenario model

1.1. ALE Example

For the purposes of this example we will develop a small ALE scenario. This scenario is described below.

" The receiver of an internal service must be able to reverse (cancel) the invoice receipt which will then cancel the applicable billing document automatically on the service provider' s system."

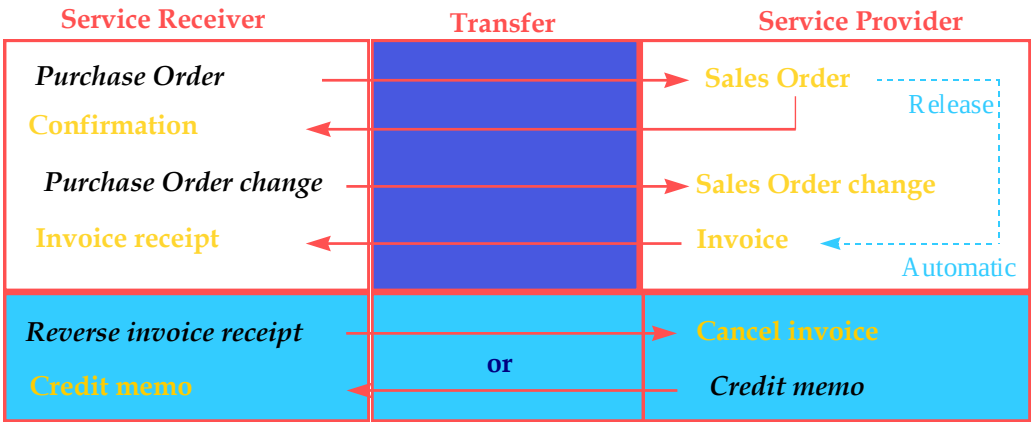

Figure 2: Example Purchasing & Selling scenario

We will develop a custom IDoc to carry the billing number from the Service Receiver' s system to the Service Provider' s system. We will populate the IDoc in a user exit on the sending side and we will process the transaction on the receiving side using a custom function module and a BDC transaction call.

No rule conversion, segment filtering or version conversion will be implemented in the model as described in Figure 1.

Requirements

- Working ALE environment - See ALE Basis Configuration Guide;
- ALE scenario design together with the business requirement;
- Development access; and
- ALE configuration access.

NOTES:

1. All IMG references to transactions are located in the transaction **SALE** which is the ALE portion of the IMG

2. This is one way of developing a scenario where no message control exists. If message control exist (EG. On purchase orders) then NAST can be used to call an outbound function module that would create the required IDocs.
3. Extensive knowledge of IDocs and ALE basis configuration is required in order to understand this guide.

2. OUTBOUND PROCESSING

2.1. Create IDoc type (WE30) Client independent

The IDoc type refers to the IDoc structure that you will require for your development. In our case the IDoc type is called *ZINVRV01*. This IDoc type will have 1 segment called *Z1INVRV* with 2 fields, *LIFNR & XBLNR*, in this segment. If you require many segments or nested segments then they are also created using the same procedure.

We will create the IDoc of the following structure:

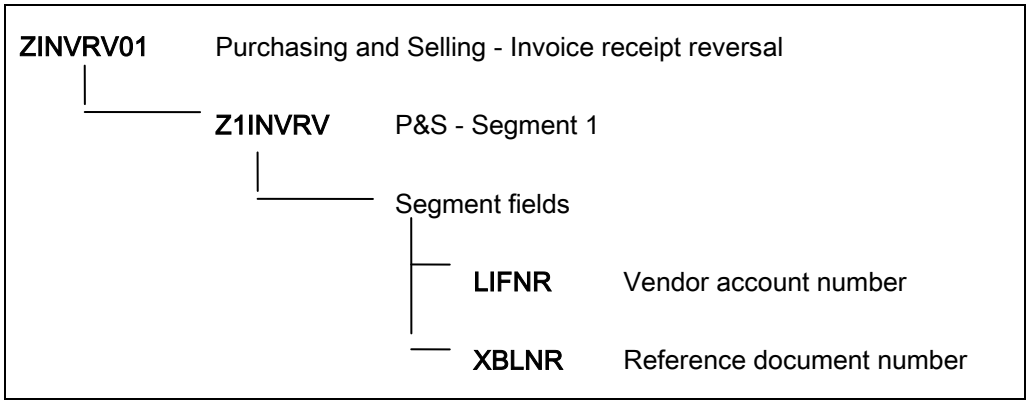

Figure 3: IDoc type ZINVRV01

To create the IDoc type, follow these next few steps:

- Enter transaction **WE30** (ALE -> Extensions -> IDoc types -> Maintain IDoc type)
- Type in **ZINVRV01** and click on **Basic IDoc type**, click the **Create** icon
- Click on **Create new** (we are creating an IDoc from scratch but you may want to copy another IDoc if it is similar to your requirements) and enter a **description**, and press **enter**
- Click on **ZINVRV01** and then on the **Create** icon
- Enter **Z1INVRV** as the segment type (must start with Z1), check mandatory if the segment must exist (in this case check it), enter **1** in minimum number and **1** as maximum number. (Make the maximum number 9999999999 if there are going to be many of these segments in each IDoc. IE. When line items are passed via IDocs), click on **Segment editor**
- Enter a **description** for your segment type and **create**

- Enter a **description** for your segment, enter each field required in your IDoc, in our case type **LIFNR** across for Field name, DE structure and DE documentation, repeat for **XBLNR** and press enter to validate.
- **Save** and **generate**, press **back**
- To release the segment choose **Goto**, **Release** from the menu
- **Check the box** on the line of your new segment
- **Save**, **back** and **enter**
- Your IDoc type structure should be displayed with your new segment
- **Save** and **back**
- To release the IDoc type choose **Extras**, **Release type** from the menu and **Yes**

Your IDoc is now ready for use. If you need to add fields or segments to your IDoc type, you will need to cancel the release of the IDoc type as well as the segment release using a similar procedure followed above (except now you uncheck the release box for the segment and you choose cancel release for the IDoc type).

2.2. Create message type (WE81) Client independent

To create a new message type, follow these next few steps:

- Enter transaction **WE81** (ALE -> Extensions -> IDoc types -> Maintain message type for intermed. Structure -> Create logical message type)
- Choose **Create logical message type** by double clicking on it
- Click on **change** icon to enter change mode
- Click on **New entries** to add a new type
- Enter the required message type, in our case it is **ZINVRV** and an appropriate **description**
- **Save** and **exit**.

Your message type has now been created. The next step will be to link it to the IDoc.

2.2.1. Link message to IDoc type (WE82 & BD69) Client independent

To link the message type to the IDoc type follow these next few steps:

- Enter transaction **WE82** (ALE -> Extensions -> IDoc types -> Maintain message type for intermed. Structure -> EDI: Message Types and Assignment to IDoc Types)
- Click on **change** icon to enter change mode
- Click on **New entries** to create the link
- Enter the message type **ZINVRV** and the BasicIDoc type as **ZINVRV01**
- **Save** and **exit**
- Enter transaction **BD69** (ALE -> Extensions -> IDoc types -> Maintain message type for intermed. Structure -> Assign message type to IDoc for ALE)
- Click on **change** icon to enter change mode
- Click on **New entries** to create the link
- Enter the message type **ZINVRV** and the BasicIDoc type as **ZINVRV01**
- **Save** and **exit**

Your IDoc is now linked to your message type. We still need to link object types and add the message to the model before we can use the message.

2.2.2. Maintain object type for message type (BD59) Client independent

The ALE objects are used to create links between IDocs and applications objects, to control the **serialisation**, to **filter messages** in the customer model and to use **listings**.

For our own message type and IDoc you must maintain ***object types for the links***.

If you want to check the serialisation for the message type, then you must maintain ***object types for the serialisation***. If no serialisation object has been maintained for a given message type, then the serialisation will not be checked for this message type.

To add an object type to our message type, follow these next few steps:

- Enter transaction **BD59** (ALE -> Extensions -> ALE object maintenance -> Maintain object types)
- Type in your message type **ZINVRV** and press **enter**
- Click on **New entries**

- Enter your object type, **LIFNR** (We need to use the vendor as a filter object), the segment name where LIFNR resides, **Z1INVRV**, a number **1** for the sequence followed by the actual field name **LIFNR**
- **Save** and **exit**.

You have now created an object that we' ll use as a filter object in the customer model to direct the flow of messages to the various logical systems based on the vendors in the filter of the message type ZINVRV.

We now need to add our new message type to the distribution model.

2.3. Configuring the Distribution Model

This task is performed on your ALE reference client.

2.3.1. Manual Configuration (BD64) Client dependent

To manually configure the customer distribution model, read the ALE configuration procedure, and follow these steps:

- Perform the **Maintain customer distribution model directly** function. (ALE -> Distribution customer model -> Maintain customer distribution model directly)
- Specify the customer model you want to maintain and the logical system that is to be the sender of the messages OR create a new model. (Create model ALE with logical system ALELS1C400)
- Choose the receiving systems to which the sending system must forward message type **ZINVRV** to.
- For each receiving logical system allocate the message type necessary for communication to the receiving systems as per ALE configuration procedure.
- Create filter objects (in our case **LIFNR** as the object type with the associated vendor number, **0000018001** with leading zeros, in the object area) for the message types.
- **Save** the entries.

NOTES:

You cannot maintain a message type between the same sender and receiver in more than one customer distribution model.

Only the owner is authorised to modify the model.

To change the owner of a model, choose the 'Maintain ownership of customer distribution model' function. Make sure that all changes will be distributed to all systems that know the corresponding model. To do so, you can use the correction and transport system.

To transport the customer distribution model you should use the Distribute customer model function of the IMG as described below.

2.3.2. Distribute customer model (BD71) Client dependent

After the customer model has been created centrally, it must be distributed to the other remote systems. This entails first of all setting up the communication for the distributed systems and then sending the model.

2.3.2.1. Distribute Model (BD71) Client dependent

This task is performed on your ALE reference client. To distribute the customer distribution model, read the ALE configuration procedure and follow these steps:

- Make the settings for the communication with the other decentral systems, you have not set them yet.
    - Define the RFC destination for R/3 connections whose names correspond to the name of the corresponding logical system.
    - Create the output partner profile.
- Distribute the customer model
    - Specify the name of the customer model.
    - You must specify the target system to which you want to distribute the customer model.
    - You must repeat this function for every distributed logical system.

2.3.2.2. Maintain sending system partner profile (WE20) Client dependent

With this function, you define the partner profiles for all outbound and inbound messages on the basis of the customer distribution model.

After you have defined and distributed the customer model, you will have to maintain the partner profiles locally. To do this read the ALE configuration procedure.

- Enter the output mode (background, immediately) and the package size for outbound processing.

Requirements

- The customer model must be maintained.
- RFC destinations must be maintained.
- The customer model must be distributed.
- To ensure that the appropriate persons in charge are informed if a processing error occurs, you must make settings in: Error processing Maintain organisational units.

2.4. Populate & distribute IDoc using ABAP

An IDoc consists of a control record with structure edidc and one or more data records with structure edidd. The control record contains the sender and recipient of the IDoc, as well as information on the type of message.

To be able to pass an IDoc to the ALE layer, you must set up a field string with structure edidc and an internal table with structure edidd. They are used to call function module **master_idoc_distribute**, which performs the save to the database and triggers the dispatch if necessary.

2.4.1. Example code

The code displayed below does the following:

- populates our IDoc segment Z1INVR with the 2 fields XBLNR and LIFNR, populates the segment name and appends this to an internal table used to store the IDoc data;
- populates the control record info with the message type and IDoc type; and
- calls the MASTER_IDOC_DISTRIBUTE function module which distributes the IDoc as configured in the customer distribution model.

```
*--- Data declaration statements
DATA:       C_INVREV_SEGNAME(7) TYPE C VALUE 'Z1INVRV',
            C_INVREV_MESTYPE(6) TYPE C VALUE 'ZINVRV',
            C_INVREV_IDOC_TYPE(8) TYPE C VALUE 'ZINVRV01',
            Z1INVRV LIKE Z1INVRV,
            C_INVREV_DOCTYPE LIKE BKPF-BLART VALUE 'YY',
```

```
                IDOC_CONTROL LIKE EDIDC,
                T_COMM_CONTROL LIKE EDIDC OCCURS 0 WITH HEADER LINE,
                IDOC_DATA LIKE EDIDD OCCURS 0 WITH HEADER LINE.


*--- Move the document header into a structure
LOOP AT DOC_HEAD_TAB INTO DOC_HEAD.
ENDLOOP.


*--- Move the document item data into a structure
LOOP AT DOC_ITEM_TAB INTO DOC_ITEM WHERE NOT ( LIFNR IS INITIAL ).
ENDLOOP.


*--- Populate the IDoc segment' s field with the required data
CLEAR Z1INVRV.
Z1INVRV-LIFNR = DOC_ITEM-LIFNR.          " Store vendor number for filter
Z1INVRV-XBLNR = DOC_HEAD-XBLNR.          " Billing number
IDOC_DATA-SEGNAM = C_INVREV_SEGNAME.     " Segment name
IDOC_DATA-SDATA = Z1INVRV.               " Segment data
APPEND IDOC_DATA.                        " Populate IDoc internal table


*--- Move the control data info required for the distribution
IDOC_CONTROL-MESTYP = C_INVREV_MESTYPE.
IDOC_CONTROL-DOCTYP = C_INVREV_IDOC_TYPE.


*--- Call the distribute function with the required parameters
CALL FUNCTION 'MASTER_IDOC_DISTRIBUTE' IN UPDATE TASK
                EXPORTING
                    MASTER_IDOC_CONTROL            = IDOC_CONTROL
                TABLES
                    COMMUNICATION_IDOC_CONTROL  = T_COMM_CONTROL
                    MASTER_IDOC_DATA               = IDOC_DATA
                EXCEPTIONS
```

```
ERROR_IN_IDOC_CONTROL                = 1
ERROR_WRITING_IDOC_STATUS            = 2
ERROR_IN_IDOC_DATA                   = 3
SENDING_LOGICAL_SYSTEM_UNKNOWN       = 4
OTHERS                               = 5.
```

Figure 4: Outbound processing example code

NOTE:

For debugging purposes, use transaction WE05 (IDoc overview) to see check your IDoc status, or to see whether an IDoc was created/

3. INBOUND PROCESSING

3.1. Create Function Module

This function module is called when a message type, of type ZINVRV, comes into the receiving system. This needs to be configured and is dealt with later in this section. The function module is passed the IDoc as a parameter.

Example parameters

Import parameters	Reference field	Opt Y/N
INPUT_METHOD	BDWFAP_PAR-INPUTMETHD	N
MASS_PROCESSING	BDWFAP_PAR-MASS_PROC	N

Export Parameters	Reference field	Opt Y/N
WORKFLOW_RESULT	BDWFAP_PAR-RESULT	N
APPLICATION_VARIABLE	BDWFAP_PAR-APPL_VAR	N
IN_UPDATE_TASK	BDWFAP_PAR-UPDATETASK	N
CALL_TRANSACTION_DONE	BDWFAP_PAR-CALLTRANS	N

Table Parameters	Reference field	Optional Y/N
IDOC_CONTRL	EDIDC	
IDOC_DATA	EDIDD	
IDOC_STATUS	BDIDOCSTAT	
RETURN_VARIABLES	BDWFRETVAR	
SERIALIZATION_INFO	BDI_SER	

Exceptions
WRONG_FUNCTION_CALLED

Example code

The code displayed below does the following:

- populates a BDC table with the IDoc info;
- calls the transaction via a BDC call; and
- updates the IDoc status according to the BDC error status.

```
EXTRACT FROM: Z_IDOC_INPUT_ZINVRV


*--- Declaration of local variables
DATA: C_SEGNAM(10) TYPE C VALUE 'Z1INVRV'.


*-Loop through the IDOCs
LOOP AT IDOC_CONTRL.
*---Loop through the data for the IDOC
                    LOOP AT IDOC_DATA WHERE DOCNUM = IDOC_CONTRL-DOCNUM.
                            CASE IDOC_DATA-SEGNAM.
                                WHEN C_SEGNAM.
*                                       Here we get the info from the idoc table
                                        IT_Z1INVRV = IDOC_DATA-SDATA.
                            ENDCASE.
                            PERFORM REV_INV.
                    ENDLOOP.
                    PERFORM UPDATE_IDOC_STATUS.
 ENDLOOP.


FORM REV_INV                "Reverse invoice form
*--- Local variables & constants
DATA: C_TCODE LIKE BKPF-TCODE VALUE 'VF11'.  "BDC transaction code


*--- Now we can build the bdc table to call the reversal transaction start of screen 109
CLEAR BDC_TAB.
BDC_TAB-PROGRAM  = 'SAPMV60A'.
BDC_TAB-DYNPRO   = '109'.
BDC_TAB-DYNBEGIN = 'X'.
```

```
APPEND BDC_TAB.

*--- Document number

CLEAR BDC_TAB.

BDC_TAB-FNAM = 'KOMFK-VBELN(01)'.

BDC_TAB-FVAL = IT_Z1INVRV-XBLNR.     "Billing document number

APPEND BDC_TAB.

*--- OK Code for screen 109

CLEAR BDC_TAB.

BDC_TAB-FNAM = 'BDC_OKCODE'.

BDC_TAB-FVAL = 'SICH'.

APPEND BDC_TAB.


*--- Now we can call transaction 'VF11' with the populated bdc table. The transaction is called inside the idoc-
contrl loop, so a transaction will be called for every idoc (journal). the transaction is called in no-display mode
('N') because this code runs in background as it is called by ale.  The update is specified to be synchronous
('S') because we have to wait for the result to update the idoc status correctly.

CALL TRANSACTION C_TCODE USING BDC_TAB MODE 'N' UPDATE 'S'.


*--- Store the return code for use in another form (status update)

RETURN_CODE = SY-SUBRC.


*--- Here we check the return code, if there was an error, we put the transaction in a bdc session for the user
to review and correct.

IF SY-SUBRC NE 0.
                    CALL FUNCTION 'BDC_OPEN_GROUP'
                         EXPORTING
                              CLIENT = SY-MANDT
                              GROUP  = 'ZINVRV'
                              USER   = C_ALE_USER
                              KEEP   = 'X'.
                    CALL FUNCTION 'BDC_INSERT'
                         EXPORTING
                              TCODE     = C_TCODE
                         TABLES
```

```
                                DYNPROTAB = BDC_TAB.
                        CALL FUNCTION 'BDC_CLOSE_GROUP'
                                EXCEPTIONS
                                    NOT_OPEN    = 1
                                    QUEUE_ERROR = 2
                                    OTHERS      = 3.
ELSE.                                                   "No problems
                        C_EXISTS = 'N'.
*   Select from the billing document table to get sales doc number
                        SELECT * FROM VBRP WHERE VBELN = IT_Z1INVRV-XBLNR.
*     Select from the sales document table to get user status number
                                SELECT SINGLE * FROM VBAP WHERE VBELN = VBRP-AUBEL AND
                                    POSNR = VBRP-AUPOS.
*     Select from the status table to change the user status to pending
                                SELECT * FROM JEST WHERE OBJNR = VBAP-OBJNR AND
                                    STAT LIKE C_USER_STATUS.
                                    IF JEST-STAT = C_US_PENDING.   "User status is pending
                                            JEST-INACT = C_UNCHECKED.    "Make pending the active status
                                            UPDATE JEST.
                                            C_EXISTS = 'Y'.          "I.E. An entry is already in table
                                    ELSEIF    JEST-INACT    =    C_UNCHECKED    AND    JEST-STAT    NE
C_US_PENDING.
                                            JEST-INACT = C_CHECKED.      "Make everything else inactive
                                            UPDATE JEST.
                                    ENDIF.
                                ENDSELECT.
                                IF C_EXISTS = 'N'.    "I.E. Pending has never been a status before
                                    JEST-OBJNR = VBAP-OBJNR.
                                    JEST-STAT  = C_US_PENDING.
                                    JEST-INACT = C_UNCHECKED.      "Make pending the active status
                                    INSERT JEST.
                                ENDIF.
                        ENDSELECT.      "Select from VBRP (Billing document table)
```

```
ENDIF.

ENDFORM.                    " REV_INV


FORM UPDATE_IDOC_STATUS.

*--- Now we check the CALL TRANSACTION return code and set IDOC status
                    CLEAR IDOC_STATUS.
                    IF RETURN_CODE = 0.
                            WORKFLOW_RESULT = '0'.
                            IDOC_STATUS-DOCNUM = IDOC_CONTRL-DOCNUM.
                            IDOC_STATUS-STATUS = '53'.
                            IDOC_STATUS-UNAME = SY-UNAME.
                            IDOC_STATUS-REPID = SY-REPID.
                            IDOC_STATUS-MSGTY = SY-MSGTY.
                            IDOC_STATUS-MSGID = SY-MSGID.
                            IDOC_STATUS-MSGNO = SY-MSGNO.
                            IDOC_STATUS-MSGV1 = SY-MSGV1.
                            IDOC_STATUS-MSGV2 = SY-MSGV2.
                            IDOC_STATUS-MSGV3 = SY-MSGV3.
                            IDOC_STATUS-MSGV4 = SY-MSGV4.
                            RETURN_VARIABLES-WF_PARAM = 'Processed_IDOCs'.
                            RETURN_VARIABLES-DOC_NUMBER = IDOC_CONTRL-DOCNUM.
                            APPEND RETURN_VARIABLES.
                    ELSE.
                            WORKFLOW_RESULT = '99999'.
                            IDOC_STATUS-DOCNUM = IDOC_CONTRL-DOCNUM.
                            IDOC_STATUS-STATUS = '51'.
                            IDOC_STATUS-UNAME = SY-UNAME.
                            IDOC_STATUS-REPID = SY-REPID.
                            IDOC_STATUS-MSGTY = SY-MSGTY.
                            IDOC_STATUS-MSGID = SY-MSGID.
                            IDOC_STATUS-MSGNO = SY-MSGNO.
                            IDOC_STATUS-MSGV1 = SY-MSGV1.
```

```
                    IDOC_STATUS-MSGV2 = SY-MSGV2.
                    IDOC_STATUS-MSGV3 = SY-MSGV3.
                    IDOC_STATUS-MSGV4 = SY-MSGV4.
                    RETURN_VARIABLES-WF_PARAM = 'ERROR_IDOCS'.
                    RETURN_VARIABLES-DOC_NUMBER = IDOC_CONTRL-DOCNUM.
                    APPEND RETURN_VARIABLES.
                ENDIF.
                APPEND IDOC_STATUS.
ENDFORM.                    " UPDATE_IDOC_STATUS
```

Figure 5: Inbound processing example code

3.1.1. Debugging inbound FM

Use transaction **WE19** to test inbound function module in debugging mode. Also use **WE05** to view the IDocs and their statuses.

3.2. Maintain ALE attributes

The inbound function module needs to be linked to the message type and the message type needs to be linked to the appropriate inbound process code at the partner profile level before the scenario is enabled. These steps are described below in detail.

3.2.1. Link Message Type to Function Module (WE57) Client independent

To link a message (ZINVRV) type to a function module (Z_IDOC_INPUT_ZINVRV) follow these steps:

- Enter transaction **WE57** (ALE -> Extensions -> Inbound -> Allocate function module to logical message)
- Select an entry (EG. IDOC_INPUT_ORDERS) and **copy**
- Type in module name **Z_IDOC_INPUT_ZINVRV**
- Type in basic IDoc type as **ZINVRV01**
- Type in message type as **ZINVRV**
- Type object type as **IDOCINVOIC** (Invoice document) - Used for workflow
- Direction should be set to **2** for inbound
- **Enter** and **save**

3.2.2. Define FM settings (BD51) Client independent

- Enter transaction **BD51** (ALE -> Extensions -> Inbound -> Define settings for input modules)
- Click on **New entries**
- Type in the name of the new function module **Z_IDOC_INPUT_ZINVRV**
- Enter **0** for mass processing in the output column
- **Save** and **Exit**

3.2.3. Maintain process codes (WE42) Client dependent

A process code needs to be maintained on each client. It then needs to be linked to the message via the partner profiles on each client. This allows the various clients to use a unique function module for the same message type.

To maintain the process code follow these steps:

- Log on to the appropriate receiving system client
- Execute **WE42** (ALE -> Extensions -> Inbound -> Maintaining process codes inbound)
- Choose **Inbound with ALE service**
- Choose **Processing with function module**
- Click on **Processing with function module** and choose **create** icon
- Click on **New Entries**
- Type in process code **ZINR** and give it a **description** and **save**
- Now you are asked to Please maintain codes added in ALE entry methods, **enter** and choose **Z_IDOC_INPUT_FIRVSL** and **copy** it. You should choose a FM similar to your one.
- Enter your process code **ZINR**
- Enter your function module **Z_IDOC_INPUT_ZINVRV**

NOTE: The next 6 steps are used in workflow error handling.

- Enter **IDPKFIDCMT** in object type
- Enter **MASSINPUTFINISHED** in End event
- Enter **IDOCINVOIC** in IDoc object type

- Enter **INPUTERROROCCURREDFI** in IDoc start event
- Enter **INPUTFINISHEDFI** in IDoc End event
- Enter **IDOCINVOIC** in Application object type

You will need to determine the task associated with object IDOCINVOIC, and then assign the appropriate position to it. This position will then receive the application error messages via workflow.

To set up the workflow area please consult the Workflow config guide.

3.3. Create inbound partner profile

For each message type you need to maintain the inbound partner profiles.

3.3.1. Maintain receiving system partner profile (WE20) Client dependent

To maintain inbound partner profiles read the document ALE configuration procedure:

- Add the message type **ZINVRV** with process code **ZINR**.
- Enter the output mode (background, immediately) for inbound processing and **NO** message code.
- Enter the position **S** and choose the ALE administrator **50000085**. This position will then receive all the technical ALE errors via workflow.

3.4. Test

Once the inbound function module has been debugged the scenario should be ready to test in its entirety. If problems occur, read through the relevant areas of this document to check your configuration or code.