Ensure consistency and synchronization across your master data: Integrate your classification changes into customer and vendor master data ALE scenarios

by Anthony Cecchini

Anthony Cecchini
Independent Consultant

Anthony Cecchini is the president of Information Technology Partners (ITP), an SAP consulting company headquartered in Bridgeville, Pennsylvania. ITP offers comprehensive planning, resource allocation, implementation, upgrade, and training assistance to companies. Anthony has over 15 years of experience in SAP R/3 business process analysis and SAP systems integration. His areas of expertise include SAP NetWeaver integration; ALE development; RFC, BAPI, IDoc, Dialog, and Web Dynpro development; and customized Workflow development. Currently Anthony is working on a mySAP ERP implementation (ECC 5.0). You can reach him at ajcecchini@itp-consulting.com.

I have been an SAP consultant for 15 years, and I consistently find a similar issue at almost all customer sites I visit. Let me elaborate: The RICE (reports, interfaces, conversions, and extensions) team creates the standard ALE customer or vendor master data scenario to distribute master data changes to either another SAP system or a third-party system. Because SAP does not provide classification data associated with customer or vendor master data in these scenarios, the team also needs to create a new IDoc (Intermediate Document) type from the standard DEBMAS (i.e., customer master data) or CREMAS (i.e., vendor master data) type and then extend that IDoc type with a new segment to hold the classification data. The team codes the user exit EXIT_SAPLVV01_001[1] to populate the additional segment when a new IDoc is created by the system, and configures change pointers and the distribution model to manage the changes and define the data flow, respectively.

The result? When a user changes customer data or vendor data, voilà! An IDoc is produced, the target system gets updated, and everyone is happy. However, a problem occurs when a user changes only classification data. No IDoc is produced at all! A typical workaround to this problem is to instruct users to always edit some other, unimportant customer or vendor data at the same time. Although this approach works, it certainly isn't the most effective or efficient way to address the problem. What happens if a user forgets to perform the workaround? So, how do we get the same IDoc that is triggered when customer or vendor master data is changed to trigger when only classification data is changed?

This article provides solutions for both of the issues described here — i.e., including classification data in master data scenarios and triggering an IDoc when only classification data is changed. The first half of the article is for developers who have not yet created an extended IDoc to include

[1] This user exit can be found using the SMOD transaction. It is part of enhancement VSV00001.

classification data. Here you will find the steps necessary to create that extended IDoc. The second half of the article addresses the second issue. Here we will change an existing shared master data (SMD) function module by copying the SAP original SMD and adding the necessary coding to trigger an IDoc when only classification data is changed.

This article is for developers who are already familiar with ALE, ALE configuration, and the concept of IDocs. I also assume that you understand the problems as stated and are looking for a technical how-to approach to resolve them. And although developers will perform the steps outlined here, administrators and managers can also benefit from this article — an understanding of what is at stake and how these problems can be remedied will help you manage and maintain master data and classification data on a regular and ongoing basis.

For the purposes of this article, I use a customer master data scenario. We will begin "at the beginning" and travel from configuration through code development to deployment. Each step in the process builds upon its predecessor and finally converges into a solution. First let's walk through the steps needed to add classification data to an IDoc. These steps must be completed in order to perform the second set of steps needed to trigger that same IDoc for classification-only changes. If you already have modified an IDoc to include classification data, you can skip the first section, although reading through the process in preparation for performing the second set of steps might be helpful. And what you learn in this article about updating customer master data applies to updating vendor master data as well — only the objects will change.

Note!

This article applies to system landscapes prior to SAP NetWeaver '04. SAP NetWeaver '04 includes SAP NetWeaver Master Data Management (MDM) 5.5, which eliminates the two issues described in this article. SAP NetWeaver MDM provides scenarios and tools that help to consolidate, harmonize, and distribute accurate, up-to-date customer, vendor, and classification data without customization. For more information on SAP NetWeaver MDM 5.5, see the article "Centralize, harmonize, and distribute your master data with SAP NetWeaver Master Data Management (MDM)," which appears on page 97 in this issue of *SAP Professional Journal*.

Adding classification data to an IDoc

Let's begin our first task by reviewing a diagram of our example customer data ALE scenario (see **Figure 1**). We'll use a generic master data flow that can represent customers or vendors. The important point is that we are using the change pointer technique to pass master data changes to the target system.

As shown in the figure, when a change to the master data is made, the application in which the SAP transaction occurred creates a change document, which tracks the change. The application analyzes the change document and creates change pointers in the database tables. The change pointers store the changes in the BDCP and BDCPS database tables. Then a user or a batch job runs the RBDMIDOC program to generate an IDoc to transfer the change to the target system via the ALE layer. At this point, the IDoc is run against the distribution model[2] so the change pointers can implement any conversion or filtering specified in the distribution model before passing the changes to the target system via the asynchronous remote function call ARFCDATA.

In our example, all customers are included (vs. individual records). Our intent is to pass the changes for all customer data to the destination, or target system (in this example, a Siebel system), and in particular to make sure that any associated classification data (which, in this case, has not been changed) is also passed to the

[2] The distribution model controls the data flow between the sending and receiving target systems.

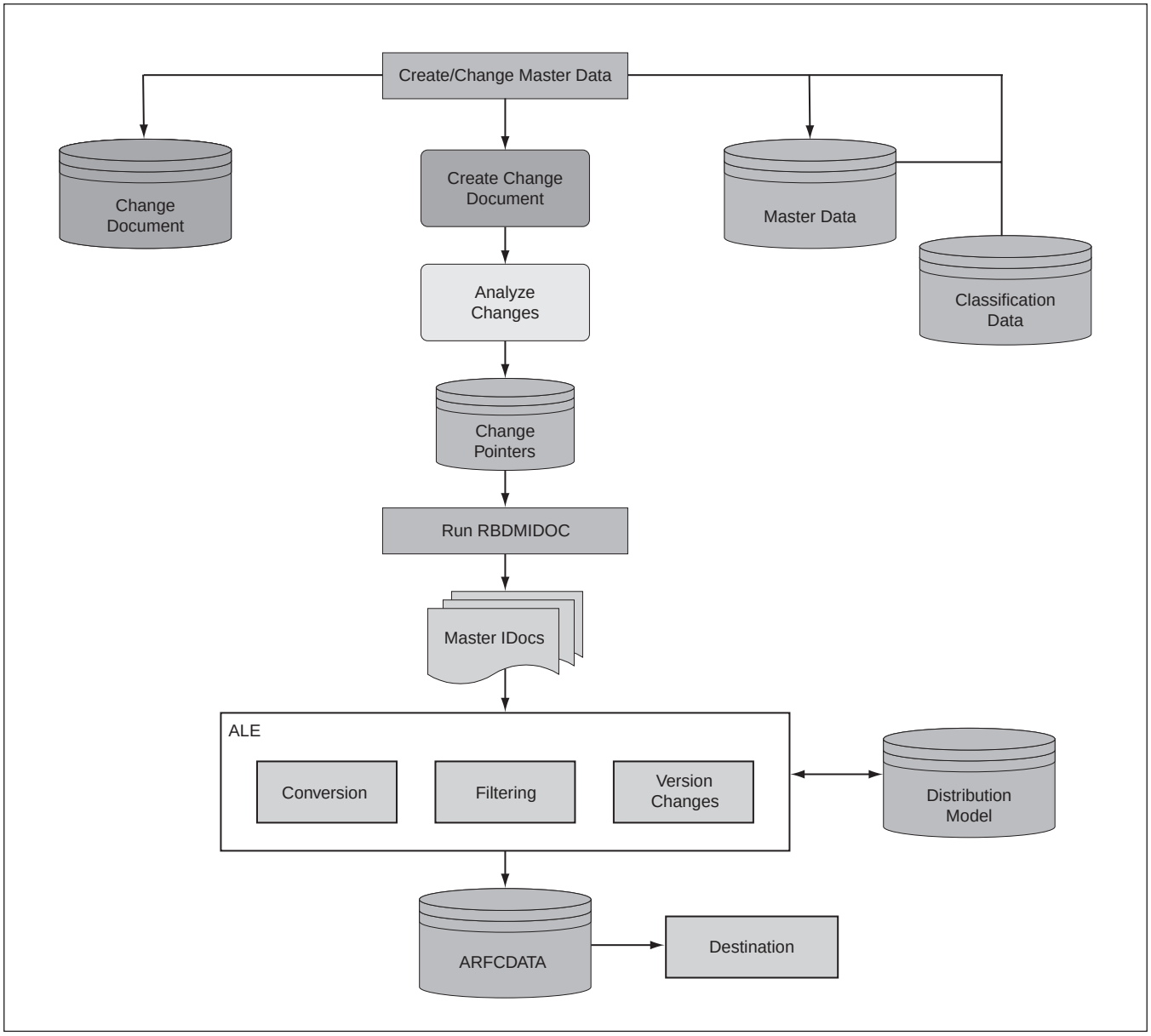

Figure 1 Generic master data flow for the example customer scenario

destination, or target system. Passing both the customer master data with changes and the classification data without changes to the target system ensures that all systems (databases) are in synch.

To create the extended IDoc type to trigger the IDoc, follow these steps:

1. Create a new segment that holds all the classification fields.
2. Create an IDoc type extension for an existing IDoc.
3. Add the new segment to the IDoc type extension.
4. Create a message type.
5. Link the message type to the IDoc type extension.
6. Turn on change pointers globally and specifically for the message types.

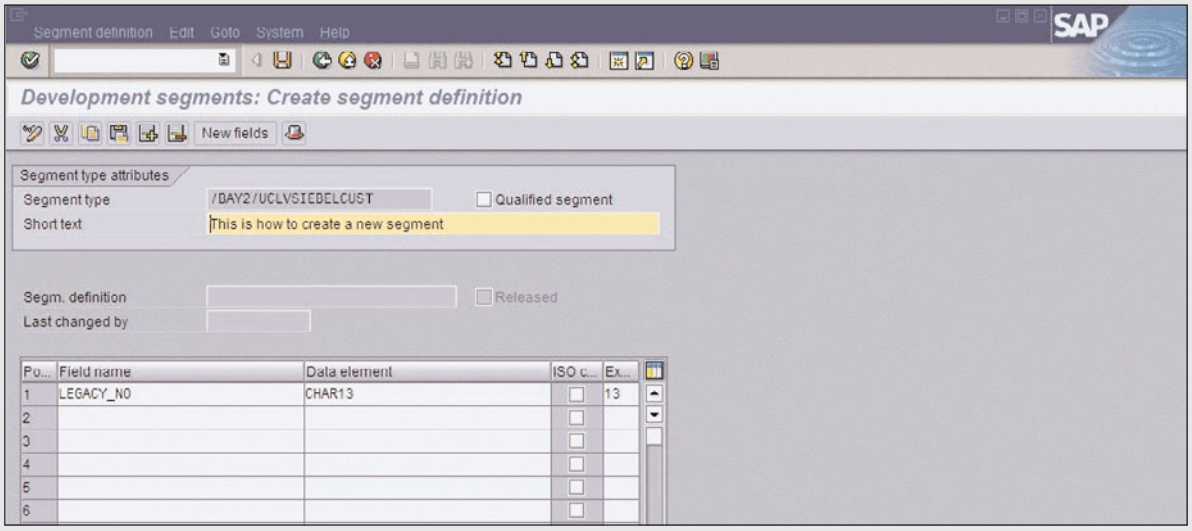

Figure 2 Creating a new segment

7. Identify the fields and tables for which change pointers need to be written.
8. Set up the distribution model.
9. Set up the partner profile.
10. Configure the user exit to capture the classification data and to append the data to the segment.

Step 1: Create a new segment that holds all the classification fields

The segment that we need to create will hold all the classification fields that we want to pass to the target system (Siebel, in this case). To create a new segment, use transaction WE31 (IDoc Segment). In the Create segment definition screen, enter the name of the segment type (/BAY2/UCLVSIEBELCUST in the example). The segment type is the name of the segment and is independent of the SAP release; in other words, the segment will be compatible with future releases. Next, enter a short description of the segment (e.g., This is how to create a new segment), and then enter the names of the fields (in the example, field LEGACY_NO with data element CHAR13) that will be passed along through the IDoc (see **Figure 2**). Save the segment.

> ***Note!***
>
> For the examples in this article, I use /BAY2/ instead of Z or Y as the prefix for the custom objects (e.g., message types, function modules, etc.).

The new segment needs to be linked to the existing IDoc that we want to use. But to do that, we first need to create an IDoc type extension.

Step 2: Create an IDoc type extension for an existing IDoc

The existing IDoc that we are going to use already has IDoc types associated with it, but none that do what we need them to do, which is pass the classification data. We will create a new IDoc type by adding our new segment to the existing "basic type" IDoc and giving it a new name (customer namespace).

To create a new IDoc type extension, use transaction WE30 (IDoc Type). Enter a name for the new

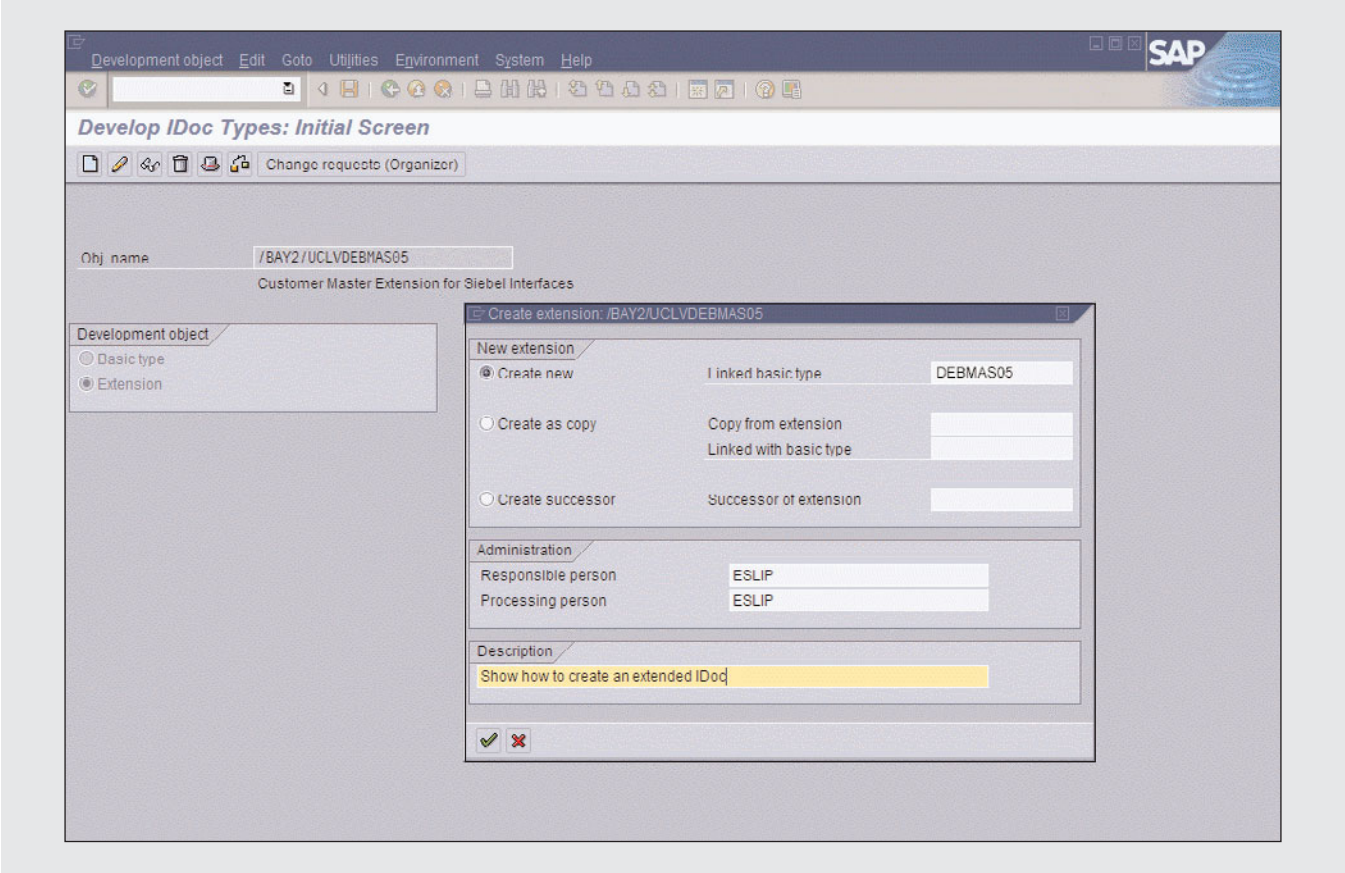

Figure 3 Creating an extended IDoc type for the basic DEBMAS05 type

extended IDoc (/BAY2/UCLVDEBMAS05 in the example), select the Extension option as the development object, and then click on the Create button. Because this is an extension of an existing IDoc, you need to link the new IDoc type to the basic type, which in this case is DEBMAS05, as shown in **Figure 3**. Enter a description (e.g., Show how to create an extended IDoc), and then click on the Enter button.

Next we will add the new segment to the new IDoc extension.

Step 3: Add the new segment to the IDoc type extension

When you add a segment that you have created (or modified) to the extended IDoc type, you must add it at the correct level of the segment hierarchy in the IDoc type. Because classification data is header-level data, we will add our new segment after E1KNA1M. Place your cursor on the E1KNA1M segment and click on the Create button. The Maintain Attributes dialog opens, as shown in **Figure 4** on the next page.

Here you enter the name of the new segment (/BAY2/UCLVSIEBELCUST in the example). You also have to indicate whether the segment is mandatory by checking the Mandatory seg. checkbox (which you want to do because this segment should always be implemented). Then click on the Enter button to insert the new segment after E1KNA1M, with the fields that you specified in Step 1 (in the example, the LEAGACY_NO field). Save the segment. We now have a new, extended IDoc type with a new custom segment that will be populated with classification data via a user exit when the IDoc is created.

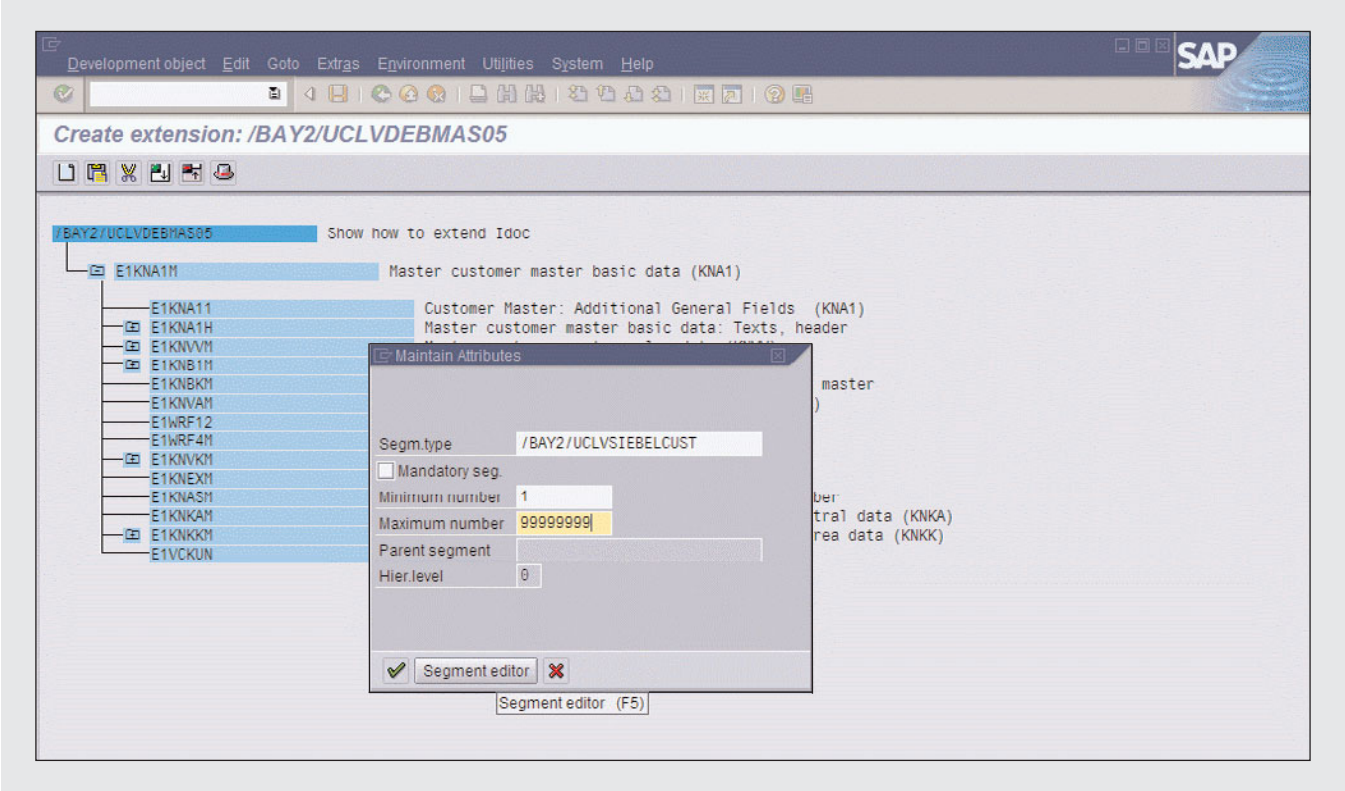

Figure 4 Adding the new segment to the extended IDoc at the header level

Step 4: Create a message type

Message types ensure that the changes made to customer, vendor, or classification data are recognized by the target system. The message types for changes to customer and vendor master data already exist as part of the SAP-supplied IDocs DEBMAS and CREMAS, respectively, so we need to create a message type that will identify changes made to classification data.

> ***Tip!***
>
> You can go to http://ifr.sap.com/index.html (the SAP Interface Repository) to find message types and IDocs for specific objects, such as customer or vendor objects.

To create a message type, use transaction WE81 (Logical Message Types). Enter the message type (/BAY2/UCLVSBCUSTRED in the example) and a description (e.g., AH Outbound Customers), as shown in **Figure 5**, and then save the message type.

The next step is to link the message type to the IDoc type extension.

Step 5: Link the message type to the IDoc type extension

Now we need to link the message type and IDoc so that when the change request is received by the target system, the sending system will know which IDoc to create. To create the link, use transaction WE82 (Assignment Messages for IDoc Type). Enter the basic type (in the example, DEBMAS05), its extension (/BAY2/UCLVDEBMAS05), and the message type

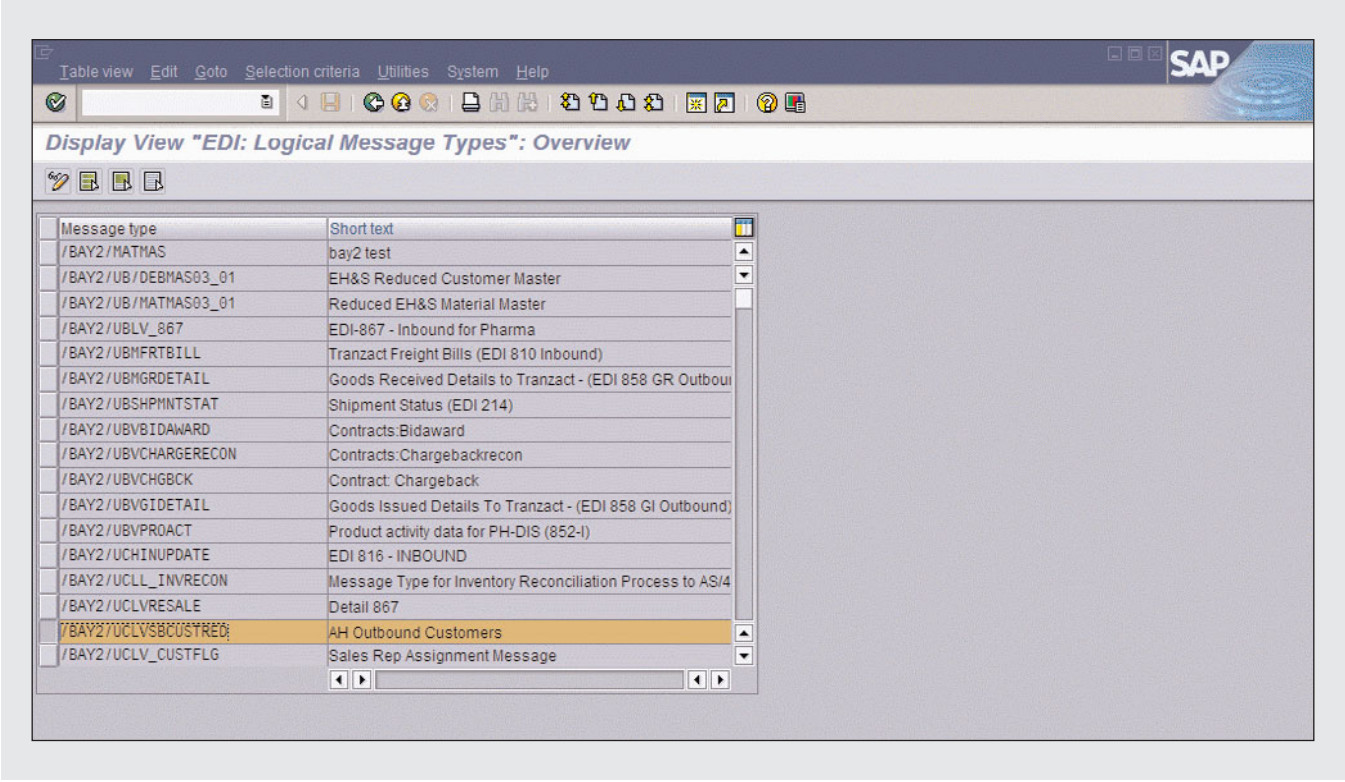

Figure 5 Creating a message type for the extended IDoc

(/BAY2/UCLVSBCUSTRED). You must also specify the current release, which is SAP R/3 4.6C in the example.

This step enables the partner profile (which we'll discuss in Step 9) to check the triggered IDoc for the message type that describes the data being passed.

Next, we need to turn on the change pointers. First, we need to turn on the change pointers globally so that when the sending system has been notified of a change, it can accept the data. We also need to turn on the change pointers for the specific message type so that the changes to the fields and data elements specified in the message type can be passed when a change occurs.

Step 6: Turn on change pointers globally and specifically for the message types

We first need to turn on change pointers globally so all changes are tracked. The changes will be recorded in table BDCP. Use transaction BD61 (Activate Change Pointers). Specify the activation status by selecting the Change pointers activated - generally checkbox, as shown in **Figure 6** on the next page.

Next we need to turn on the change pointers for our message type. Use transaction BD50 (Activate Change Pointers for Message Type). In the display screen, make sure there's a checkmark in the Activity column next to the message type (in this example, /BAY2/UCLVSBCUSTRED), as shown in **Figure 7** on the next page.

Step 7: Identify the fields and tables for which change pointers need to be written

Next we need to tell the SAP system which fields and tables (i.e., newly created or modified) should have a

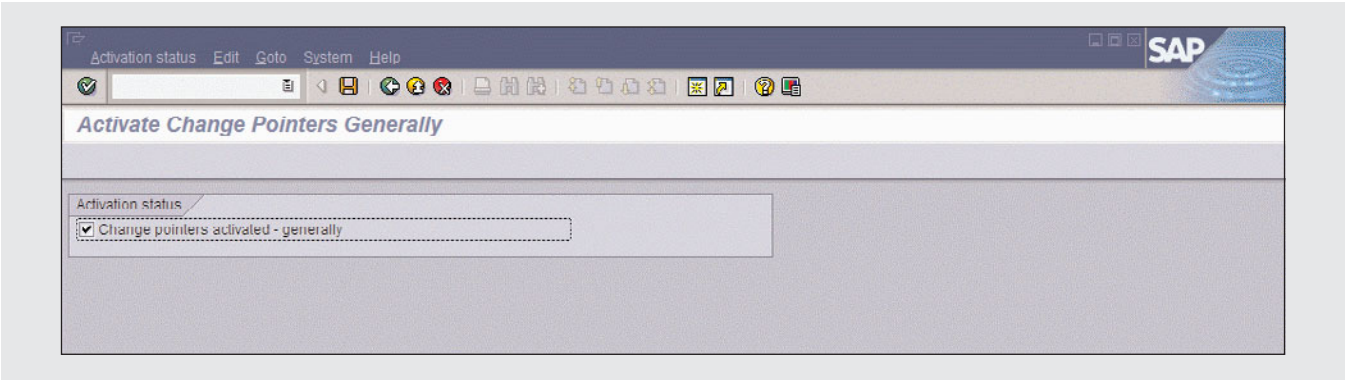

Figure 6 Turning on change pointers globally

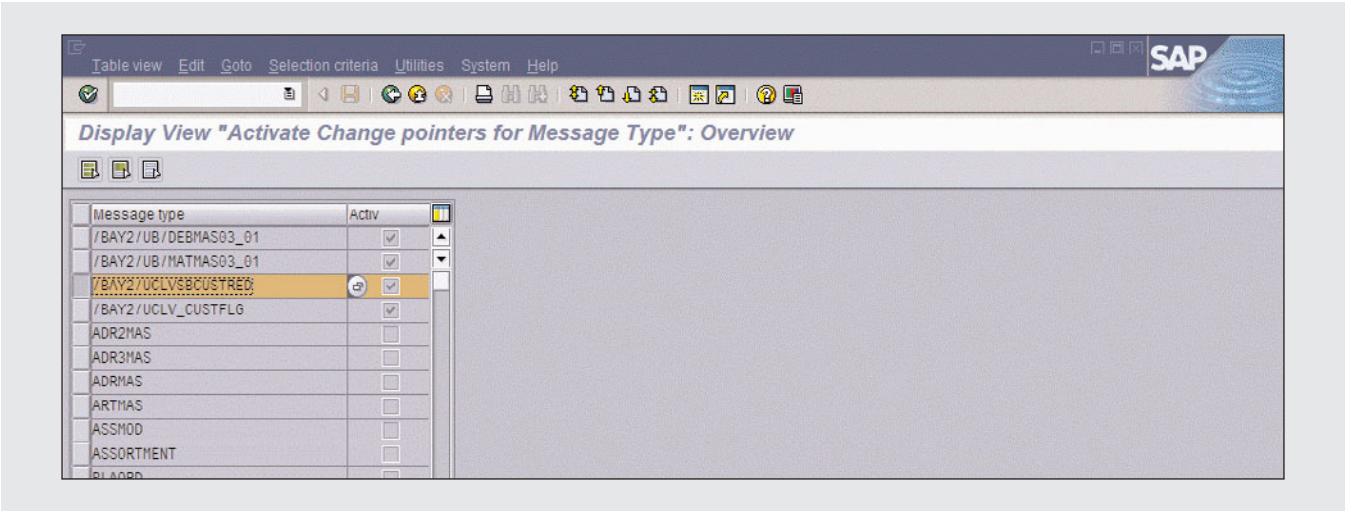

Figure 7 Turning on change pointers for the message type

change pointer written for them. Use transaction BD52 (Activate Change Pointer Per Change Document Item), and enter the message type for each field that should trigger a change pointer, or (as I usually do) cut and paste the information from the standard message type. **Figure 8** shows the fields in the /BAY2/UCLVSBCUSTRED message type.

Step 8: Set up the distribution model

The distribution model provides different views of the logical flow between systems. In this case, we need to specify the Siebel system as the target system. To set up the distribution model, use transaction BD64 (Maintenance of Distribution Model). Setting up the distribution model is part of the ALE layer, which uses the distribution model to control the flow of data. **Figure 9** shows the configured distribution model.

Next we're going to set up the partner profile, which identifies who can exchange messages with the back-end system and which port will be used for the exchange (setting up a port is beyond the scope of this article).

Step 9: Set up the partner profile

The partner profile enables the communication

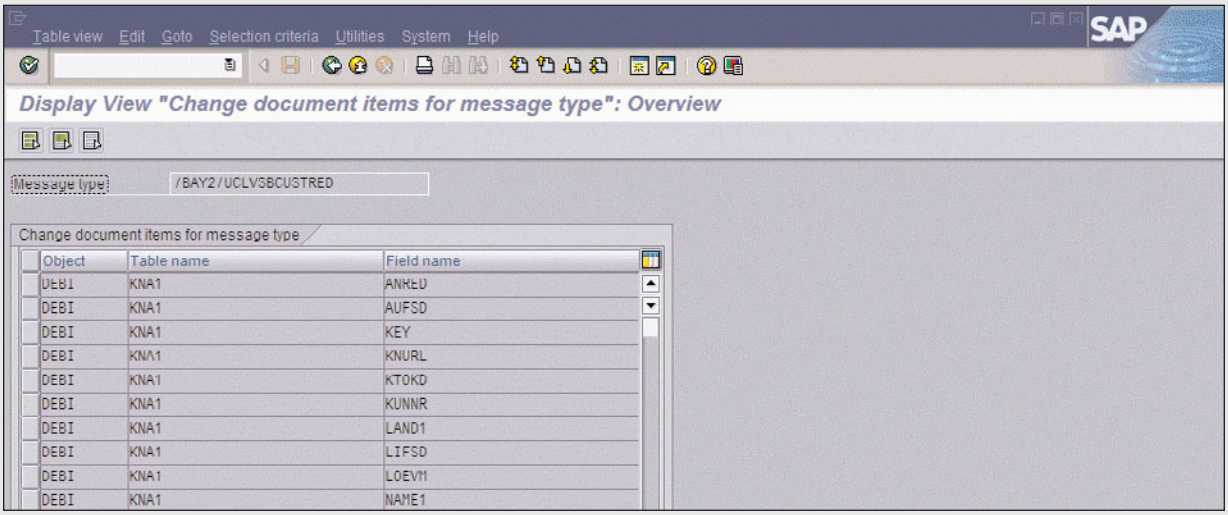

Figure 8 Identifying the table and fields included in the message type

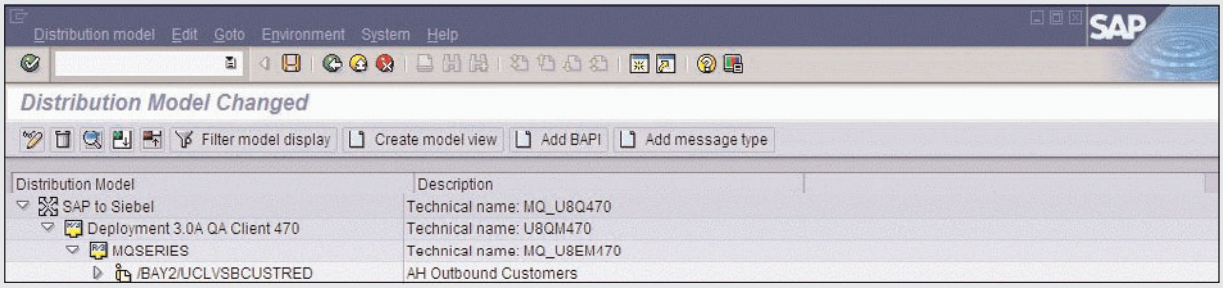

Figure 9 Setting up the distribution model for the ALE configuration

between the systems so the data exchange can happen. In the partner profile, you identify the systems that are involved in the exchange (i.e., the "partners") and the port that will be used to facilitate the exchange. To begin, use transaction BD82 (Generate Partner Profiles).

The profile in **Figure 10** (on the next page) shows that we are using the standard DEBMAS05 IDoc, the extension is /BAY2/UCLVDEBMAS05, and the message type is /BAY2/UCLVSBCUSTRED. Also indicated in the profile is that we are using a transactional RFC (tRFC) port connected to a third-party EAI tool, in our case, the Siebel system.

> ***Note!***
>
> The partner profile is part of any normal IDoc configuration and may apply to EDI as well as ALE.

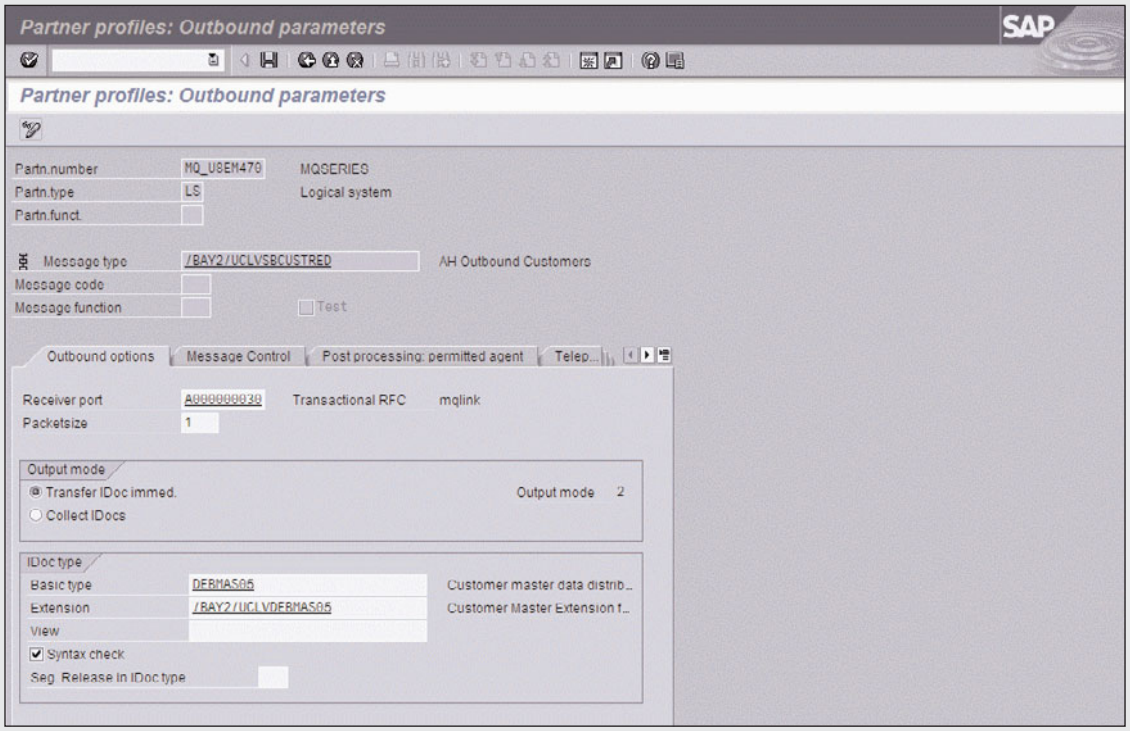

Figure 10 Setting up the partner profile for the ALE configuration

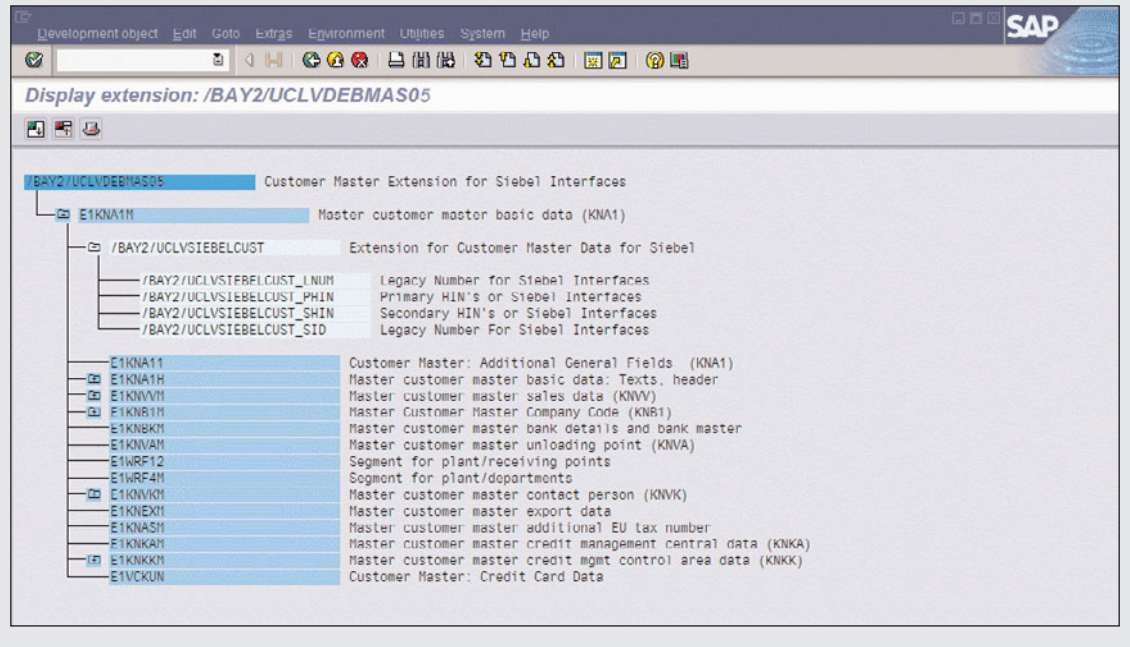

Figure 11 Displaying the characteristics of the extended IDoc

Let's take a closer look at the extended IDoc using transaction WE30 (IDoc Type). **Figure 11** shows an overview of the characteristics of the extended IDoc.

As you can see, we have created a custom segment to hold all of the characteristics of the customer class.

Step 10: Configure the user exit to capture the classification data and to append the data to the segment

Remember the user exit mentioned at the beginning of the article, EXIT_SAPLVV01_001? This is where you encapsulate the code by the message type and insert the fields from the classification data. There are several BAPIs available for doing this — one is `BAPI_CLASS_GET_CLASSIFICATIONS`, and another would be `BAPI_OBJCL_GETDETAIL`. The BAPI you use will depend on your specific business requirements.

Figure 12 is an example of the code needed for capturing the classification data and appending it to the custom segment. As you can see, this example uses `BAPI_CLASS_GET_CLASSIFICATIONS`.

```
CONSTANTS:   c_011(3)      TYPE klassenart VALUE '011',
                           c_kna1(4)    TYPE c VALUE 'KNA1'.

DATA: i_class_objects LIKE bapi_class_objects
                        OCCURS 0 WITH HEADER LINE,
*
      i_obj_classfctn TYPE STANDARD TABLE
                      OF bapi_object_values
                      WITH NON-UNIQUE KEY name_char
                      WITH HEADER LINE .
*
IF idoc_type = 'DEBMAS05'.

* Append the IDOC

        i_class_objects-object_key  = e1kna1m-kunnr.
        i_class_objects-object_type = c_kna1.
        APPEND i_class_objects.
        CLEAR i_class_objects.

        CALL FUNCTION 'BAPI_CLASS_GET_CLASSIFICATIONS'
             EXPORTING
                  classtype            = c_011
                  classnum             = c_ahhinp
                  key_date             = sy-datum
             TABLES
                  object_classification = i_obj_classfctn
                  class_objects        = i_class_objects.
```

Figure 12 Capturing the classification data and appending it to the custom segment *Continues on next page*

***Figure 12** continued*

```
        SORT i_obj_classfctn BY name_char.
*
         READ TABLE i_obj_classfctn
              WITH KEY name_char = c_ah_hin_primary.
*
         IF sy-subrc = 0.
           x_sbcust-hinprm = i_obj_classfctn-char_value.
         ENDIF.

        idoc_data-sdata = x_sbcust.
        APPEND idoc_data.

ENDIF.
```

We've now completed the steps required to implement the first solution — we have created an extended IDoc for including classification data in master data scenarios. Before we move onto the second solution, we need to test the results of our work so far.

Testing the solution

To test our solution, we are going to make a change in the customer master data. To begin, use transaction XD02 (Change Customer) to bring up a customer (customer 5237547 in the example) and to make a change to the data. Let's first take a moment to look at the classification data that's already there (see **Figure 13**).

Note that the characteristic AH HIN Primary has the value BT0000035 for class type 011. Now, let's change something in the non-classification data. As you can see in **Figure 14**, I have entered the prefix "Mr."

Saving this change to the customer master data triggers the IDoc. An IDoc is created for that customer, and based on what is found in class type 011 (the customer class type), the user exit will pull the characteristics and insert the custom segments respectively, as shown in **Figure 15** on page 40.

As you can see in Figure 15, the IDoc has successfully captured the classification data within the user exit, appended it in the correct spot to the custom segment, and will now be sent to the Siebel system via the port predefined in the partner profile.

For many of you, the solution we have just walked through may already be in place. However, it is essential that it not only be in place, but also work correctly before you can attempt the next solution, so be sure to test it thoroughly before moving on to the next part. Now comes the solution you all have been waiting for: triggering the correct IDoc when changes are only made to the classification data.

Triggering an IDoc when only classification data changes

This section provides a solution to the problem that arises when only classification data is changed, which eliminates the need for the "workaround" described in the introduction (i.e., to always edit unimportant customer master data when you edit classification data in order to trigger the IDoc). The goal is that when a user changes the classification data only, the same IDoc is produced without having to change something in the customer master data.

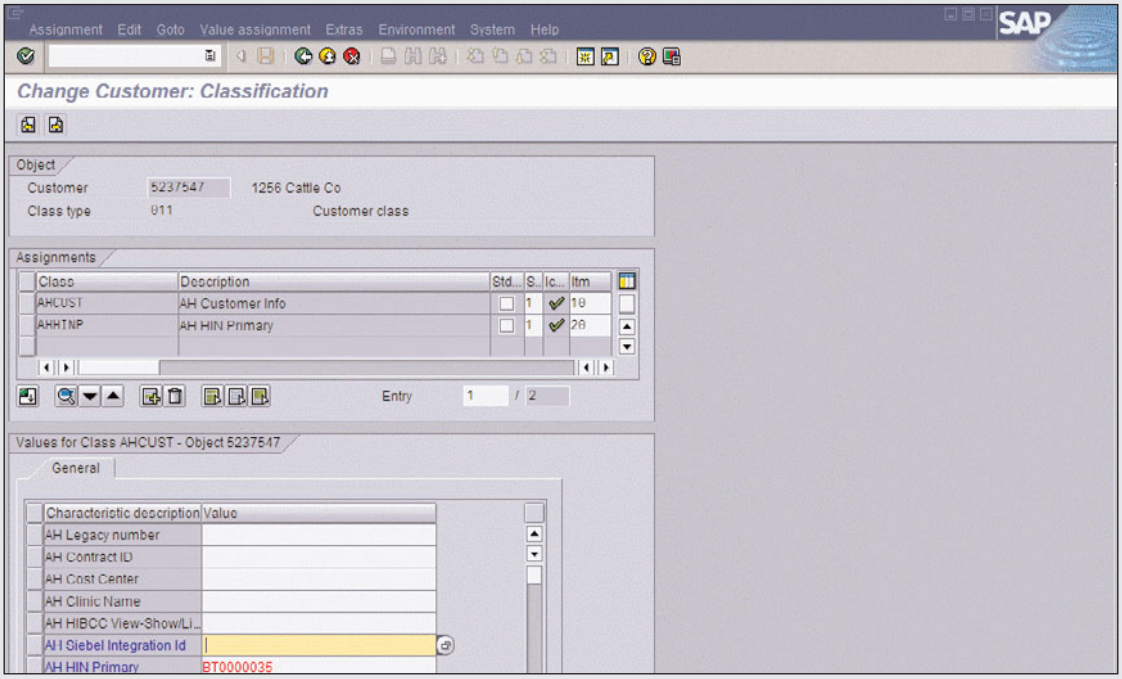

Figure 13 Classification data before making a change to the customer master data

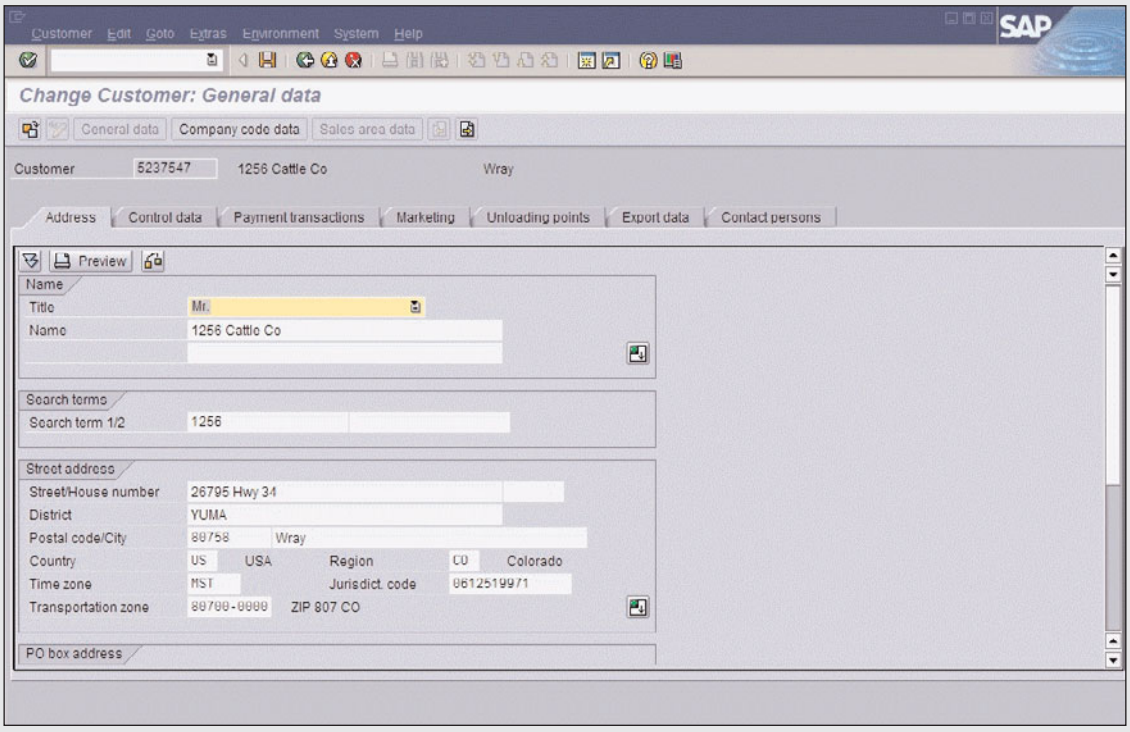

Figure 14 Changing non-classification data in the customer master data

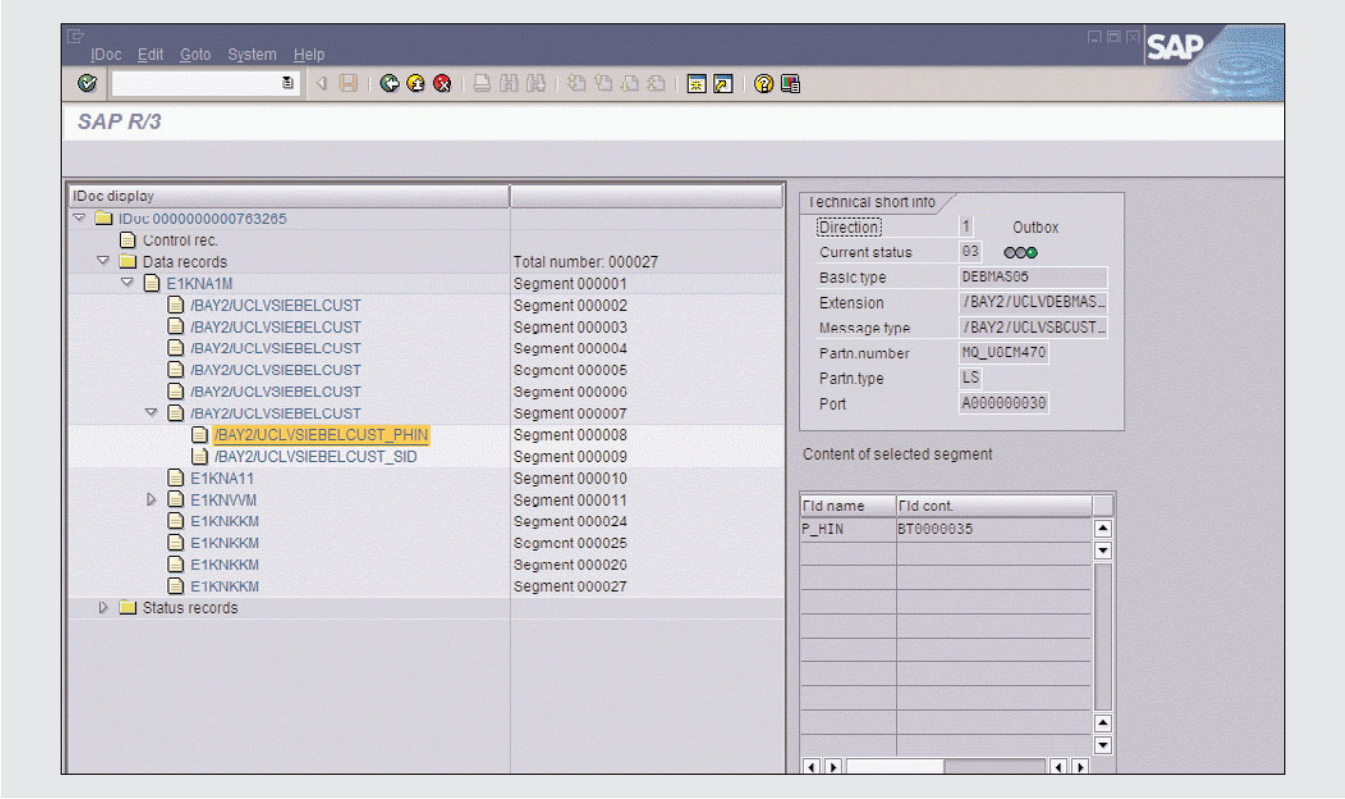

Figure 15 IDoc triggered by the change to the customer master data includes the classification data

As explained at the beginning of this article, if a user makes a change to the classification data, no IDoc is triggered, even with the extended IDoc for adding classification data in place, so the customer or vendor master data is not in synch with the associated classification data. Let's look at how you can address that problem and trigger the IDoc the user expects.

To create a function module that will generate an IDoc when only classification data has been changed, follow these steps:

1. Identify the current shared master data (SMD) function module.
2. Copy the current SMD function module and save it as a new function module.
3. Turn on change pointers for the CLFMAS (Classes) message type.

Note!

SAP does provide the ability to distribute master data between SAP systems using the message types CLFMAS (Classes) and CHRMAS (Characteristics and character values) as their own ALE scenarios. But using or implementing either of these to address the issue of triggering the correct IDoc when classification data changes would necessitate an additional interface and would require a new IDoc, possibly EAI changes or additions, and target system modifications, which in the end is a great deal of work. The solution provided here is a better and more efficient way of handling this situation, and this solution leverages an existing function module, again, making the task easier and more technically sound.

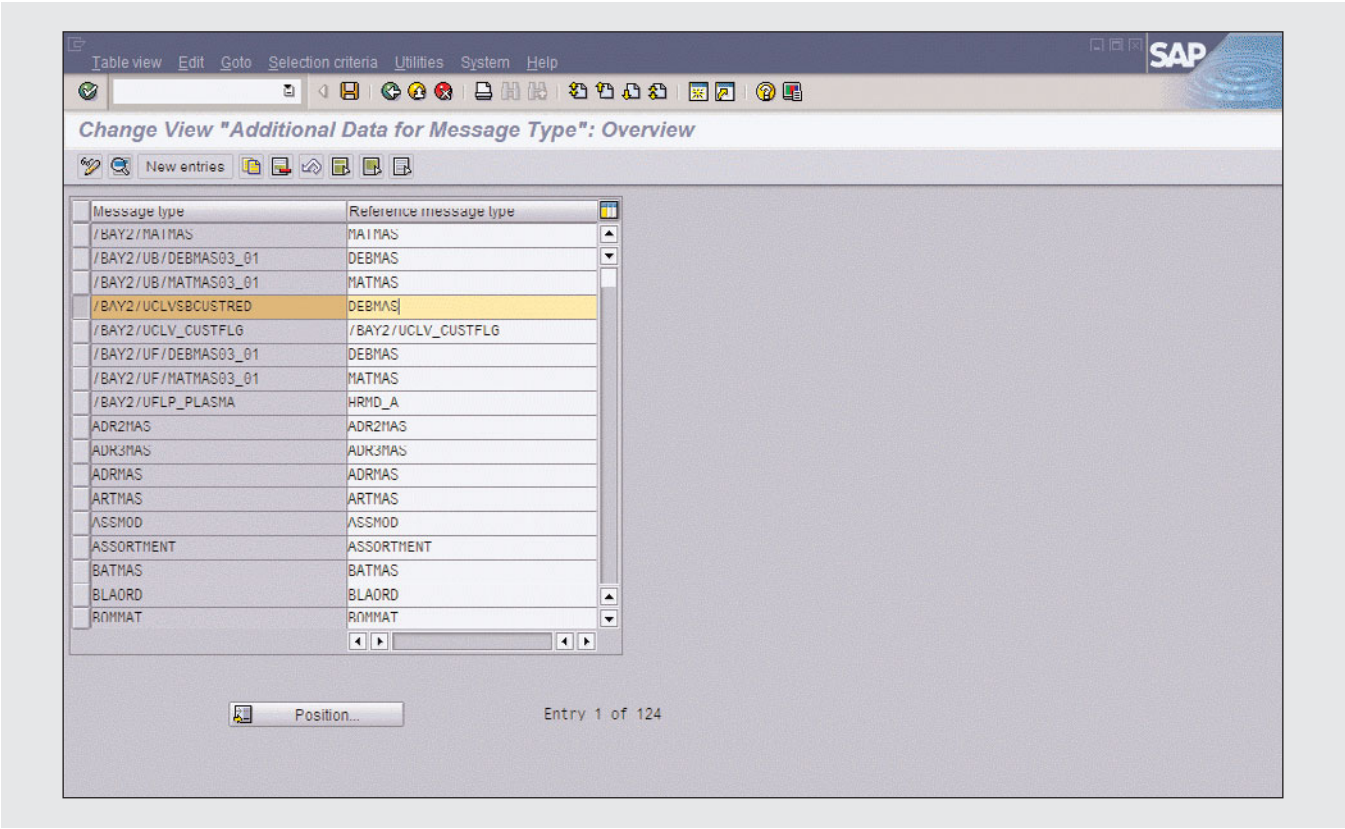

Figure 16 Displaying the function modules running on your SAP R/3 back-end system

4. Modify the new function module.

5. Implement and activate the new function module.

After completing these steps, you will have the ability to change classification data associated with this customer master data and produce the expected IDoc.

Step 1: Identify the current shared master data (SMD) function module

First, we need to find out what function module is being called for the message type in question. The standard message type for customers is DEBMAS, but for this example it is /BAY2/UCLVSBCUSTRED. If you do not know the name of the function module, use transaction BD60 (Additional Data for Message Type) to determine this information (see **Figure 16**).

The table shown in Figure 16 is a client-independent table and is configured in the respective client. Double-click on the line that represents the message type being used. As shown in **Figure 17**, the function module being called is MASTERIDOC_CREATE_SMD_DEBMAS. This is the standard function module in SMD that is used for customer master data distribution. This is the function module that we need to use to create one that will generate an IDoc.

Step 2: Copy the current SMD function module and save it as a new function module

Next, we make a copy of function module MASTERIDOC_CREATE_SMD_DEBMAS. (In Step 4, we'll customize the function module to address the issue of triggering the correct IDoc when

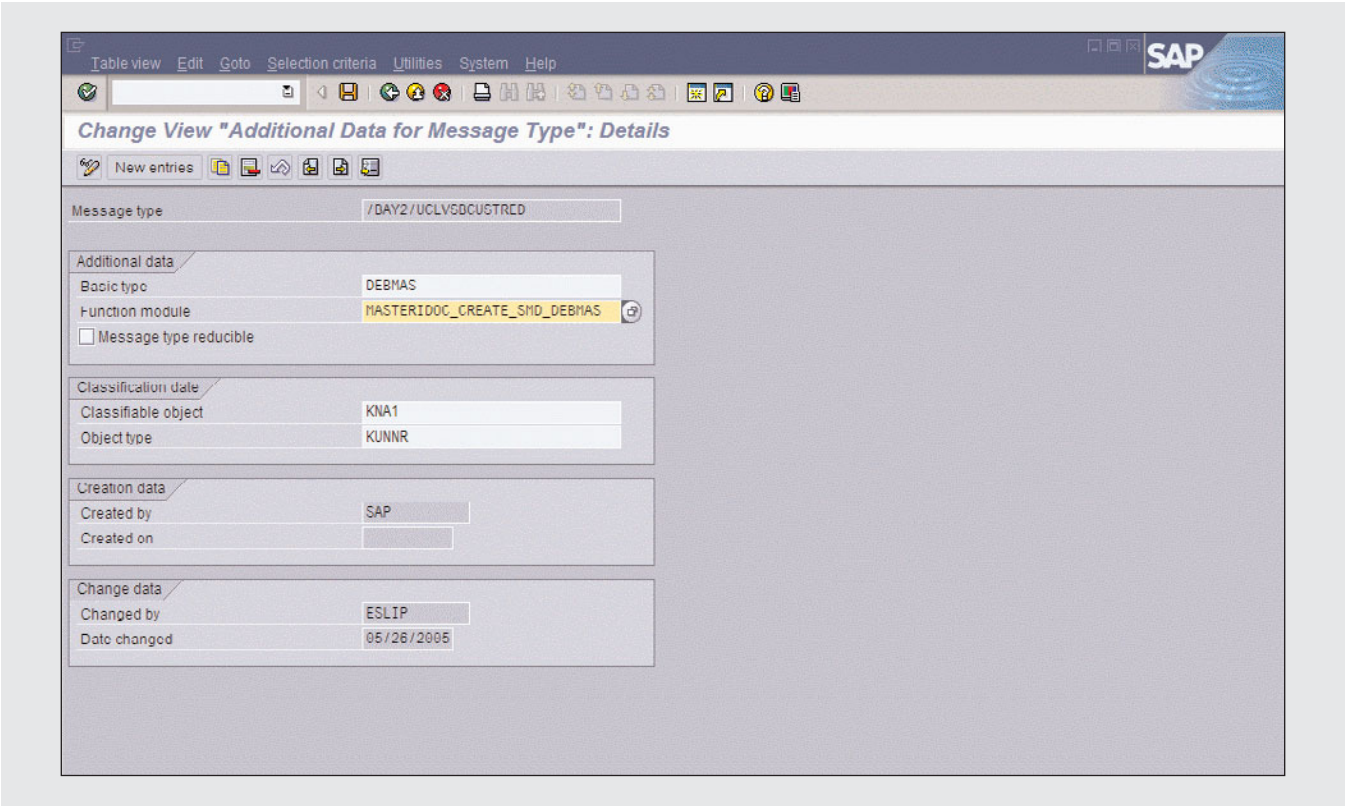

Figure 17 The MASTERIDOC_CREATE_SMD_DEBMAS function module being called

only classification data has been changed.) Enter a name for the new function module (in this example, /BAY2/CREATE_SMD_DEBMAS), as shown in **Figure 18**.

We will come back to this new function module in a moment, but first we need to make sure that changes to the classification data are tracked and implemented.

Step 3: Turn on change pointers for the CLFMAS (Classes) message type

We need to turn on change pointers for the message type CLFMAS (Classes), which is part of the MASTERIDOC_CREATE_SMD_DEBMAS function module we just copied. Use transaction BD50 (Activate Change Pointers for Message Type) to display the message types, as shown in **Figure 19**.

Make sure there is a checkmark in the Activate column for the CLFMAS message type. With the change pointers activated, we can return to the new function module to complete the modifications.

Step 4: Modify the new function module

Now, we need to add data statements to our new function module /BAY2/CREATE_SMD_DEBMAS (see **Figure 20** on page 44). The code identifies the internal table `itab_chgptrs` and tells the system to include the database table `BDCP` when executing changes. Also, we want the changes to be part of a second internal table, `itab_cpident`.

We next need to add the code that will read the change pointers for this message type. We'll add the code, which is shown in **Figure 21** on page 44, *after* the standard SAP code. Note that SAP accumulates the change pointers for /BAY2/UCLVSBCUSTRED

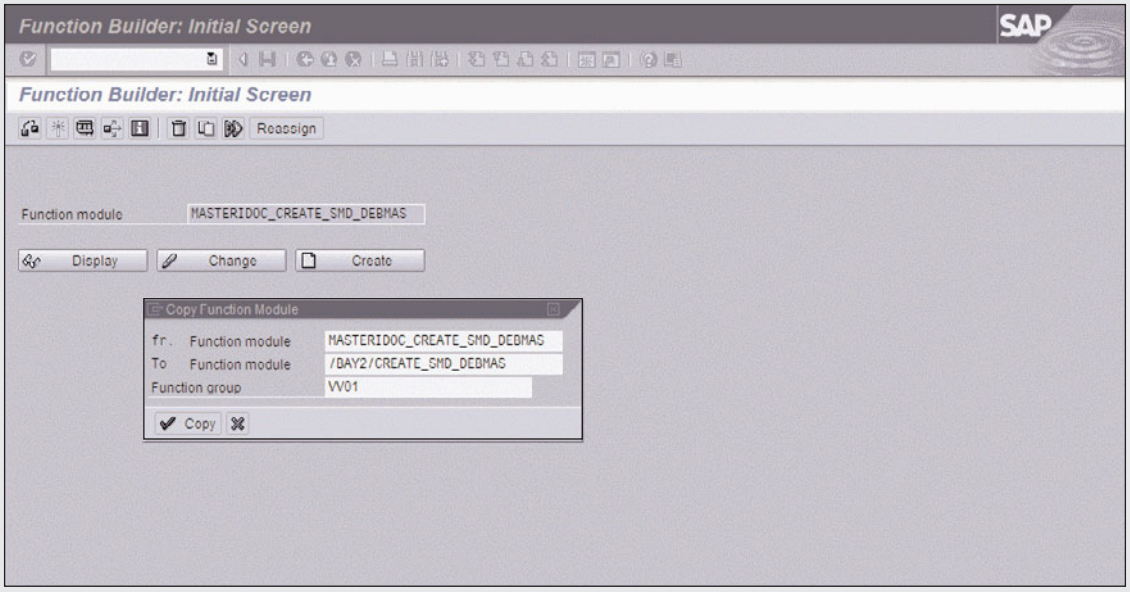

Figure 18 Creating a new function module from an existing one

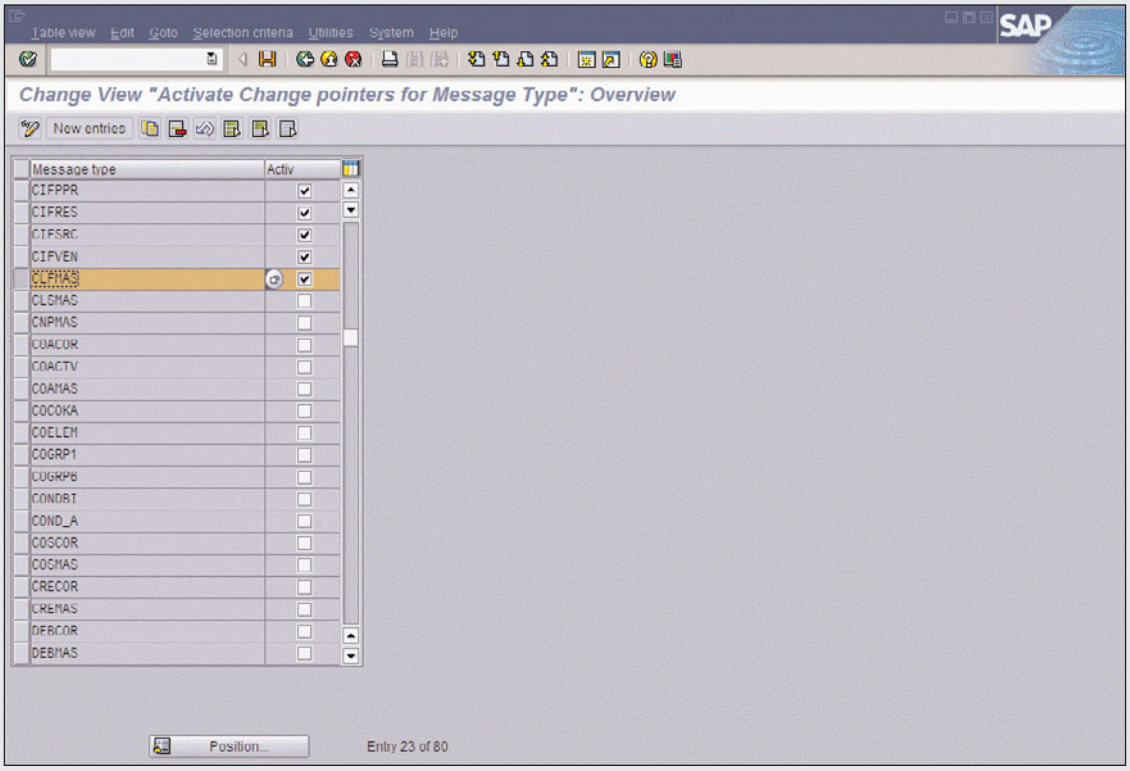

Figure 19 Displaying message types in the MASTERIDOC_CREATE_SMD_DEBMAS function module

```
DATA: BEGIN OF itab_chgptrs OCCURS 10.
         INCLUDE STRUCTURE bdcp.
       END OF itab_chgptrs.

 DATA: BEGIN OF itab_cpident OCCURS 10,
         cpident LIKE bdcp-cpident,
       END OF itab_cpident.
```

Figure 20 Adding data statements to identify the internal tables that will store classification changes

```
* read all not processed change pointers for the given message type,
* object class DEBI
  CALL FUNCTION 'CHANGE_POINTERS_READ'
       EXPORTING
            change_document_object_class = c_cdobjcl_debi
            message_type                 = message_type
            creation_date_high           = creation_date_high
            creation_time_high           = creation_time_high
            read_not_processed_pointers  = c_x
       TABLES
            change_pointers              = t_chgptrs.
```

Figure 21 Storing changes to the internal table t_chgptrs

```
  CLEAR   itab_chgptrs.
  CLEAR   itab_cpident.
  REFRESH itab_chgptrs.
  REFRESH itab_cpident.
```

Figure 22 Clearing and refreshing custom data areas

message type in the internal table `t_chgptrs`, which stores the information for all changes to names, addresses, phone numbers, etc.

Following the code that reads the change pointers, we add the code needed to clear and refresh the custom data areas, as shown in **Figure 22**.

We are now ready to read any and all classification changes using our modified message type CLFMAS (which we enabled in Step 3) as shown in **Figure 23**. Note that the results are being externalized to our custom table `itab_chptrs`.

Because you can have classification changes for


```
* read all not processed change pointers for the given message type,
* object class CLFMAS
  CALL FUNCTION 'CHANGE_POINTERS_READ'
       EXPORTING
            change_document_object_class = 'CLASSIF'
            message_type                 = 'CLFMAS'
            creation_date_high           = sy-datum
            creation_time_high           = '235959'
            read_not_processed_pointers  = c_x
       TABLES
            change_pointers              = itab_chgptrs.
```

Figure 23 Using the modified message type CLFMAS to read any changes to classification data

```
  LOOP AT itab_chgptrs.
    CLEAR: t_chgptrs.
    MOVE-CORRESPONDING itab_chgptrs TO t_chgptrs.
    MOVE t_chgptrs-tabkey+4    TO t_chgptrs-tabkey.
    MOVE 'DEBI'                TO t_chgptrs-cdobjcl.
    MOVE t_chgptrs-tabkey      TO t_chgptrs-cdobjid.
    APPEND t_chgptrs.
    MOVE   t_chgptrs-cpident   TO itab_cpident.
    APPEND itab_cpident.
  ENDLOOP.
```

Figure 24 Looping through our custom table to generate the IDoc

objects other than customers, we need to delete all entries that are not tied to classification object KNA1 by entering the following code in the function module:

```
DELETE itab_chgptrs WHERE tabname NE 'KNA1'.
```

Here is where the magic occurs. We want to loop through our custom table and insert entries into the SAP table, so it looks as if SAP put them there in the first place (see **Figure 24**). We also want to save the `CPIDENT` field for later use, so we append the row to the second custom internal table `itab_cpident`. We will use this table at the end to set the status as processed, so we do not keep getting the same changes over and over for subsequent runs.

This internal table will then be passed down the line to ALE code that will use the `t_chgptrs` table to generate the IDoc. There is still one last piece of code, however. We have to ensure that we set the status for the change pointers we create manually as processed, so we need to enter this code, shown in **Figure 25** on the next page, *after* the standard code in the function module.

Step 5: Implement and activate the new function module

Now, regardless of whether a user changes only

```
* write status of all processed pointers
    CALL FUNCTION 'CHANGE_POINTERS_STATUS_WRITE'
         EXPORTING
              message_type           = message_type
         TABLES
              change_pointers_idents = t_cpident.

    COMMIT WORK.

    CALL FUNCTION 'CHANGE_POINTERS_STATUS_WRITE'
         EXPORTING
              message_type           = 'CLFMAS'
         TABLES
              change_pointers_idents = itab_cpident.
    COMMIT WORK.

    CALL FUNCTION 'DEQUEUE_ALL'.

  ENDIF.

  MESSAGE ID 'B1' TYPE 'I' NUMBER '038'
          WITH created_m_idocs message_type.
  MESSAGE ID 'B1' TYPE 'I' NUMBER '039'
          WITH created_c_idocs message_type.
```

Figure 25 Passing the t_cpident table so that the change pointers are marked as processed

classification data, just general customer data, or both, an IDoc will be produced and sent via the partner profile and distribution module entries already in place. That's it!

Let's look at an example. We will use the same customer we used earlier (customer 5237547), and use transaction XD02 (Change Customer) to navigate to the classification data (see **Figure 26**).

Let's change characteristic AH HIN Primary from BT0000035 to BT0000036. Remember, we are only making a change to a classification characteristic and not to any customer data. The message shown at the bottom of **Figure 27** indicates a change has been successfully made.

Having saved the change, let's see what IDoc has been generated. Use transaction WE05 (IDoc Lists). **Figure 28** (on page 48) shows that IDoc 0000000000776144 has been generated and the P_HIN characteristic reflects the new value (BT0000036).

You now have resolved the issue of triggering the same IDoc when only classification data has changed. This will help to keep your master data synchronized and accurate.

Conclusion

This article has shown you how you can start distributing classification data along with your customer and vendor master data changes, rather than transmitting it separately. Unless you are running SAP NetWeaver

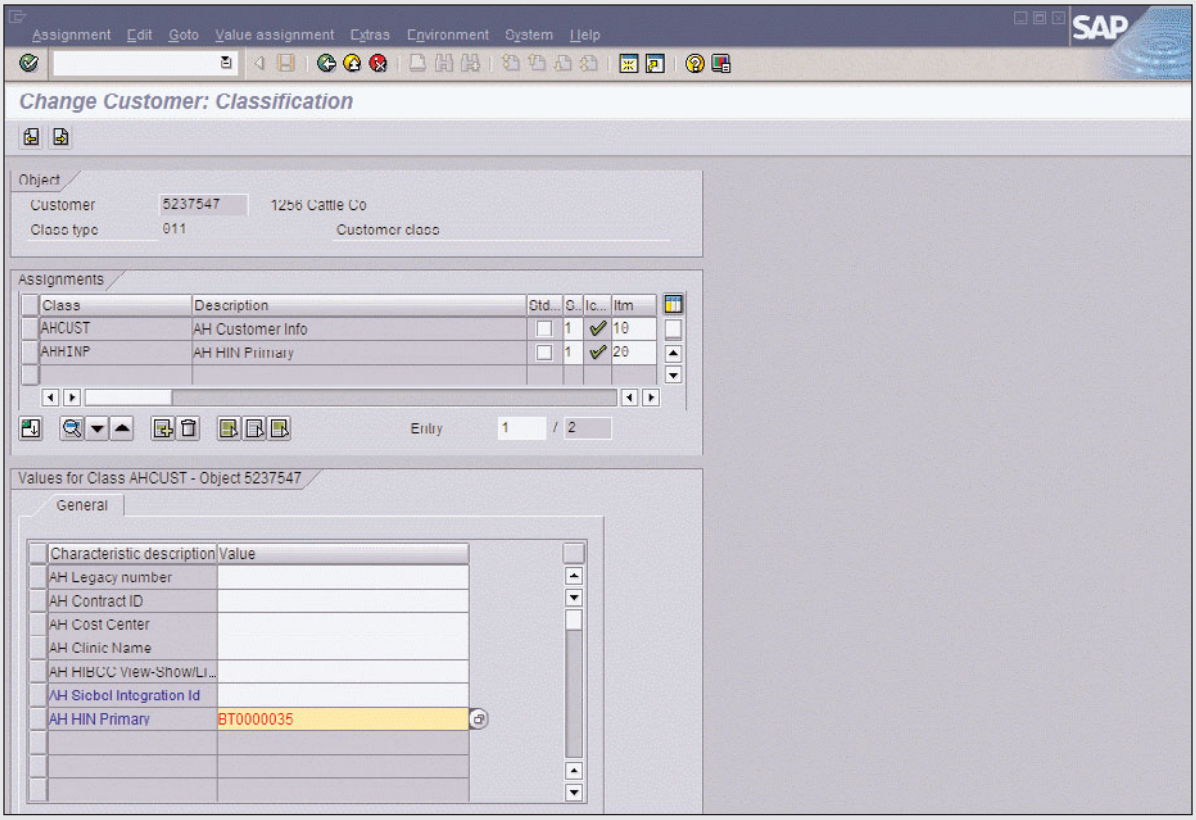

Figure 26 Classification data before a change is made

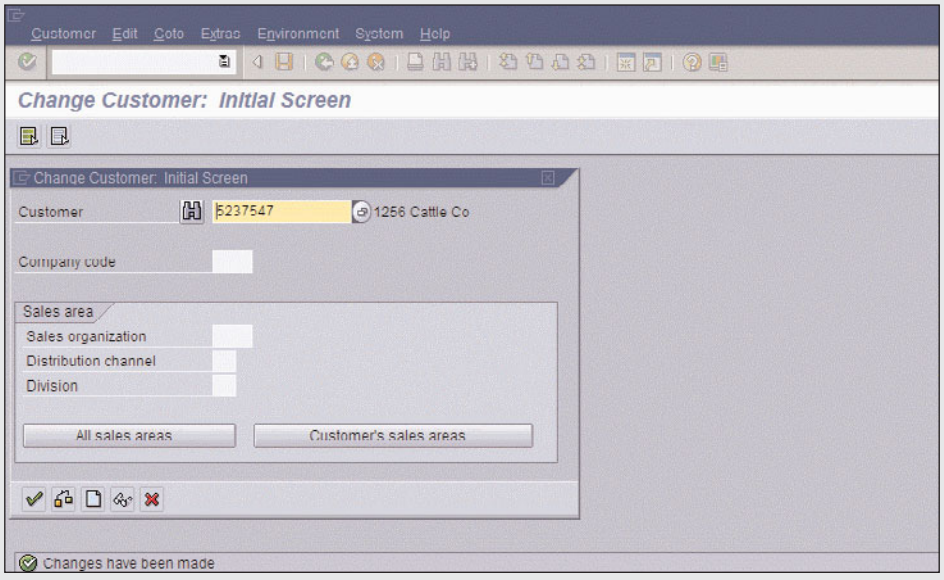

Figure 27 Changing classification data only

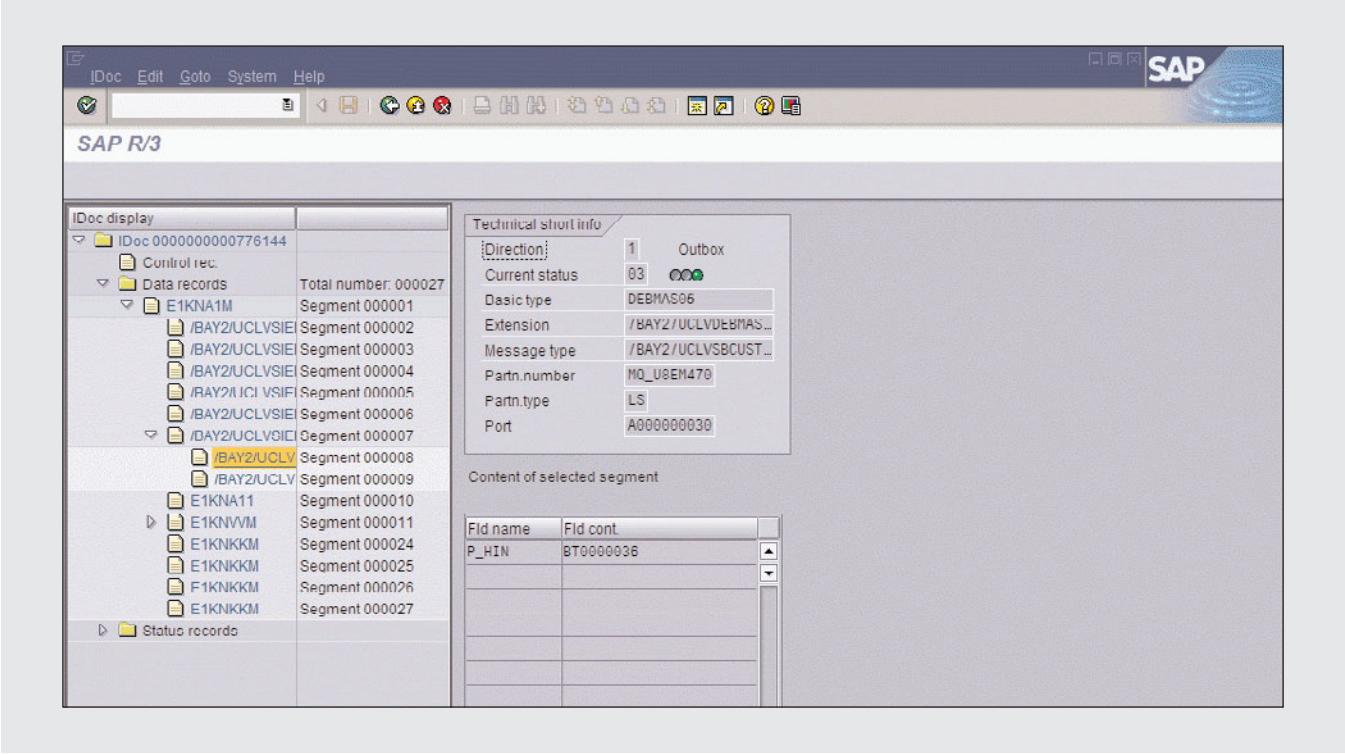

Figure 28 IDoc triggered for classification data change

'04, which includes SAP NetWeaver Master Data Management (MDM) 5.5, there are no standard SAP classification ALE distribution scenarios, so a custom solution, such as the one described here, is required.

You also learned how to overcome the common "gotcha" when changes are made only to classification data. Transmitting classification data only when master data record changes occur can lead to data integrity problems, since users might maintain classification data independently.

So, where do you go from here? Take a look around your enterprise and see if any business processes are exchanging customer or vendor master data with third-party or other SAP systems. If they are not sending classification data along with the customer or vendor master data, you now know how to enable that ability. If they are, see if they are running into the gotcha we just explored. You will now be able to fix the problem, adding value to the business processes that deal with master data and classification in your organization.

