Data Transformation in SAP Standard ALE Distributed Business Processes: How to Ensure an Efficient, Effective Implementation

Arthur Wirthensohn

Arthur Wirthensohn is a senior consultant at EDS Switzerland and a member of EDS's international Technical Leadership Network. Currently, he works as project manager and is the technical lead of the EAI department, which delivers SAP-related services such as ALE/EDI, SAP Business Workflow, Web Application Server, Data Migration, and ABAP Programming Services.

(complete bio appears on page 80)

SAP's Application Link Enabling (ALE) is an integrated toolset for enabling the distribution of data between SAP systems as well as between SAP and non-SAP systems. It supplies all the features necessary for exchanging and monitoring data, including tools for transforming data[1] to meet the requirements of each partner[2] in a distributed business process. If you are implementing a distributed business process that does not require any transformation of the data to be distributed, or that requires only a small amount, then setting up the distribution via ALE is fairly simple.[3] It's also fairly easy to estimate the feasibility of the implementation and the amount of time and effort it will require. However, if the business process requires lots of data transformation, it's a different story.

ALE provides a variety of tools and features (filters, conversion rules, program exits, and the like) for transforming the data that is exchanged between systems, each with its own advantages and limitations. Understanding these tools — what they can do and how to use them — is a prerequisite for being able to do three very important things:

- Determine whether you will be able to carry out your data distribution using a standard ALE process by utilizing the provided data

---

[1] A data transformation might consist of changing the data within a given IDoc data structure (e.g., changing the representation of a unit of measurement), or it might mean filtering out data that is not relevant to the applications or processes that will receive the data.

[2] In an ALE connection between SAP systems like R/3, CRM/SRM, BW, or APO, a partner is a logical system addressing exactly one client of an SAP system.

[3] The main challenge in inter-company distributed business scenarios is not a technical one; it is having all parties communicate with each other completely and clearly about the distributed business process and reaching a detailed agreement on what constitutes that process.

transformation tools. If you can't, you will need to employ an application integration tool like the SAP Business Connector or the state-of-the-art SAP Exchange Infrastructure, which is part of SAP's NetWeaver product portfolio. You don't want to move to one of these tools unnecessarily, however, because along with the increased functionality they provide, they also can bring increased implementation, maintenance, and system management costs.

- Determine how much time and effort it will take to implement the distribution using ALE.
- Efficiently and effectively implement your data distribution. Data transformation in ALE can be easy if you know the tools, but complex and time-consuming if you do not.

The purpose of this article is to equip you with the knowledge and skills you need to accomplish all three of these objectives. You'll learn everything you need to know about ALE's data transformation tools and features — their capabilities, relative merits, and how to use them. Most important, I'll show you how to evaluate each tool for your project requirements, so that you can select the tools that ensure an easy-to-implement, cost-effective, and highly maintainable solution.

In my experience, just knowing what each tool can do and how to use it isn't enough. You also need to know *when* to use one tool rather than another, even when they are both capable of getting the job done. For each data transformation, you want to use the tool that has the fewest performance and processing costs, saving more "expensive" tools for those tasks that can't be accomplished any other way. That's why I provide you with a structured approach (a set of principles and procedures for evaluating and implementing tools) that you can use for selecting the right tool or combination of tools for your own implementation, no matter how complex the data transformation requirements are. Then, to help you learn by doing, I take you through a step-by-step data transformation example that puts the approach I recommend into action.

We'll begin by looking at the information flow of a business process that is distributed via ALE, paying particular attention to how and when data transformations take place.

✓ ***Note!***

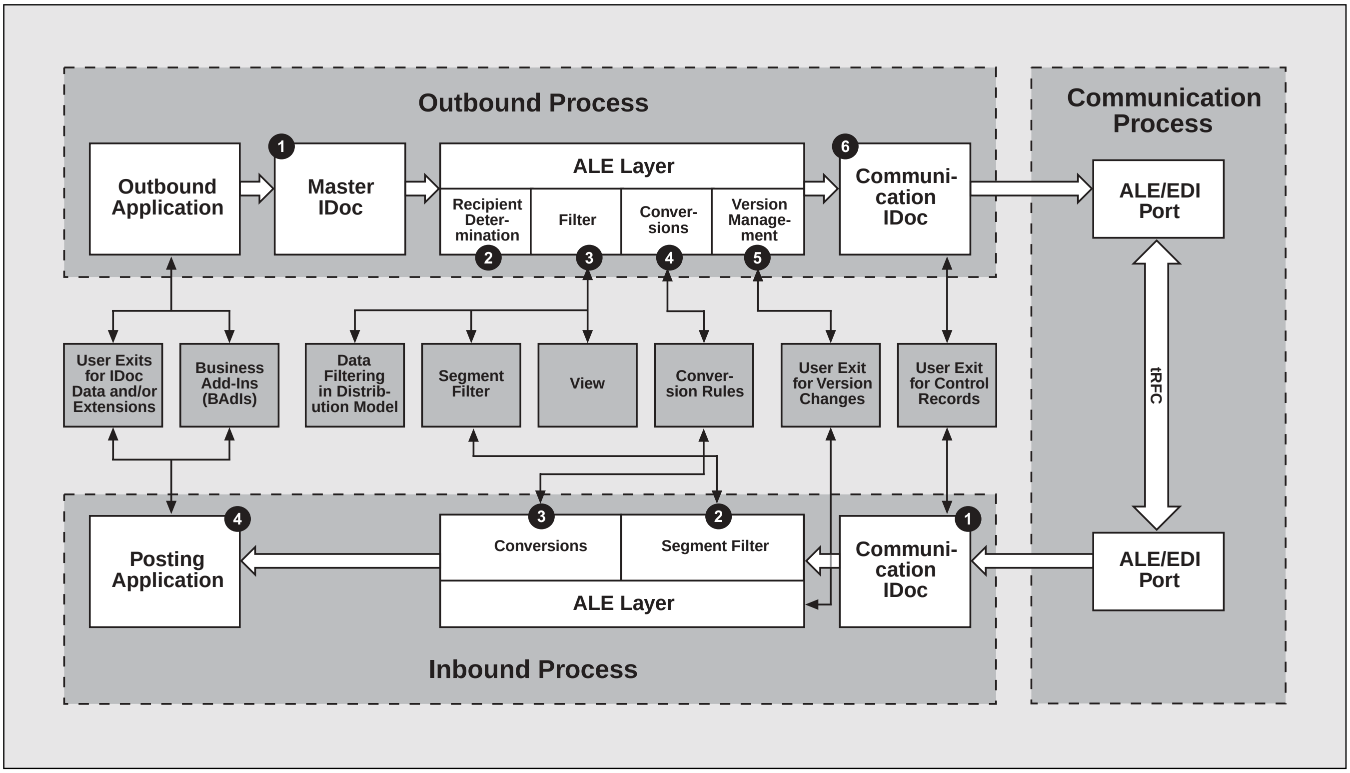
Though the focus of this article is on transforming SAP standard messages and not IDoc Extensions or custom IDoc/ALE processing, all of the information in this article can be applied to extended and custom ALE distributed business processes as well.

How Data Is Distributed with ALE

When data is distributed from one SAP system to another via ALE, it passes through an information flow like the one depicted in **Figure 1**. As the diagram shows, ALE has three major processes for moving data from a sender system to a receiver system:

- **Outbound process:** The outbound process begins with the creation of an Intermediate Document (IDoc) containing the data to be transmitted. It ends with the creation of a communication IDoc, which contains the data ready to be transferred to the receiver system.
- **Communication process:** The communication process encompasses the (physical) transportation of the IDoc from the sender to the receiver system. ALE supports several means of communication, each controlled by a different port type (the port specifies the technical properties of the communication process).
- **Inbound process:** The inbound process directs the IDoc to the appropriate posting application, while applying certain checks to the IDoc.

Figure 1 Information Flow in an ALE-Enabled Distributed Business Process



We'll look at each of these processes in turn to get a detailed picture of what happens to the data as it travels from the sender system to the receiver system.

The Outbound Process

The outbound process takes place in six distinct stages, as shown in Figure 1:

1. The outbound application creates a master IDoc containing the data to be transferred.
2. The ALE layer determines the destination for the data (one or more receiver systems).
3. The ALE layer applies filters to the data.
4. The ALE layer applies data conversion rules.
5. To guarantee version compatibility between the sender and receiver systems, ALE performs version management.
6. ALE completes the process by creating a communication IDoc.

Stage 1: The Outbound Application Creates a Master IDoc

In the first stage of the outbound process, the outbound application — an ALE function module or a BAPI — creates a master IDoc containing the data to be transferred.[4] The outbound application can be used for distributing either master data or business documents.

Usually, outbound applications are triggered to distribute **master data** by one of the following:

- The processing of change pointers

[4] For more details on ALE processing and IDocs, see the SAP online help at **http://help.sap.com** (choose *SAP Library* → *Basis Components* → *Middleware (BC-MID)* → *Application Link Enabling (BC-MID-ALE)* → *ALE Introduction and Administration* → *ALE Integration Technology* → *Message Distribution*).

- Workflow events[5] that are coupled with receiver function modules for data replication

- Programs that create data (e.g., *RBDSEMAT* for material master) for initial data transfer

There are various ways of triggering outbound applications to distribute **business documents**. For instance, in MM and SD modules business documents are triggered through the issuing of output — in other words, through the processing of table *NAST* with communication medium *A* (for ALE) or *6* (for EDI).[6]

An outbound application might have program exits for applying custom coding that manipulates IDoc data. These exits can be in the form of *user exits* or *Business Add-Ins*. The outbound application will call program exits for all of the basic IDoc types, but it will call program exits for a customer-defined extension of an IDoc only if the extension is defined in the partner profile[7] for the distributed business process. In that case, the program exit should be used to fill the extended segments of the IDoc with data.

To control the creation of the master IDoc, ALE uses a *distribution model*, which consists of *distribution views*. A distribution view is a technical definition of a distributed business process; it labels the process and specifies the sender system, receiver systems, and message types or BAPIs. A distribution view can be sent from the system regarded as the "maintenance" system to one or more receiver systems to ensure a consistent distribution scenario across the entire system landscape. A master IDoc will be produced by the outbound application only if a distribution view with the appropriate partner and message type/BAPI exists.

You create a distribution model in the ALE customizing of the maintenance system. The ALE layer[8] "reads" the distribution model to get the appropriate control information.[9] From the distribution model, all other relevant ALE customizing tables (e.g., port definition and partner profile) can be generated.

After creating the master IDoc, the outbound application passes it to the ALE layer, where the following activities take place:

- The recipient of the data is determined (the receiver system or application).
- Filters are applied to the data.
- Data conversion rules are applied.
- Version control (as it relates to the R/3 version of the receiver system) is performed.

Stage 2: The ALE Layer Determines the Destination for the Data

The ALE layer determines the recipients of the IDoc according to the distribution model.

Stage 3: The ALE Layer Applies Filters to the Data

After it determines the receiver systems, the ALE layer applies any existing filters to the data segments of the master IDoc. There are three types of filters:

- **Data filters in the distribution model:** A data filter in a distribution model is actually a set of filter conditions specified for a message type or BAPI. Each condition consists of a filter object and a value for the object. When the filter is applied, only segments that carry fields with the

[5] For details on how to set up online interfaces, see Amy Stapleton's article "Real-Time, Outbound Interfaces to Non-R/3 Systems Made Simple with Change Pointers, Message Control, and Workflow" in the Premiere Issue of this publication.

[6] Technically speaking, the ALE and EDI tools are the same. The only difference lies in how the partner is defined — in ALE the communication partner is a logical system; in EDI the communication partner is a business partner (a vendor or sold-to party, for example).

[7] The partner profile defines common administration attributes and the inbound and outbound parameters of a partner (i.e., logical system).

[8] The "ALE layer" refers to that part of ALE that performs the partner determination, applies filters and conversion rules, and performs version management on the data to be distributed.

[9] If you are not familiar with ALE, you can learn about it using the "Hands-on exercise for ALE distribution of material master" (see the "Documentation Resources" download available at **www.SAPpro.com**).

defined filter values pass the filter. For example, if the defined filter value is the string *DO-IT*, the specified field must contain that value to pass the filter. A distribution model filter is specific to a distribution view message.

- **Segment filters:** A segment filter will filter out any data segment of the IDoc with a given combination of message type, sender partner, and receiver partner in its control record. This type of filter is said to be "static," which means that a segment will *always* be filtered out if it has the specified message type and partners in its control record, regardless of its content.
- **Views:** A view is a subset of the segments belonging to a given message type. You can assign a view to any outbound partner profile that contains that message type. When a view is assigned, only the defined subset of segments will be sent.

These filter types are explained in more detail later, in the section on ALE's customizing tools.

Stage 4: The ALE Layer Applies Data Conversion Rules

After filtering the IDoc's data segments, the ALE layer searches for applicable conversion rules. A conversion rule is an abstract definition that specifies how to convert the data for all fields of one segment. For example, a rule might specify that the material number in field *Material Number* should be moved to field *Old Material Number* in segment *E1MARAM* (which contains the general data of the material master).

In order for a conversion rule to be applied, it has to be assigned to a specified combination of message type, sender partner, and receiver partner.

When a conversion rule is to be applied, user exit *KKCD0001* will be called during the conversion. This user exit allows you to add ABAP-coded conversions to the segment's fields. One component of this exit allows the manipulation of the sender data structure and another component allows manipulation of the receiver data structure. When using this user exit to add logic to a specific field (e.g., to select data from the database and move an extract of it to a specific field), you should avoid adding a rule to that field in the conversion rule itself. This would complicate the maintainability of the interface.

Stage 5: ALE Performs Version Management

Because SAP guarantees the compatibility of message types in different SAP R/3 releases, ALE has a version management feature (a "version changer") that looks for the correct message version for the receiver system before creating the communication IDoc. The receiver system's release information is specified in the port definition (this is discussed in detail in the upcoming section "The Communication Process").

The ALE layer calls user exit *ALE00001* (a generic ALE user exit) before it calls the version changer; ALE gives this user exit access to the data fields of the IDoc, which means that all of the IDoc's data segments (but not the IDoc's control record) can be manipulated in this user exit. For more details, go to transaction *CMOD* and navigate to the user exit documentation; examine the documentation for user exit *ALE00001* and its function module *EXIT_SAPLBD11_001*.

> ✓ ***Note!***
>
> *ALE calls user exit ALE00001 only if the IDoc type (which identifies the basic IDoc structure) <u>or</u> the CIM type (which identifies the structure of an extension) in the control record is different from what the ALE layer expects (the user exit documentation says that both the IDoc type <u>and</u> the CIM type have to be different from what is expected, but that is not true). The IDoc content in the control record is controlled by the outbound parameters of the partner profile (which is customizable). See the sidebar on page 67 for instructions on how to find out what IDoc type and CIM type the ALE layer expects.*

Stage 6: ALE Creates a Communication IDoc

Before creating the communication IDoc, the ALE layer calls user exit *SIDOC001*,[10] which allows you to modify the IDoc's control record, so that you can, for example, add control information required for sending the IDoc to an EDI subsystem.

Finally, the communication IDoc is created. The communication IDoc consists of one control record and one or more data records. The data records are derived from the data records of the master IDoc, including all transformations (filters, conversions, and so forth) that have taken place in the ALE layer of the outbound process.

The Communication Process

ALE's communication process controls the physical transportation of the communication IDoc from the sender system to the receiver system. ALE supports several kinds of communication, each controlled by a different port type:

- **Transactional Remote Function Calls (tRFCs):** The most common type of communication, tRFCs are used for communicating with SAP systems or application integration systems that provide RFC functionality (e.g., the SAP Business Connector and many SAP-certified application integration and EDI subsystems). Communication via tRFCs takes place over the tRFC port type. As the name implies, a tRFC communication features transaction control, which means that the RFC will be performed only once and the transaction will be secure (i.e., its state will be defined as "committed" or "rolled back"). In addition, tRFC communication has features like load distribution and the ability to perform repeated tries in case of failure.

- **File ports:** Another common communication technology, file ports are used for communicating with subsystems (mainly EDI subsystems) or partners that do not provide the SAP-proprietary RFC functionality. This port type is for sending or receiving a communication IDoc that is coming to or from a file system in the form of a sequential IDoc file or an XML data stream.

- **XML/HTTP:** With the SAP Web Application Server (as of Release 6.20), ALE supports communication using XML as the data format and HTTP as the transport protocol.

- **ABAP:** The ABAP port type is for sending or receiving the communication IDoc to or from a function module.

- **CPIC:** The CPIC port is for communication via Common user Programming Interface Communication, a protocol based on SNA (System Network Architecture), which is used for communication with mainframe-based architectures. The CPIC port type is mainly used for communication with R/2 systems (as of R/2 Release 5.0F).

A port defines the interface to a partner and is assigned to the outbound parameters of the partner profile. In a connection between SAP systems (e.g., *SAP R/3* ↔ *SAP R/3*), the port type is usually tRFC and has two control attributes. One of these attributes is the version, which will be checked by the ALE layer's version management feature in the outbound process. The other control attribute is the RFC destination. An RFC destination of type *3* (R/3 connection) specifies the partner system in terms of connectivity and logon information. Non-SAP R/3 systems that are tRFC-enabled, like the SAP Business Connector, are usually addressed by an RFC destination of type *T* (TCP/IP). The tRFC port calls a remote-enabled function module *IDOC_INBOUND_ASYNCHRONOUS* on the partner side, which receives the communication IDoc via table parameters for control and document (i.e., business) data.

[10] For details on user exit *SIDOC*, use transaction *CMOD* to refer to the documentation for this user exit and its function module *EXIT_SAPLEDI1_001*.

In this article, we assume that communication takes place via tRFC because it is the most convenient communication technology for connecting R/3 systems.

The Inbound Process

ALE's inbound process begins by reading the communication IDoc it has received from the communication process. This process takes place in four stages:

1. ALE reads the control information.
2. The ALE layer applies any segment filters.
3. The ALE layer applies conversion rules for data transformation.
4. The posting application processes the IDoc and posts its data to the receiver application or system as a business document.

Stage 1: ALE Reads the Control Information

Before the control data passes field checks, ALE calls user exit *SIDOC001*. As in the outbound process, the control record can be modified using this exit. You might want to modify control data in the inbound process in order to get the right processing parameters and options (e.g., CIM type).

After user exit *SIDOC001*, ALE's inbound process reads the partner information (sender and receiver) and the message type from the control data record. All further processing information will be determined from the partner profile's inbound parameters. The most important information in the partner profile's inbound parameters is the process code. The process code defines the processing type (task, function module, or process) and whether or not ALE services, such as filters and conversion rules, are used. The most common processing type is "function module," which is used for ALE inbound function modules (e.g., *IDOC_INPUT_MATMAS01* for material master) and BAPIs (*BAPI_IDOC_INPUT1* for all ALE inbound processing with BAPIs).

At the start of IDoc inbound processing, it's possible to have ALE again call user exit *ALE00001* (a generic ALE user exit). The same conditions must be met here as in the outbound process for this user exit to be called — i.e., the IDoc's control record must contain a different IDoc type or CIM type than what is expected by the ALE layer (refer back to the note on page 55). For the inbound process, the sender partner supplies the IDoc content in the control record.

Stage 2: The ALE Layer Applies Any Segment Filters

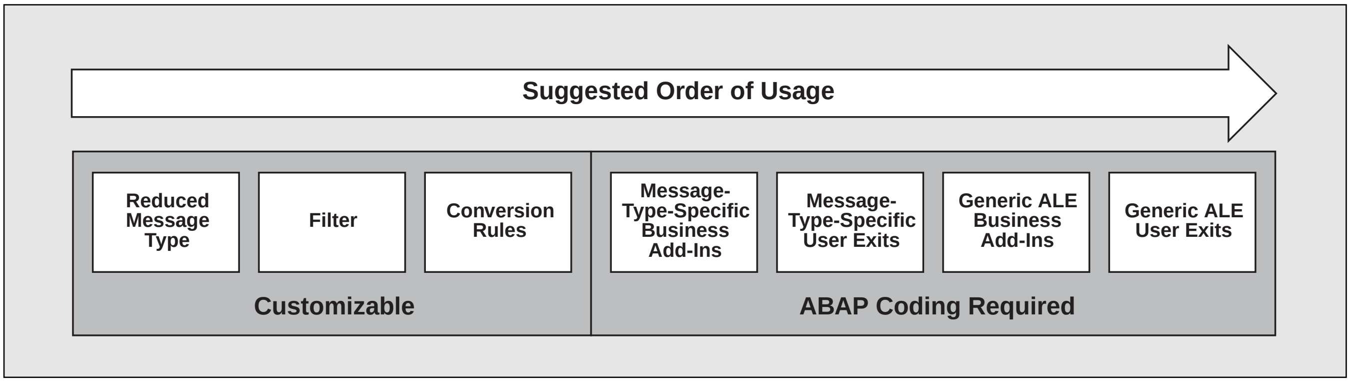
If customized segment filters exist, the ALE layer applies them. Just as in the outbound process, a segment filter is static, filtering segments of IDocs with a control record that has the given combination of message type, sender partner, and receiver partner. Later, you'll see why sometimes it makes sense to apply a segment filter in the inbound process, rather than the outbound process.

> ✓ ***Note!***
>
> *Other than segment filters, none of the filters mentioned in the discussion of the outbound process can be used in the inbound process.*

Stage 3: The ALE Layer Applies Conversion Rules

As in the outbound process, the ALE layer looks for conversion rules that can be applied. Technically, the same kinds of conversions can be performed in the inbound process as in the outbound process. As a practical matter, it might simply make more sense for the receiver partner's business users to specify and perform their own data transformations.

Figure 2 *Implementation Approach*



Stage 4: The Posting Application Processes the IDoc and Posts Its Data

After the conversion rules have been applied, the posting application processes the IDoc's data. The result is a newly created or changed business document, such as a material master or a sales order.

At this point, the posting application might call application-specific program exits (user exits or Business Add-Ins). The posting application will call program exits for an IDoc extension only if the extension is defined for the distributed business process (in the CIM type field of the IDoc's control record). In that case, the program exit should be used for adding the data of the extended segments to the business document.

Selecting and Using ALE's Data Transformation Tools — A Structured Approach

ALE provides a wealth of tools for transforming message data as it moves from sender to receiver. The challenge is to choose the right tool (or tools) for each required transformation. By definition, the right tools are the ones that make the most sense in terms of feasibility, cost-effectiveness, and maintainability. I recommend a structured approach to evaluating and selecting data transformation tools that is based on these principles:

- ☑ Filter out what you do not need.
- ☑ Utilize customizing tools before applying custom coding.
- ☑ Where possible, utilize application-specific program exits before utilizing generic program exits.
- ☑ Employ a Business Add-In, if one exists, in preference to a user exit.

The diagram in **Figure 2** shows the recommended order for evaluating the tools. First, try to determine whether or not you can solve your data transformation requirements with customizing tools like filters or conversion rules. Then, if these tools are not flexible enough to meet your requirements, search for an appropriate program exit (a user exit or Business Add-In) you can use to meet whatever requirements cannot be met using customizing tools. Add your transformation rules to the program exit in the form of ABAP coding and activate the program exit. Adhering to this approach will guarantee an efficient data transformation implementation within SAP standard ALE.

You can also use the suggested order as a checklist when you need to prove the feasibility of an implementation. Just go through the tools one by one and check to see if your needs can be met. If you still have some open issues when you have reached the last tool, you can be relatively sure that you have to do one of the following:

- Extend an IDoc type.
- Build your own ALE interface.
- Implement a third-party tool for data transformation (i.e., an application integration tool like the SAP Business Connector or SAP Exchange Infrastructure).

Of course, if you have to build your own ALE interface, the time and effort required to implement your data transformations will rise considerably.

> ✓ ***Tip***
>
> *If you need to implement an application integration tool in order to connect various processes and/or applications, then it is much better — in terms of maintainability — to perform your data transformations in the application integration tool, rather than in ALE. In the application integration tool, you will be able to define a master message and map each process or application's data to that master. Then, if there is a change in one system, you just have to change the mapping rules for the affected system.*

Tools for Customizing Your Data

ALE's customizing tools for data transformations are summarized in **Figure 3**. They include a reduced message type, filters, and conversion rules. Let's take a closer look at each of these tools in turn.

Reduced Message Type

A reduced message type allows you to create a customizable subset of an IDoc's master data. Unlike a filter, the reduced message type does not create its data subset from the entire IDoc. Rather, it consists of a subset of fields and segments right from the beginning, when the IDoc is created. There are two main reasons to use a reduced message type:

- Because only the subset of data to be sent to the receiver system has to be created, a reduced message optimizes performance in the creation and communication of the message. Therefore, it's a good idea to reduce the amount of data to be distributed (with master data) whenever possible; you can apply all other data transformations to the reduced data as dictated by your requirements.

Figure 3 *ALE's Customizing Tools for Data Transformation*

Tool	Define using…	Assign using…	Comments
Reduced Message Type Defines a reference IDoc of *reducible* message type by specifying a subset of segments and fields.	• BD60 (Set Message Type to Reducible) • BD65 (Define Mandatory Fields) • BD53 (IDoc Reduction Maintenance)	• WE20 (Partner Profile) — Assign as a message type in inbound/outbound parameters of the partner profile.	• Transactions BD60, BD65, and BD53 are client-independent. • Reduced message type is for master data only.
Data Filtering in the Distribution Model Specifies filter attributes and corresponding values. A segment and its subsegments will be sent only if the IDoc contains the specified values.	• BD95 (Define Filter Object Type) • BD59 (Assign Filter Object Type to IDoc Field)	• BD64 (Maintain Distribution Model) — Create filter by adding a filter group to the message type assignment of a model view and specifying valid filter values.	• Transactions BD95 and BD59 are client-independent.

(continued on next page)

Figure 3 (continued)

Tool	Define using…	Assign using…	Comments
View A subset of segments of an IDoc type and, optionally, of its extension. Can be said to filter a subset of the IDoc's segments statically.	• WE32 (View Development) — Specify view for a given combination of IDoc type, extension, and message type.	• WE20 (Partner Profile) — Assign view in the outbound parameters of the partner profile.	• Transaction WE32 is client-independent. • Only a few outbound processing modules process view definitions (e.g., DESADV).
Segment Filter Filters all defined segments statically.	Not applicable.	• BD56 (Define Segment Filter) — Define all segments that combine message type, sender, and receiver.	Not applicable.
Conversion Rules Converts and maps a field's content.	• BD62 (Create Rules) — Define the name of a rule and relate it to a segment. • BD79 (Maintain Rules) — Define the data conversion and/or mapping.	• BD55 (Assign Rule to Message Type) — Assign rule to a combination of message type, sender, and receiver.	• Only the specified segment's data is available for conversion/mapping — e.g., in a material master IDoc, the material number is not available in the plant data segment E1MARCM; it is in segment E1MARAM.

- A reduced message type can provide a well-defined data set for change pointers, thus allowing for more precise data replication in master data interfaces.[11]

The limitation of the reduced message type is that it can only be used for the distribution of master data. There are no reducible message types for document data, such as orders or invoices. To see all of the standard SAP message types that can be used as a reference for creating a reduced message type, use transaction *BD60*.

To create a reduced message type, go to transaction *BD53* (IDoc Reduction Maintenance) and define a subset of data with reference to the latest IDoc version of the message type to be reduced (in 4.6C it would be *MATMAS04* for message type *MATMAS*). You can choose from among all the segments and fields of the reference IDoc, except those that have been defined as mandatory in transaction *BD65* (they are always in the scope of the reduced message type).

In an ALE distributed business process, only the defined segments and fields of a reduced message type will be created in the outbound process. To be able to process and post the message, the receiver system must be able to recognize the reduced message type. In other words, a reduced message type with the same name and data definition must exist on both the sender and receiver sides.

[11] For guidance on applying change pointers, see Amy Stapleton's article "Real-Time, Outbound Interfaces to Non-R/3 Systems Made Simple with Change Pointers, Message Control, and Workflow" in the Premiere Issue of this publication.

Data Filtering in the Distribution Model

Using a data filter in the distribution model, you can filter segments based on filter values. For example, you can filter a material master's plant data segments so that only plant data with the specified values (for example, *1000*) can pass the filter. This kind of filter is a *content-based filter*, which means that segments are filtered based on whether their content matches the filter value — as opposed to a *static filter*, which filters a segment regardless of its content, as long as the segment meets the filter definition (a specified combination of message type, sender partner, and receiver partner). Segment filters and views are examples of static filter types. A distribution model filter can only be applied in the outbound process.

A data filter in a distribution model is based on a *filter object*, which is a field that belongs to a data dictionary table. Each filter object should have a control table to verify that the filter value exists (if you define a filter value that cannot be validated against the control table, you will get a warning).

Many filter objects have been predefined by SAP. You can also define your own filter objects in transaction *BD95*. Custom-defined filter objects have to be in the customer namespace (i.e., custom filter objects have to start with a *Y* or *Z*). To assign a filter object to a field of an IDoc segment, you use transaction *BD59*.

In a distribution view, you can add *filter groups* to message types and BAPIs alike. A filter group contains all possible filter objects of the message type or BAPI. When you assign a value to a filter object, only those segments that possess the value specified in that field will pass through the filter. If more than one filter object is valid for one segment, the conditions are linked with a logical *AND* — and the segment will pass the filter only if all the specified filter conditions are true. Let's continue with our material master example and assume that we want to use the plant (field *WERKS*) and loading group (field *LADGR*) as filter objects; both fields belong to the plant segment (more properly called the *C* segment or segment *E1MARC*). We want to define *1000* as the filter value for *WERKS* and *0001* as the filter value for *LADGR*. Now, the filter will pass only those plant segments that contain both the value *1000* in field *WERKS* and the value *0001* in field *LADGR*.

You can add more than one filter group to a message type or BAPI that is in a view of the distribution model. If you have more than one filter group, the filter conditions of the filter groups are linked with a logical *OR* — if a filter condition in one of the filter groups is true, then the corresponding segment will be sent. For example, let's add another filter group to the one in the previous example and define *2000* as the filter value for *WERKS* and *0002* as the filter value for *LADGR* in the new filter group. Now the following segments will pass the filter: all plant segments that contain either the value *1000* in field *WERKS* and the value *0001* in field *LADGR*, or the value *2000* in field *WERKS* and value *0002* in field *LADGR*.

✓ ***Note!***

A distribution model filter will be applied to the segment and its sub-segments of a filter object. When using a filter object that is part of the root segment of the IDoc, the entire IDoc will be filtered.

Distribution model filters are best suited for content-dependent segment filtering. For example, if you want to distribute only the data of particular organizations, using filter objects based on organizational units (company code, plant, and the like) is a good way to do that.

There are no disadvantages to distribution model filters; you should feel free to filter as much as you want. The only way you would run into problems is if you were to filter out too much content, so that there was not enough data for the receiver partner.

In the data distribution example that I walk you through later, there is an example of how to use data filtering in the distribution model.

> ✓ ***Tip***
>
> *For a content-based selection of IDocs, if there is no field with content to be filtered in the root segment, you can use a filter in the distribution model in which a random field of the root segment serves as a filter object type (we will see more of this in the upcoming example). You simply choose a random field and fill it with the value defined in the filter (do this in a program exit of the outbound processing module). However, you must keep the system load in mind if you use this technique because an IDoc will always be created, even if the IDoc will not be sent. If too many IDocs are created (which occurs mainly with master data IDocs) that will only be filtered out, then you should consider: (1) using the classification as selection criteria when available in the IDoc sender program (e.g., RBDSEMAT for material master); (2) extending a copy of the IDoc sender program with appropriate selections; or (3) restricting the permanent interface (change pointer processing or the rules of workflow events) so that only the needed IDocs will be created.*

View

As mentioned earlier, a view is a kind of data filter. It defines a subset of an IDoc's segments. When applied, only the segments defined in the view will be sent. A view is different from a reduced message in that it creates a data subset out of the entire message (like a filter), whereas a reduced message is created with its reduced scope right from the beginning, when the IDoc is created.

Views can be processed in the outbound process only. You can assign a view to any outbound parameter in the partner profile (transaction *WE20*) by referencing a combination of IDoc, extension (if one exists), and message type. For instance, you can define a view for an invoice (message type *INVOIC*) that contains all segments except summary segments; you can then assign this view to all the *INVOIC* outbound parameters that do not require summary segments in their *INVOIC* messages.

The advantage of using a view is that it gives you the option to use a standardized message subset for distributed processes with various partners.

The disadvantage of views is that they are processed by only a few of the outbound processing modules, such as the modules for message types *DESADV* and *INVOIC*. Even though you can define views for all IDoc types in transaction *WE32* (View Definition) and assign them to any partner profile's outbound parameter in transaction *WE20* (Partner Profiles), there is no guarantee that the view will be processed! To date, I know of no SAP reference that lists the outbound processing modules that will process views. To find out whether a view will be processed by modules other than those for message types *DESADV* or *INVOIC*, it is best to use trial and error.

Segment Filter

A segment filter allows you to explicitly filter a segment for a given message type and partners (sender and receiver). As mentioned earlier, segment filters are static — that is, the segment is always filtered out as long as it belongs to the specified type and partners; its content is irrelevant to whether or not the segment is filtered out.

A segment filter can be used in both outbound and inbound processing. All the tools discussed up to this point can only be used in the outbound process. In most cases, you should filter on the outbound side to reduce the amount of data to be communicated. But there are circumstances in which segment filtering in the inbound process makes sense. For instance, let's say you have a standard message coming from a sender who does not want to change his message. Certain segments of this message supply information that is not needed or that might lead to processing errors. In a case like this, you can solve the problem by filtering the unwanted segments with a segment filter on the inbound process.

Figure 4 Mapping and Conversion Features of Conversion Rules

Feature	Additional features (see Figure 5)	Can be used as a template for a general rule*	Can be used to define processing rules for undefined conditions**
Transfer Sender Field Moves the specified sender field to the receiver field. The sender field can be any sender field of the segment with the same type as (or a type convertible to the type of) the receiver field.	Conversion routine, string processing, conditions, restrict value range		✓
Convert Sender Field Converts one or more sender fields of the segment according to translation tables. Multiple sender fields can be defined with an associated translation table for each sender field. ALE applies the first sender field that meets the reference value in the translation to the receiver field.	Conversion routine, string processing, conditions	✓	✓
Convert and Copy Sender Field A combination of the Convert Sender Field and Transfer Sender Field conversion rules.	Conversion routine, string processing, conditions, restrict value range	✓	✓
* See the “Use General Rule” feature defined in this table. ** For example, classify as error, set to “0,” etc.			

(continued on next page)

A segment filter is a good choice if you are sure that (1) you do not need certain segments of a message under any circumstances; and (2) you cannot apply a view. A view is the more flexible type of static filter, because you define it once and then you can apply it on an as-needed basis as often as you want. A segment filter, on the other hand, requires you to define each segment for each message type and partner combination every time you need it.

Conversion Rules

After filtering out the content that you do not require, the next logical step is to convert or transform the remaining data as needed.

In ALE, data conversion and mapping is implemented with conversion rules, with each rule related to a specified segment. In a conversion rule, all of a segment’s fields exist as receiver fields (which have to be provided with data) and as sender fields (which contain the original segment data of the message). Imagine the sender and receiver fields as the input and output of the data conversion. Defining a conversion rule means to map or transform the sender fields of the segment to the receiver fields, or to assign constants to the receiver fields.[12]

Figure 4 describes the types of conversion rules

[12] See Figure 14 on page 76 for an example of a 1:1 mapping.

Figure 4 (continued)

Feature	Additional features (see Figure 5)	Can be used as a template for a general rule*	Can be used to define processing rules for undefined conditions**
Set Constant Moves the specified constant to the receiver field. Note that character # can be used to initialize the receiver field.	Conversion routine	✓	
Set Variable Applies a predefined variable. A variable can be a centrally defined constant.	Conversion routine	✓	
Use General Rule Applies a previously defined general rule to the receiver field. To define a general rule, use one of the other rules (except Convert and Copy Sender Field) as a template.	Conversion routine	✓	✓

* See the "Use General Rule" feature defined in this table.

** For example, classify as error, set to "0," etc.

available in ALE, and **Figure 5** explains ALE's additional features for conversion rules. For more details on conversion rules, refer to SAP's Implementation Guide (IMG).

> ✓ ***Tip***
>
> *If the features supplied by ALE's conversion rules are not sufficient for your needs, you might consider using user exit KKCD0001. This user exit allows you to program data mapping and conversion within a single segment. Like the conversion rules, however, it does not provide you with access to any data in the IDoc other than the segment data with which you are working. Read SAP Note 84374, as using this exit can be a bit tricky.*

Conversion rules are the best choice for data mapping and conversions because they do not require programming. They have one serious shortcoming, however: the only input values available for conversion are in the specified segment, whereas in a user exit or Business Add-In you might have all the message data available for data transformations. The upcoming data distribution example includes a demonstration of how to create and apply a conversion rule.

Program Exits for Applying Custom Coding

If the features of the previously discussed tools are not sufficient for transforming the data in a message, you will need to utilize program exits. SAP provides program exits for specific message types as well as

Figure 5 *ALE's Additional Features for Conversion Rules*

Additional Feature	Description
Conversion Routine	May be used with all types of conversion rules. All function modules that meet the following naming convention can be used as conversion routines: CONVERSION_EXIT_cccc_INPUT where "cccc" is the name of the conversion routine. This is the same conversion routine that will be used by a data dictionary domain to convert inbound strings (e.g., MATN1 on the domain MATNR). See the note for programmers below.
String Processing	May be used with Transfer Sender Field, Convert Sender Field, and Convert and Copy Sender Field. Allows you to specify an offset and length for the string (e.g., sender field) to be transferred. The data of the sender field can be accessed as a string.
Value Range	May be used with Transfer Sender Field and also with Convert and Copy Sender Field. Allows you to restrict the value range for which a sender field will be copied.
Conditions	May be used with Transfer Sender Field, Convert Sender Field, and Convert and Copy Sender Field. Conditions in Transfer Sender Field are based on another sender field, which have to be met so that the sender field to be transferred will be copied. Conditions in Convert Sender Field are translation tables for "1:1" (e.g., "10" → "AA," "11" → "AB") and "1:Interval" (e.g., "00" will result from input value range "1000" to "2000") data conversions.

Programming Note!

You can create your own conversion routines by adhering to the naming convention

CONVERSION_EXIT_cccc_INPUT

and strictly following these function module parameters:

IMPORTING parameter → INPUT (pass by value)

EXPORTING parameter → OUTPUT (pass by value)

EXCEPTIONS → LENGTH_ERROR

Be sure to check the specific length of the string to be converted; the length of the importing and exporting parameters is not restricted to the length specified in the sender/receiver field.

Assign a newly created conversion rule to a function group of the customer namespace, and use "Z" or "Y" for the first character in the "cccc" segment of the naming convention shown above. The name of the function module itself will be in the SAP namespace.

for generic message types. In the structured approach I recommend, you use message-type-specific exits in preference to generic exits. Generic exits should be used only if there is no other way to meet the requirements. The reason for this strategy is quite simple: if you were to use a generic exit for processing a specific message, you would have to keep in mind that this exit might be used for various other messages and structure its logic accordingly. Program logic that must work properly for different kinds of messages not only reduces maintainability, but it can also be an unnecessary source of errors. In addition, if you were to use a generic exit for applying more than one message-type-specific coding, your options for programming it in a transparent way would be limited.

ALE provides two technologies you can use to add custom functionality to standard code:

- Business Add-Ins (BAdIs)
- User exits

Both of these technologies can be used for message-type-specific program exits as well as generic exits.

Business Add-Ins (BAdIs)

Business Add-Ins (BAdIs) are a comparatively new (as of Release 4.6) object-oriented technology for applying custom coding in program exits.

In a BAdI definition, you specify an interface (essentially a frame to which logic can be added), including its method definition. From the BAdI definition, adapter class methods are then generated that provide the basis for the implementation of custom functions. The addition of custom logic is an implementation of the method and is therefore called a "BAdI implementation." Two of a BAdI's features make this technology preferable to a user exit:

- An adapter class method can be enabled for multiple active implementations. This means that multiple active BAdI implementations will be processed in order (one by one, in the order of their implementation). This feature allows adding numerous custom functions to one program exit by programming each customer function in exactly one BAdI implementation.

- Filter-dependency can be utilized in a BAdI implementation so that its methods will be processed only if the application that calls the BAdI delivers the filter value specified in the implementation.

I think that the BAdI technology is quite elegant, and because SAP has announced that it plans to replace user exits with BAdIs as the preferred interface technology, it is well worth taking some time to learn about it.

In R/3 Release 4.6C, there are a bit more than 300 BAdIs available. In R/3 Enterprise, there are almost 1,500 BAdIs already defined in the standard system. In ALE, however, only a few BAdIs are available. Definition *IDOC_CREATION_CHECK* (as of Release 4.6C) is an example of a generic ALE BAdI (it checks the creation of an IDoc). I will show you an example of a message-type-specific BAdI later on.[13]

User Exits

While user-exit technology is well understood by SAP consultants and ABAP programmers, there are some ALE-specific features of user exits that require a closer look.

In ALE, message-type-specific user exits can be categorized as exits for basic IDoc types and exits for IDoc extensions. Exits for basic types will always be called by the application. Exits for IDoc extensions are called only if the message contains an IDoc extension.

To get an idea of which message-type-specific user exits are available in ALE, search the SAP Service Marketplace (**http://service.sap.com**) for the document "List IDocs with User Exits." This document is not new (it became available with Release 4.5), but as far as I know it is the best starting point for learning about message-type-specific user exits.

You will notice that the IDocs of many message types can't be reached by message-type-specific user exits without an extension. If you have to transform the data of a basic IDoc type that does not have a message-type-specific program exit, then you've almost reached a dead-end. As a last resort, you can use the generic user exit *ALE00001*. This exit is intended to add functionality to ALE's version control, but you can adapt it to change the message content of almost any IDoc. For details, see the sidebar

[13] For more information on BAdIs, you can refer to the online documentation or to Karl Kessler's article "Extending and Modifying the SAP Standard with Business Add-Ins and the New Modification Assistant" in the Premiere Issue of this publication.

When No Message-Type-Specific Program Exit Can Be Found

User exit ALE00001 is intended to add functionality to ALE's version control, but it can be adapted to change the message content of almost any IDoc.*

According to the program exit documentation in transaction CMOD, user exit ALE00001 is called only when the IDoc type *and* the CIM type are different from the type expected by the ALE layer; actually, the exit gets called when the IDoc type *or* CIM type is different from the type expected by the ALE layer.

To find out what IDoc type the ALE layer expects, you can single-test the function module EDI_GET_LAST_IDOCTYP (or EDI_GET_LAST_CIMTYP for the expected extension type). Call transaction SE37, enter the name of the function module, and press the single-test button (F8). Enter the actual IDoc type in input parameter IDOCTYP_START and press execute () or F8. The output parameter IDOCTYP_LAST will show the IDoc type expected by the ALE layer. If the IDoc types of IDOCTYP_START and IDOCTYP_LAST are different from each other, then you know that user exit ALE00001 will be called. For example, when you enter MATMAS03 as an input parameter in a 4.6C system, you will get MATMAS04 as an output parameter value. This means that if the IDoc comes with MATMAS03, the user exit will be called. If you need the user exit to be called, but the IDoc or CIM types match what the ALE layer expects, change the IDoc type in the outbound parameters of the partner profile to a lower version than the one shown in IDOCTYP_LAST. Then, exit ALE00001 will be called in both the outbound and inbound processes.

If you cannot change the IDoc type to a lower version because you need the latest IDoc version available in the message, you might try the following trick. Take the row of the IDoc type you want to use in table EDBAS as a starting point for a copy. Copy it and then change the name of the IDoc type to a higher version (e.g., IDoc type MATMAS04 to MATMAS99) and enter the original IDoc type in field *Predecessor IDoc*. Now when you single-test the function module EDI_GET_LAST_IDOCTYP with MATMAS04 as the import parameter value for IDOCTYP_START, you will get MATMAS99 as the export parameter value in IDOCTYP_LAST. This customizing causes the user exit to be called, because MATMAS04 is different from (doesn't match) MATMAS99.

This trick only works in the ALE layer of the inbound process. In the outbound process, wherever the EDBAS IDoc type does not exist, control tables other than EDBAS will be checked for the new IDoc type. In these cases, you cannot use the user exit in the outbound process.

✓ ***Caveat!***

Use ALE00001 for data manipulation <u>only</u> if there is no other standard way to transform message data. The conditions that have to be met in order for the user exit to be called might change after release upgrades, which means that it might not be called. Another reason to avoid using this method is that it can negatively affect the maintainability of the ALE interface, especially if it is used for numerous message types or functions.

* As of SAP Web Application Server 6.20 (e.g., R/3 Enterprise), use BAdI IDOC_DATA_MAPPER instead of user exit ALE00001 as a generic program exit for data manipulations. In addition to the technological advantages of BAdIs compared to user exits, you can log the manipulations in a status record.

above ("When No Message-Type-Specific Program Exit Can Be Found").

Let's see now how to use ALE's data transformation features by walking through an example.

Example — Controlling and Transforming a Message

To show you how to set up data transformation in an ALE distributed process, I'll take you step-by-step through the procedure for controlling and transforming one particular IDoc message in a distributed business process.

The Task

Imagine that we want to maintain a material master centrally on an SAP R/3 system.

We need to replicate the material master data from this application to another R/3 system, which is customized differently than the original R/3 system. We want to send only material masters with the string *DO-IT* (use uppercase) in the English material description to the receiver system. The base unit of measure will be copied if it is *ST* (item); otherwise it will be converted to *KG* (kilogram).

System Requirements

Our system requirements are two SAP clients on SAP R/3 Releases 4.6C or later. For testing purposes, it's best if these clients have identical customizing (otherwise you would probably have to mess around with getting the material master data posted on the inbound side). In the examples used here, the logical systems are called *LOGSYS0100* for the sender system and *LOGSYS0200* for the receiver system.

If you have only one client available for testing, you will find a short description of how to set up an ALE scenario on one client at **www.SAPpro.com**. Keep in mind, however, that when the sender and receiver system are on the same client, the transformed data will be written to the original data source (which is the material master).

Preparation

Our example is based on the SAP Library's ALE Quick Start, which is a hands-on exercise for ALE distribution of material master data (see the "Documentation Resources" download available at **www.SAPpro.com**).

To prepare, do the following:

1. Set up a distributed business process between two clients with a material master message type. If you are not familiar with ALE, I recommend going through the online help's example of setting up a distributed business process between two clients with a material master message type.

2. Check the *MATMAS* inbound parameters in the partner profiles in transaction *WE20* for the processing code. The parameter has to be *MATM*.

3. If you have an SAP R/3 4.6x system, adapt partner profiles in transaction *WE20* for the *MATMAS* outbound parameters. Change *IDoc Basic type* to *MATMAS03*.

> ✓ **Tip**
>
> *To get more detailed help when going through the customizing transactions of ALE, install the SAP Library (or get it installed by your system administrator) for your test system, and navigate to Application Help (F1 → F4).*

4. Test the material master distribution by using transaction *BD10* to send the material master. Specify the material number and the receiver system and press execute ( ) or *F8*.

> ✓ ***Tip***
>
> *If you get posting errors when testing material master distribution, you can analyze them in transaction SLG1. You can also navigate to the status record in the IDoc list (transaction WE02) by double-clicking on the erroneous IDoc and opening the node of the red, highlighted status record. In the toolbar of the record's detail screen, there is a button named Application Log. Click it (or press Shift+F8) to get the posting log.*

If you have problems setting up the ALE environment as described in the Quick Start reference, ask your team's ALE consultant to do it for you.

Before going any further, get acquainted with the IDoc structure of *MATMAS03* by browsing through the tested IDoc with transaction *WE02*.

Implementation of the Example

Adhering to the structured approach laid out previously, the first tool we will use is a distribution model filter: we want to filter out any material masters that do not contain the string *DO-IT* in the English material description. This information resides in segment *E1MAKTM* (the material master short text segment), in the field *MAKTX* (the material master description). If the value of field *MAKTX* is *DO-IT*, then the message should be passed to the receiver partner; otherwise, the entire message should be filtered out.

Filtering out the entire message means we will have to utilize a filter object of the material master root segment (*E1MARAM*) because we want all child segments of the root segment to be filtered out as well. Because segment *E1MAKTM* is not a root segment, we will have to move the content to be filtered into a random field of segment *E1MARAM*. None of the filtering or data conversion tools can move this content for us, so we will need to utilize a program exit to move the content to the root segment *before* the ALE layer applies the distribution model filter. According to the structured approach that I have recommended, the first type of program exit we should look for is a message-type-specific BAdI. Fortunately, there is a BAdI for outbound material masters that we can use.

For translating the unit of measure in the IDocs that pass this filter, we will use a conversion rule, which is well suited to translation tasks. The ALE layer calls conversion rules after the distribution model filter is applied.

The order in which we will implement our data transformation is as follows:

1. We will implement a BAdI that copies the string *DO-IT* (if it exists) into field *BISMT* (old material number) of segment *E1MARAM*.

2. We will then define a filter value (*DO-IT*) for field *BISMT* in the distribution model.

3. Finally, we will define a conversion rule to perform the unit conversions.

Step 1: Implement the BAdI

Follow these steps:

1. Call transaction *SE19* (Business Add-Ins Implementation).

2. Insert a name for the implementation (e.g., *Z_MATMAS_FILTER_PREP*) and press *Create* (or *F5*).

3. Choose BAdI definition *BADI_MATMAS_ALE_CR* (Change Data in MATMAS IDoc When Generating an IDoc) in the pop-up that appears when creating the implementation.

Figure 6 BAdI Implementation Attributes

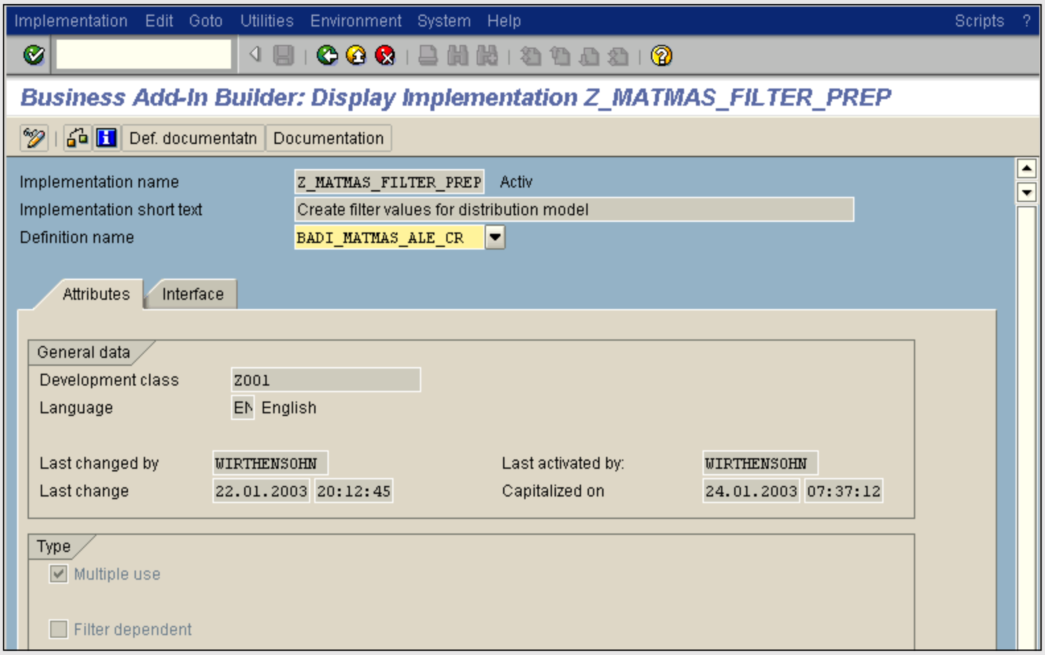
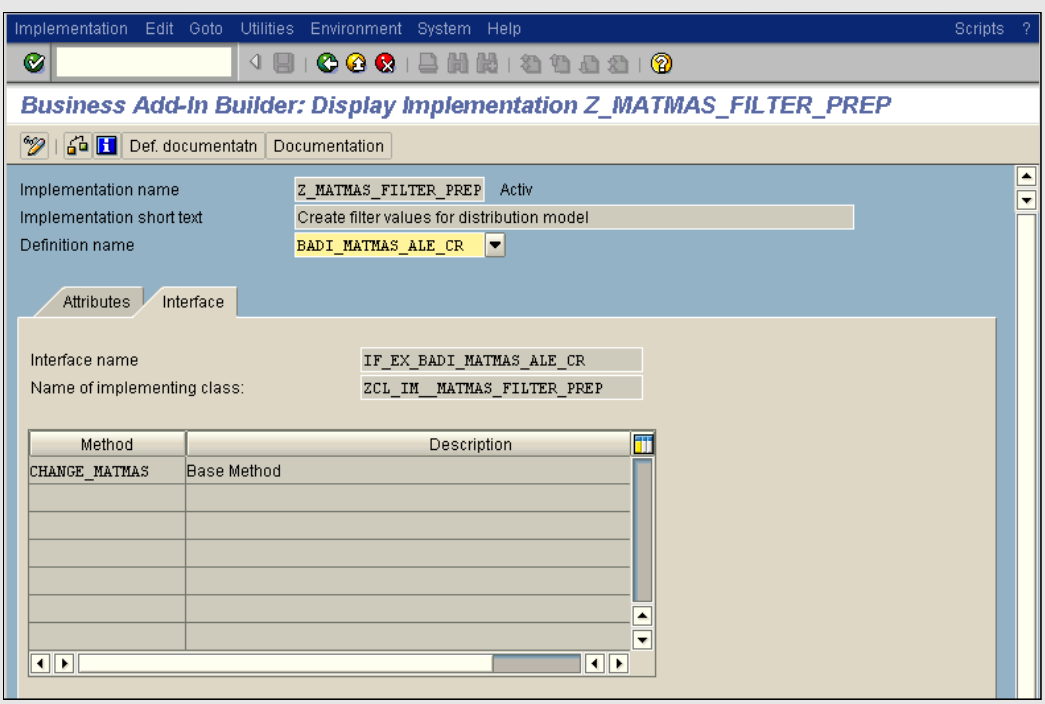


Figure 7 BAdI Implementation Interface



4. Add a short description (e.g., *Create filter values for distribution model*) and save it. You will get a result like the one shown in **Figure 6**.

   Note the BAdI's attribute type; the definition is of type *Multiple use*, which means that you can have more than one implementation of that BAdI.

5. Choose the *Interface* tab and you get a screen similar to the one shown in **Figure 7**. Look at the name of the implementing class, which is derived from the BAdI implementation name. In our example, the name of the implementing class would be *ZCL_IM__MATMAS_FILTER_PREP*.

6. Double-click on the *CHANGE_MATMAS* method, add the program logic shown in **Listing 1** to the method, and adapt the coding if necessary (e.g., adapt the constant for field *i_receiver* to your logical receiver system).

Listing 1: BAdI Implementation Sample Coding

```
method if_ex_badi_matmas_ale_cr~change_matmas.

* Data declaration
  data: i_receiver   type edi_rcvprn value 'LOGSYS0200',
        i_idoctype   type edi_idoctp value 'MATMAS03',
        i_language   type spras_iso  value 'EN',
        i_doit(10)   type c          value 'DO-IT',
        i_dont(10)   type c          value 'DO NOT',
        i_true       type c          value 'X',
        i_processing type c,
        i_passidoc   type c.

  data: is_idoc_data type edidd,
        is_e1maram   type e1maram,
        is_e1maktm   type e1maktm.

* Determine processing restriction
  if ( f_idoc_header-rcvprn = i_receiver ) and
     ( f_idoc_header-idoctp = i_idoctype ).

    i_processing = i_true.

  endif.

* Processing logic
  if i_processing = i_true.

*   Find out if the required string is there

    loop at t_idoc_data into is_idoc_data where segnam = 'E1MAKTM'.

      is_e1maktm = is_idoc_data-sdata.
      if ( is_e1maktm-spras_iso = i_language ) and
```

(continued on next page)

(continued from previous page)

```
        ( is_e1maktm-maktx cs i_doit ).

        i_passidoc = i_true.

      endif.

    endloop.

*   Change field BISMT according to previous findings

    loop at t_idoc_data into is_idoc_data where segnam = 'E1MARAM'.

    is_e1maram = is_idoc_data-sdata.

      if i_passidoc = i_true.
        is_e1maram-bismt = i_doit.
      else.
        is_e1maram-bismt = i_dont.
      endif.

      is_idoc_data-sdata = is_e1maram.
      modify t_idoc_data from is_idoc_data.

    endloop.

  endif.

endmethod.
```

7. Click the *Signature* button (*Ctrl+Shift+F6*) to get the method's signature displayed, as shown in **Figure 8**.

8. Save and activate your coding.

> ✓ ***Note!***
>
> *Take care that only the messages you want to have processed will be processed by the BAdI implementation. It's best to restrict the processing logic to a specified receiver partner and IDoc type by defining conditions for corresponding control record fields.*

9. Activate the BAdI implementation (*Ctrl+Shift+F4*) and test the data distribution.

   The best way to test your BAdI is with two sample material masters — one that contains the string *DO-IT* (uppercase!) in the English short text, and one that does not contain the string. Check the results by viewing the created IDocs (transaction *WE02*) and by looking at the material masters (transaction *MM03*) in the target system.

Step 2: Define the Filter in the Distribution Model

Follow these steps:

Figure 8 *BAdI Implementation Method Signature*

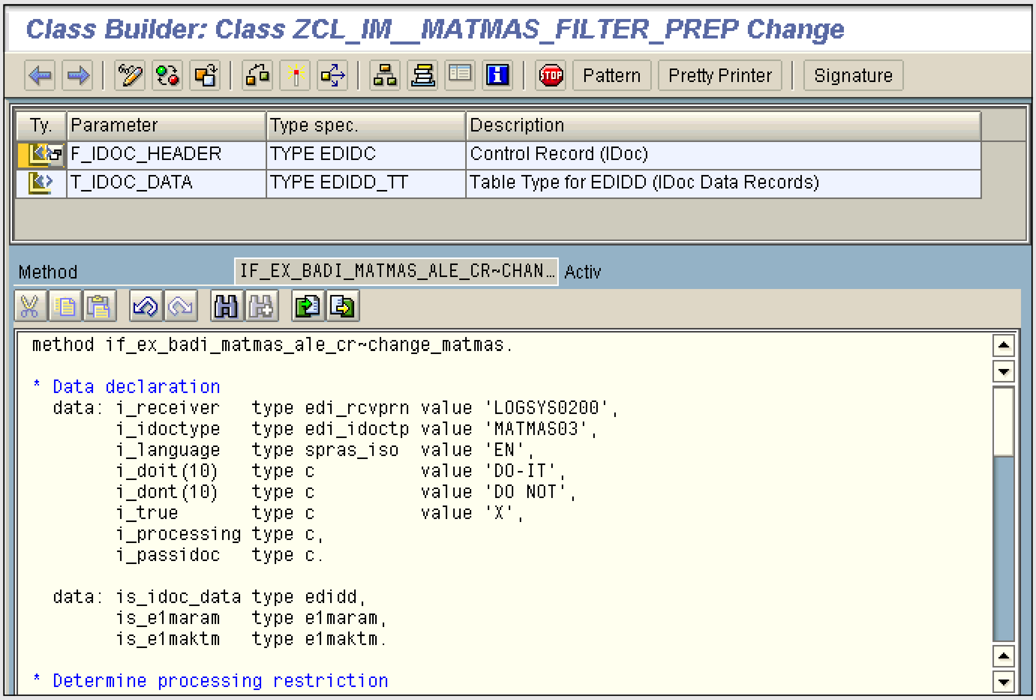
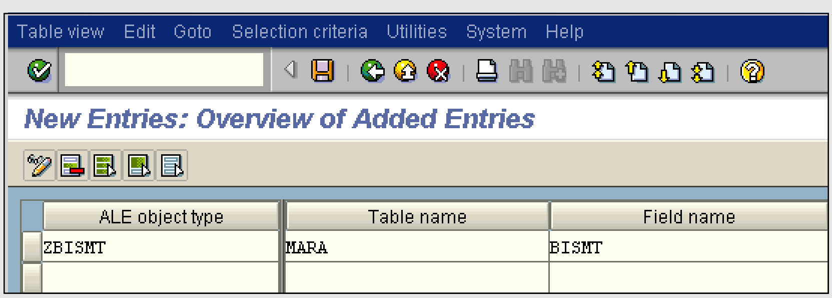


Figure 9 *Object Type Definition*



1. Call transaction *BD95* (View maintenance for table TBD11: ALE Object Type).

2. Create a new entry for object type *ZBISMT*; assign table name *MARA* and field name *BISMT* to it. *BISMT* in table *MARA* does not have a check table, so you can ignore the warning. The result will look like **Figure 9**.

3. Save your entries.

Figure 10 Object Type Assignment

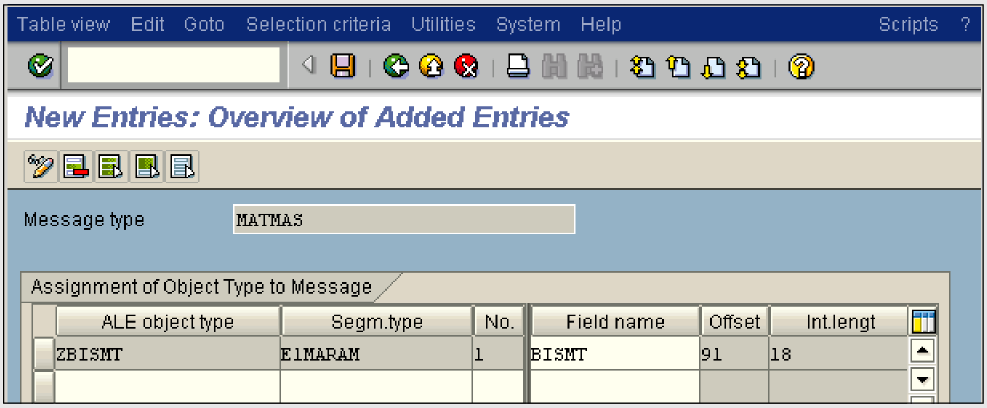
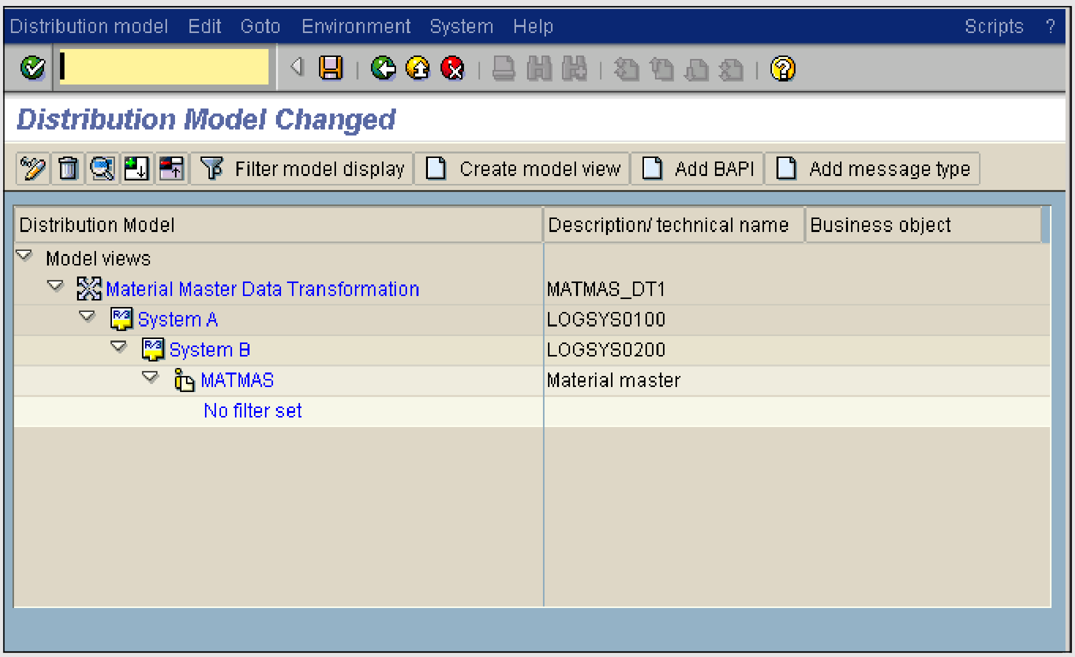


Figure 11 Distribution Model



4. Call transaction *BD59* (View maintenance for view V_TBD10: Assignment of Object Type to message).

5. In the pop-up that appears, specify *MATMAS* as the message type; create a new entry for object type *ZBISMT*, segment type *E1MARAM*, and sequence number *1*; and choose segment field *BISMT*. When confirming your entries, the offset and internal length of field *BISMT* in

segment *E1MARAM* will be displayed as shown in **Figure 10**.

6. Save your settings.

7. Call transaction *BD64* (Maintain Distribution Model). Switch to change mode (*F9*) and expand the node of your material master model view. The last element of the node — the one after message type *MATMAS* — says *No filter set* (see **Figure 11**). Double-click on this element and the pop-up for filter definition will appear.

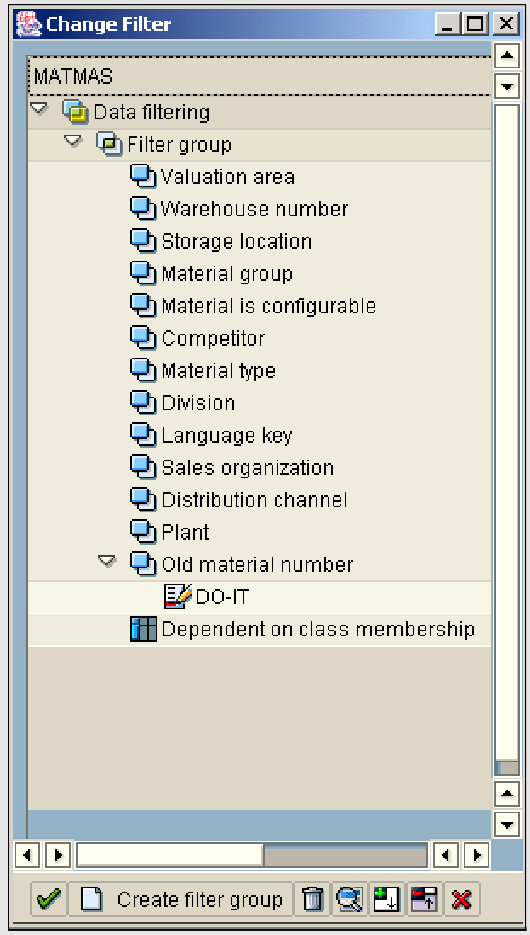
8. Press the *Create Filter Group* button (*F5*) and open the node *Data filtering*, including its sub-element *Filter group*. Near the end of the filter group's object types you will find *Old material number*. Double-click on *Old material number*, and you will get a pop-up showing a list of values for that object type.

9. Press the button *Insert line* and specify the filter value *DO-IT*. Press continue ( ) to return to the filter group editor. *Old material number* now appears as a node with sub-elements. When you expand it, you will see the newly defined filter value, as shown in **Figure 12**.

10. Return to the distribution model maintenance screen (Figure 11) and save your changes.

11. Test the data distribution. When you send the two sample material masters, note the pop-ups that appear when sending the material master: you will see that two master IDocs have been created, but only one communication IDoc. The two master IDocs have been created from the two selected material masters, but one of these material masters has been filtered out, so that only the other material master will be transmitted (as a communication IDoc) to the receiver partner. Only the communication IDoc will be stored in the IDoc database tables. This means that you can only investigate the communication IDoc in the monitoring tools (e.g., *WE02*). The master IDoc that has been filtered out hasn't been stored in the database. But this, of course, is just what we wanted — the ability to filter out an entire IDoc.

Figure 12 Distribution Model with Filter



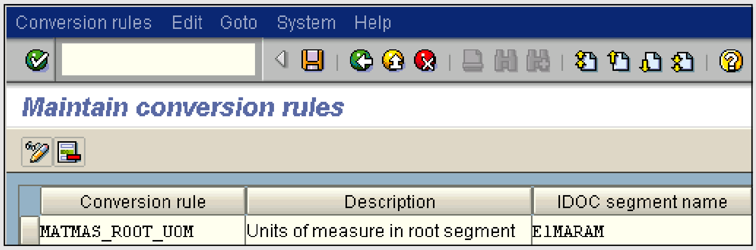
Step 3: Define and Apply the Conversion Rule

Follow these steps:

1. Call transaction *BD62* (Create Conversion Rules) and switch to change mode (*Ctrl+F1*).

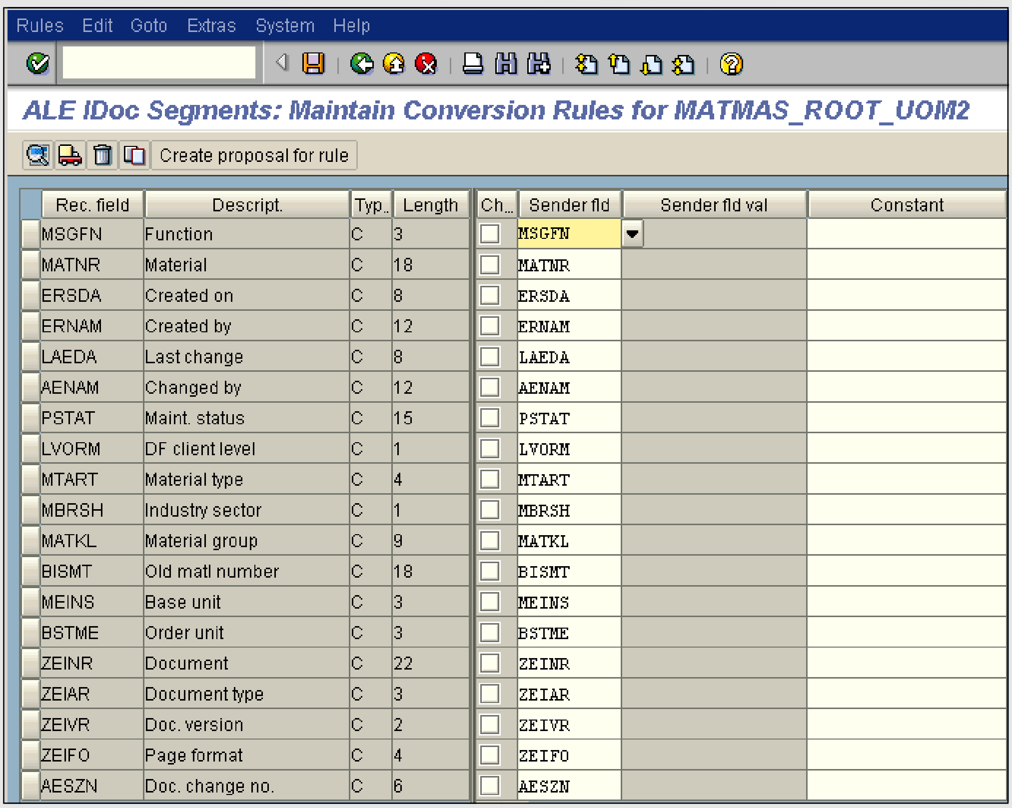
2. Enter a new rule, *MATMAS_ROOT_UOM*, add a short description (e.g., *Units of measure in root segment*), and assign segment *E1MARAM* to it.

Figure 13 *Conversion Rule Creation*



Conversion rule	Description	IDOC segment name
MATMAS_ROOT_UOM	Units of measure in root segment	E1MARAM

Figure 14 *Conversion Rule Overview After "Create Proposal for Rule"*



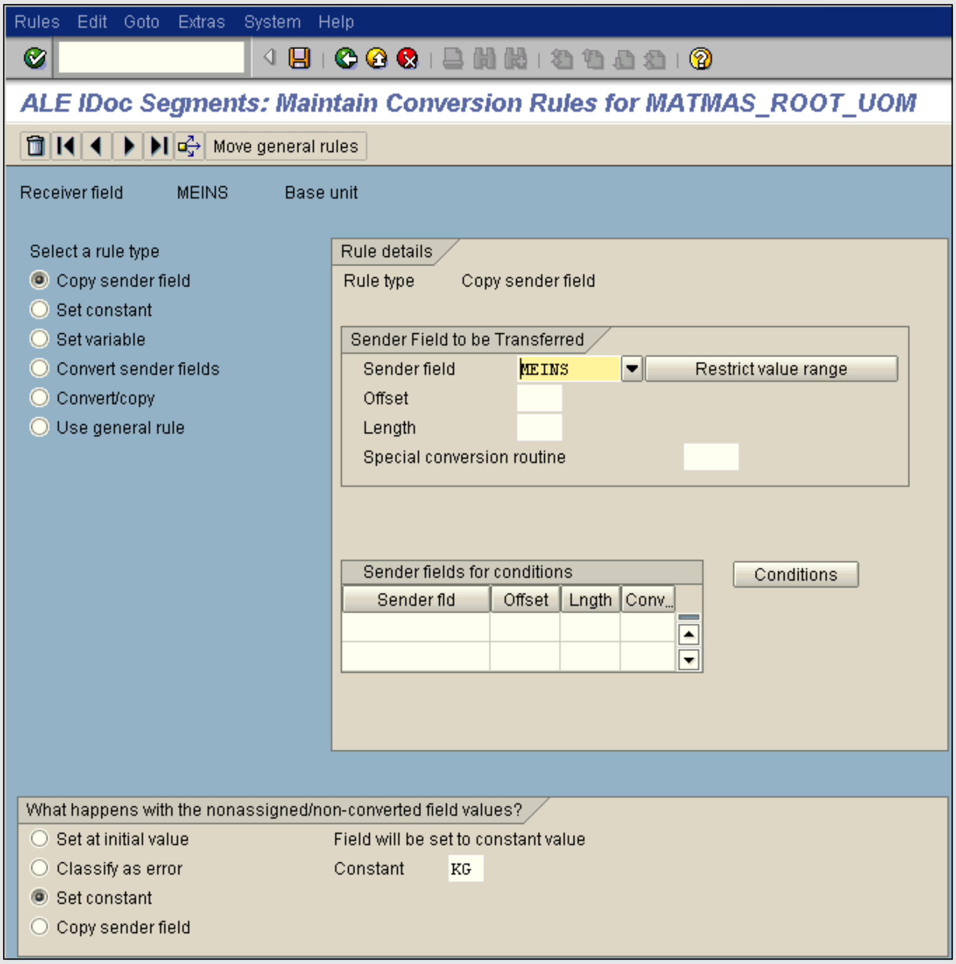
Rec. field	Descript.	Typ..	Length	Ch..	Sender fld	Sender fld val	Constant
MSGFN	Function	C	3		MSGFN		
MATNR	Material	C	18		MATNR		
ERSDA	Created on	C	8		ERSDA		
ERNAM	Created by	C	12		ERNAM		
LAEDA	Last change	C	8		LAEDA		
AENAM	Changed by	C	12		AENAM		
PSTAT	Maint. status	C	15		PSTAT		
LVORM	DF client level	C	1		LVORM		
MTART	Material type	C	4		MTART		
MBRSH	Industry sector	C	1		MBRSH		
MATKL	Material group	C	9		MATKL		
BISMT	Old matl number	C	18		BISMT		
MEINS	Base unit	C	3		MEINS		
BSTME	Order unit	C	3		BSTME		
ZEINR	Document	C	22		ZEINR		
ZEIAR	Document type	C	3		ZEIAR		
ZEIVR	Doc. version	C	2		ZEIVR		
ZEIFO	Page format	C	4		ZEIFO		
AESZN	Doc. change no.	C	6		AESZN		

3. Save the rule. The result should look like what you see in **Figure 13**.

4. Call transaction *BD79* (Maintain Conversion Rules), enter the newly created conversion rule, and press the *Maintain* button (*F6*).

5. Now we've reached the actual ALE conversion

Figure 15 *Conversion Rule Detail for Copy Sender Field*



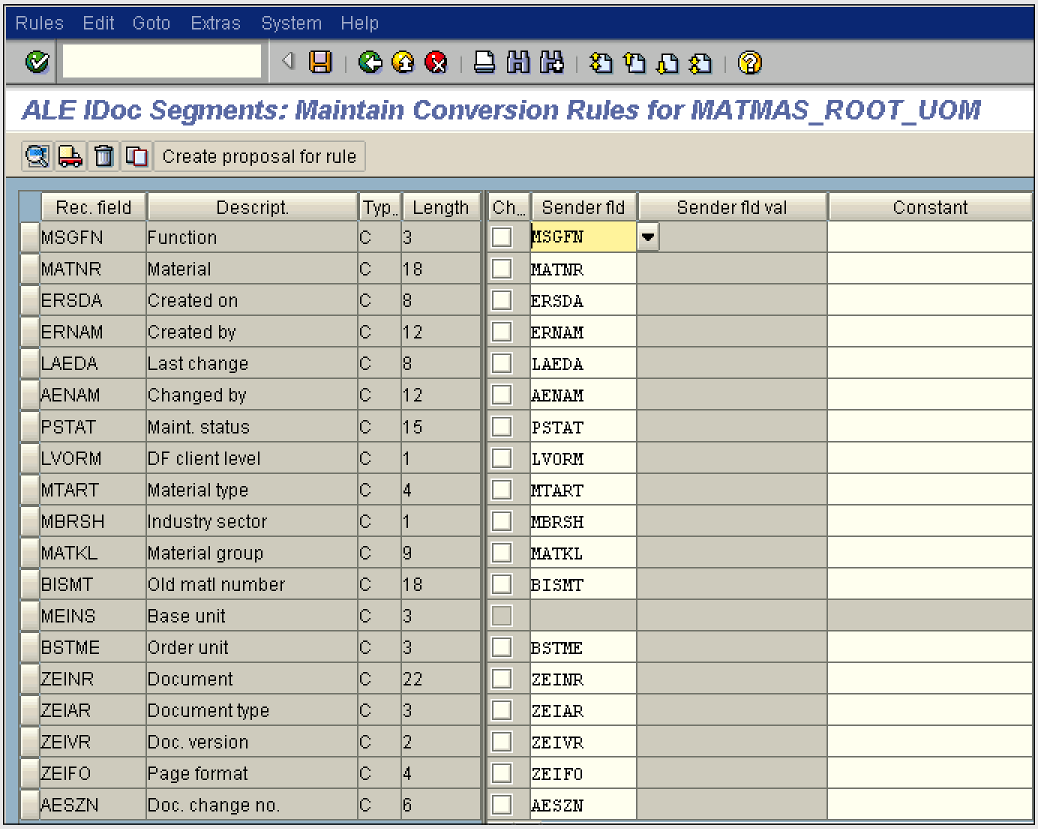
and mapping tool. On the left side, you'll see all the segment fields as receiver fields, which have to be provided with content. Press the *Create proposal for rule* button (*F5*) and you will get a 1:1 data mapping like the one shown in **Figure 14**, in which each receiver field gets its input from the corresponding sender field.

6. We want to copy the base unit (field *MEINS*) from the sender field if it is *ST*, and otherwise convert it to *KG*. So, select the row of receiver field *MEINS* and press the choose button ( ) or *F2*. Press the button *Restrict value range* in the field group *Sender Field to be Transferred*. Here you can specify the value range for the sender fields to be copied to the receiver field.

7. Enter *PCE* (the ISO code for *ST*) at the column's lower and upper limits, press *Enter*, and return (*F3*) to the detail menu of receiver field *MEINS*.

8. Further down on that screen, there is a field group called *What happens with the nonassigned/non-converted field values?* Select radio button *Set constant* and enter *KG* as the constant value. The screen should now look similar to what you see in **Figure 15**.

Figure 16 *Conversion Rule Overview After Specifying Details*



Rec. field	Descript.	Typ..	Length	Ch..	Sender fld	Sender fld val	Constant
MSGFN	Function	C	3		MSGFN		
MATNR	Material	C	18		MATNR		
ERSDA	Created on	C	8		ERSDA		
ERNAM	Created by	C	12		ERNAM		
LAEDA	Last change	C	8		LAEDA		
AENAM	Changed by	C	12		AENAM		
PSTAT	Maint. status	C	15		PSTAT		
LVORM	DF client level	C	1		LVORM		
MTART	Material type	C	4		MTART		
MBRSH	Industry sector	C	1		MBRSH		
MATKL	Material group	C	9		MATKL		
BISMT	Old matl number	C	18		BISMT		
MEINS	Base unit	C	3				
BSTME	Order unit	C	3		BSTME		
ZEINR	Document	C	22		ZEINR		
ZEIAR	Document type	C	3		ZEIAR		
ZEIVR	Doc. version	C	2		ZEIVR		
ZEIFO	Page format	C	4		ZEIFO		
AESZN	Doc. change no.	C	6		AESZN		

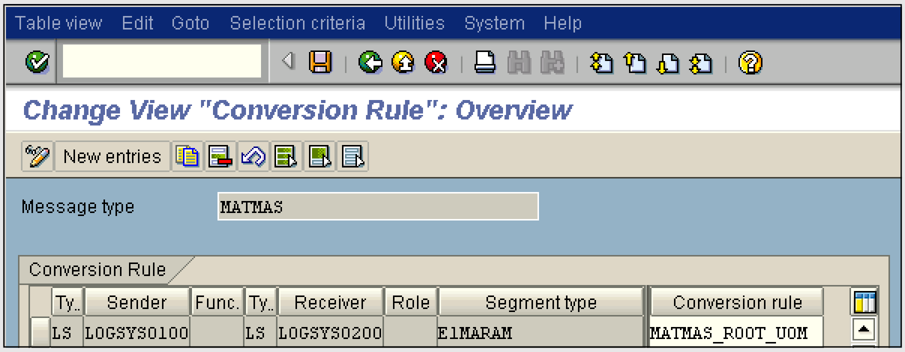
✓ ***Note!***

Internally, ALE interfaces use ISO units. There are two implications for your implementation:

- *The units to be transferred must always have an assigned ISO unit. Check units in transaction CUNI (Check Units of Measure) by navigating to Units of Measurement and selecting the details of the unit you are interested in. There you will find field "ISO code" in the field group ALE/EDI. This field must be filled with a value that is a unit defined by ISO.*
- *When applying data transformation in the conversion rules, refer to the ISO code for unit comparisons.*

9. Save the rule, navigate back to the conversion rule maintenance screen, save again, and the overview screen will look like **Figure 16**. No automatic transport will be created. You can create a transport manually with the transport button (🚚) or *F8*.

Figure 17 *Conversion Rule Assignment*



✓ ***Note!***

In an implementation project, you have to define all possible partner combinations for which the conversion rule will be valid in the development system; you then have to transport these rule assignments for the different partner combinations (of the quality assurance and production systems) throughout your system landscape. For security reasons, it is important to have unique logical system names for each system in a system landscape (e.g., development, quality assurance, and production), so you will have different partners on the QA system than on the PRD system.

10. The conversion is now defined, but the system still does not know when to apply it. We tell it with transaction *BD55* (Assign Rule to Message Type). Call this transaction, enter *MATMAS* as the message type, and confirm.

11. Press *New Entries* (*F5*). Specify the sender and receiver partners, the segment name *E1MARAM*, and the conversion rule *MATMAS_ROOT_UOM*. The type of our partners is *LS* (logical system). You do not have to specify fields *Func.* and *Role*.

12. Save what you've done. The results of your actions should be similar to what you see in **Figure 17**.

13. Test the data conversion:

    - In the sample material master that has *DO-IT* in the short text, maintain the base unit as *ST*. Save the material master and send it to the receiver system. The base unit of the sent material master should remain the same.

    - Next, change the base unit to something other than *ST* (e.g., *PAC*). Save the material master and send it again.

    - Finally, check the result in the IDoc or in the material master of the receiver system. You should see that the base unit has been changed to *KG*.

> ✓ ***Tip***
>
> *To gain a good understanding of what you can and cannot do with conversion rules, read the section on conversion rules in the IMG (Implementation Guide) very carefully, and test all the features supplied.*

Conclusion

As you have seen, SAP standard ALE offers a wide range of tools to perform data conversions. Knowing the features of the tools and adhering to the implementation approach set forth in this article will help you to plan and build more cost-effective, more easily maintainable ALE interfaces. Remember:

- ☑ Filter out what you do not need. Always pay attention to the amount of data to be distributed. To avoid sending extraneous data, filter out all unnecessary data in the outbound process.
- ☑ Utilize customizing tools before applying custom coding.
- ☑ Where possible, utilize application-specific program exits before utilizing generic program exits.
- ☑ Employ a Business Add-In, if one exists, in preference to a user exit.

When you're ready to get started in your own environment, I recommend using the tables and charts in this article as guidelines for planning and implementing your own ALE interfaces.

Arthur Wirthensohn is a senior consultant at EDS Switzerland and a member of EDS's international Technical Leadership Network. He has worked both as a project manager and product manager in the ERP and IT integration business for many years, mainly in the retail, consumer products, manufacturing, and trade industries. For the past two years, Arthur was the product manager responsible for Application Services based on SAP systems. Currently, he works as project manager and is the technical lead of the Enterprise Application Integration (EAI) department, which delivers SAP-related services, such as ALE/EDI, SAP Business Workflow, Web Application Server, Data Migration, and ABAP Programming Services. Arthur can be reached at arthur.wirthensohn@eds.com.

