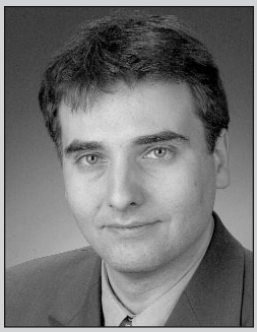
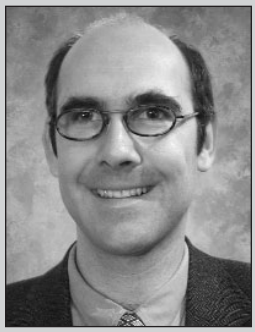
Improve the Efficiency of Your SAP Records Management Implementation with Automated Record Updates

Joachim Becker and Ulrich Spinola



Joachim Becker, Product Manager, SAP Records Management, SAP AG



Ulrich Spinola, Development Project Lead, SAP AG

(complete bios appear on page 124)

As digital documents have taken the place of paper documents, so too have far-flung data stores taken the place of myriad file cabinets, folders, and other paper organizers. With its sheer volume and possible formats and locations, however, what once was heralded as a way to simplify and consolidate your information has become as difficult to manage as the system it replaced, preventing companies from achieving the highest possible ROI from their document management investment. SAP Records Management, a standard component of SAP Web Application Server (SAP Web AS) 6.20 and higher (though licensed separately), relieves this burden. Using a new, easily extensible framework called the Service Provider (SP) Framework,[1] SAP Records Management provides users with access to all of their enterprise documents, transactions, workflows, and data, regardless of their module or vendor, via a single view — a record.[2]

SAP data by nature is constantly changing, however, which means the record containing your data needs to be updated to reflect these changes. This article shows you how to automate record updates to free you from having to update them manually, to enable you to take full advantage of your SAP Records Management implementation for real time and cost savings. And even if you are not currently using SAP Records Management, the techniques presented in this article are a useful addition to your application development toolbox.

---

[1] Developed as a series of ABAP classes, the SP Framework provides a generic set of interfaces (i.e., service providers) that applications such as SAP Records Management can use to connect to, manipulate, and integrate data and documents in SAP and non-SAP systems. You can use standard SAP-supplied service providers or you can create your own.

[2] A record can represent any type of object — such as an employee, a customer, or a customer complaint case — with which you want to associate transactions and digital documents. Records provide an organized, comprehensive view of all data associated with that object (e.g., orders, faxes, and emails across SAP and non-SAP systems).

SAP Records Management Tools and Terminology

This article builds on the concepts and examples introduced in the article "Consolidate and Integrate All of Your SAP and Non-SAP Documents, Transactions, Workflows, and Data with SAP Records Management" (*SAP Professional Journal*, January/February 2005). While we recommend that you read that article to gain a solid understanding of SAP Records Management and a detailed background on the example customer record used to illustrate the steps in this article, we provide a brief overview here. Additional information is also available at http://service.sap.com/recordsmanagement.

Key SAP Records Management Tools

The **Records Organizer** acts as a central desktop from where you can search, edit, and create records, documents, business objects, etc. All objects that have been made available for SAP Records Management can be accessed from here. The Records Organizer automatically launches the Records Browser (for viewing and editing records) and Records Modeler (for defining the structure of a record) as needed.

The **Records Browser** is used to display and edit digital records. From here, users can maintain record metadata (attributes) and record content (structured list of links to objects). You can add content, delete and change content (if you have the authorizations to do so), create new documents, etc. Figure 1 in the article shows an example record displayed in the Records Browser.

The **Records Modeler** is used by the project team that models business processes and configures the system to define record models, which are templates for the structure and content of records based on those models. Figure 2 in the article shows the model defined for the example record shown in Figure 1.

Key SAP Records Management Terminology

A **record** is an electronic "container" that groups links to digital documents, business objects, and transactional data relevant to a particular employee, customer, customer complaint case, etc. A record can also contain links to other records for organizational purposes.

Records consist of a hierarchy of **nodes**: *structure nodes* serve as headers to help organize the content

To demonstrate the steps required to enable this automation, this article builds on an example customer record created in a previous *SAP Professional Journal* article that introduced you to SAP Records Management.[3] The January/February 2005 article reviewed the key components of the SAP Records Management interface — the Records Organizer, which acts as a central desktop for records administration, and the Records Browser and Records Modeler administration tools for viewing and editing records and defining the structure of a record, respectively. The article showed you how to use these tools to set up a simple example customer record and to access and manage records.

[3] "Consolidate and Integrate All of Your SAP and Non-SAP Documents, Transactions, Workflows, and Data with SAP Records Management" (*SAP Professional Journal*, January/February 2005).

of the record and *model nodes* serve as placeholders for links to documents and data.* New records are initialized with the set of defined placeholder model nodes (see Figure 1 in the article), to which **objects** are assigned (see Figure 2 in the article). An object can be any type of digital document, business object, data, or even transaction code or workflow item, that you want to include in a record. In the context of SAP Records Management, we use the term **elements** for all objects included in a digital record.

When designing a record, each element must be described as having a specific **element type**, such as an SAP R/3 invoice object, a document (either plain text or a type of Microsoft Office document), a URL, etc., which maps the element to an appropriate **service provider**.** Service providers handle the unique details required to access the actual information objects in the backend data sources and store important type-specific attributes (e.g., where logical system elements of that type are stored, where the element should be placed in the record, etc.). Service providers also give SAP Records Management a common API with which to create, change, and/or retrieve heterogeneous SAP and non-SAP objects. All the service providers and element types available for use with SAP Records Management are maintained in the underlying **Registry** tool, where you can also define **records management systems** to logically organize records into separate areas for access control.

The **record model** on which a record is based describes the default organizational structure of the record (i.e., the structure and model nodes it includes by default) as well as the **cardinality** requirements of each node (i.e., where and how often an object can be added to a record). For example, the record model can require that at least one resume be attached to a new employee record for the record to be considered technically complete, or it can specify that more than one copy of the resume can be attached to the resume node.

---

* There is a third type, *instance nodes*, which are used to persist a certain object in all records based on a particular record model, but these are infrequently used.

** SAP supplies standard service providers or you can develop your own to integrate data or workflows from third-party systems. Element types give the service provider details it needs to access information objects, such as the name and target system of the Business Object Repository object that the service provider for business objects calls to access customer master data. The DOCUMENT_CLASS element type connection parameter for elements managed by the Knowledge Provider (records, record models, documents, document templates, and notes) specifies the content model used by the element. The content model defines the attributes bound to that element. SAP defines some standard content models as part of the Knowledge Provider, and you can also define your own using the Knowledge Provider's Document Modeling Workbench tool. For more on the Knowledge Provider, see the sidebar on pages 104-106.

✓ *Note!*

To understand and apply this article, you'll need to have at least a general understanding of the purpose and building blocks of SAP Records Management. Reading the January/February 2005 article is recommended, but not strictly required, and we try to summarize prerequisite concepts as we go along. For a recap of the key tools and terms you need to know, see the sidebar above.

Figure 1 An Example Record Model

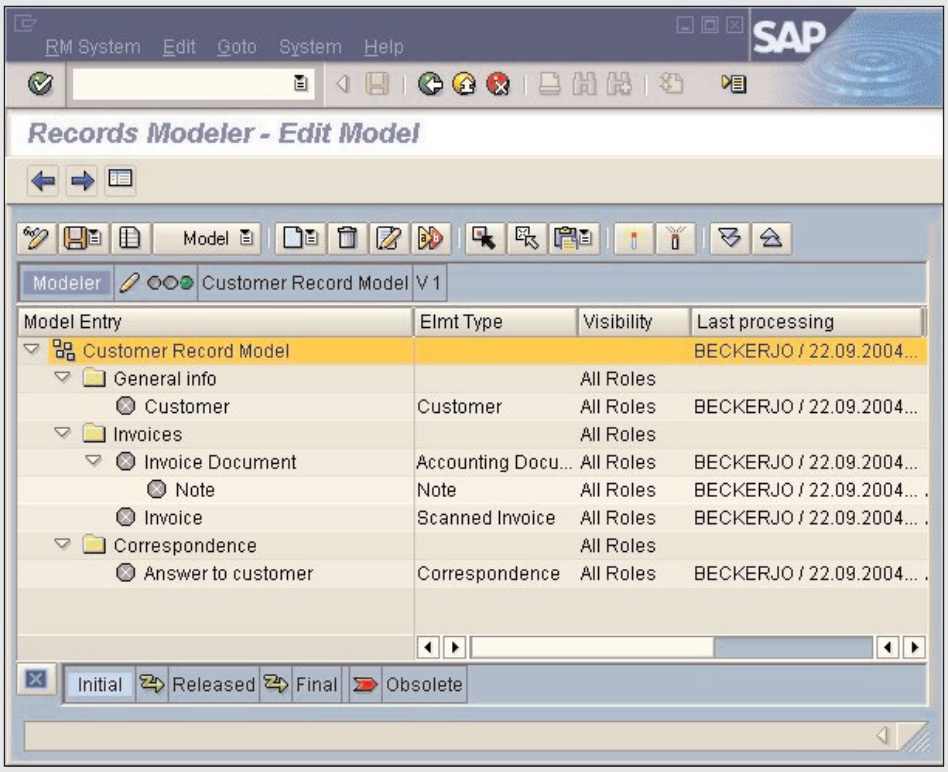


Figure 1 shows the record model designed in the previous article to serve as a template for the example record, including the nodes[4] added to the record hierarchy structure to contain the various record elements (documents, data, etc.); **Figure 2** shows a fully populated record for customer 1000 based on the defined record model in Figure 1. Our goal in this article is to configure the system to automatically add a link in the record to any new service notifications that are posted in the SAP system,[5] so that users can always view an up-to-date list of links to related service notifications in a customer record. You can apply this automatic update technique to any element in your SAP Records Management implementation.

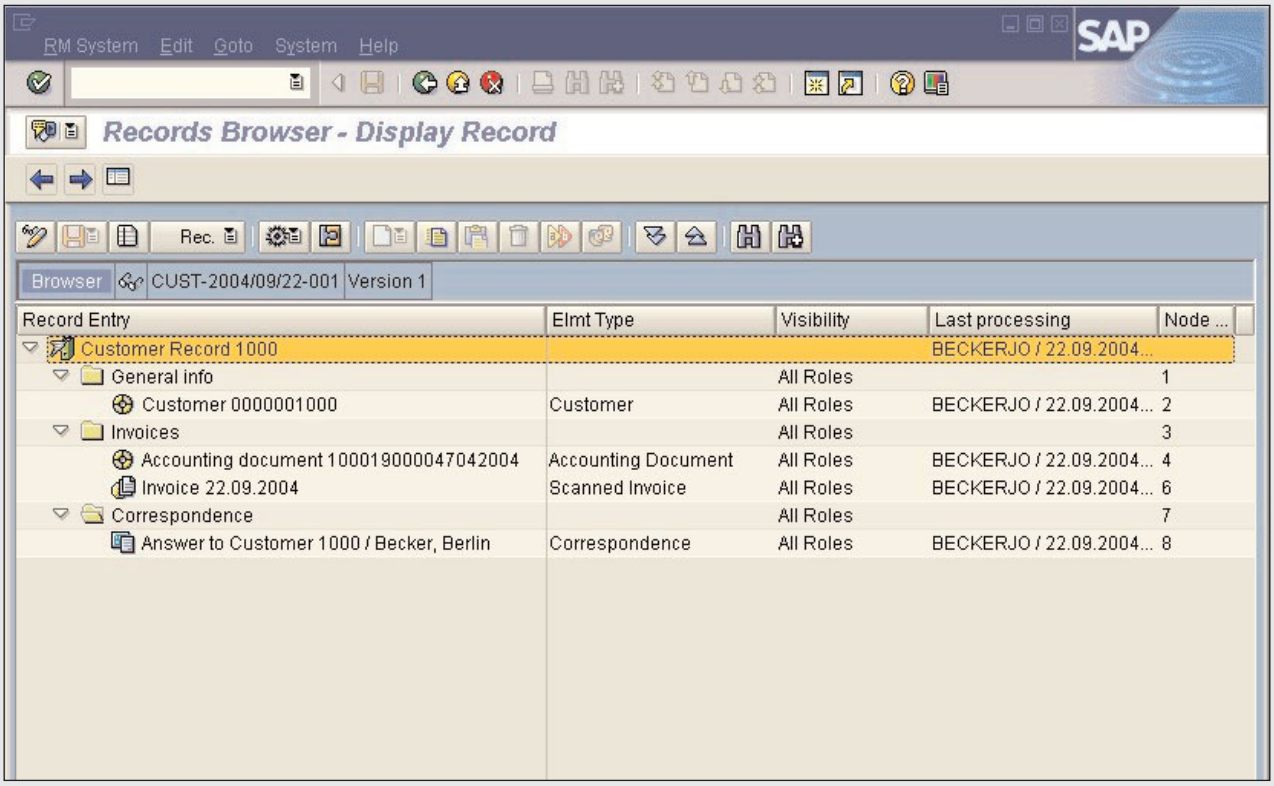
> ✓ ***Tip***
>
> *There are different ways to use SAP Records Management, depending on your needs. The sidebar to the right describes the two main approaches, and when it makes sense to use which.*

To accomplish this goal, we need to perform the following tasks:

[4] A node is a position in the record hierarchy structure (see the sidebar on pages 90-91). It can be a folder or an element (object) that is included in the structure. From an architectural perspective, a node is a building block of the hierarchy.

[5] The example record model defined in the previous article does not include a node for a service notification, as you can see in Figure 2. We will extend the record model to include one in this article.

Figure 2 An Example Record Based on the Defined Record Model



Two Approaches to Using SAP Records Management

There are two key ways to use SAP Records Management:

- **An inside-out approach**, where users use the Records Browser as a hub for all of their records-related tasks in SAP. This is similar to using digital records as a portal or work center. In SAP documentation, you'll find the term "operational usage."

- **An outside-in approach**, where users continue to retrieve and post documents by directly launching individual transactions, as they traditionally do. In this case, digital records are mainly used for documentation of business processes. The term "administrative usage" is sometimes used.

Consider the difference. In the outside-in approach, a user who needs to post a service notification in QM would go directly to one or more transactions, as needed, to do the posting. Sometimes this means going to only one transaction to do the posting, but frequently the user must look up some related information before being able to do the posting. For example, if the user wanted to know the service

(continued on next page)

(continued from previous page)

history for the customer's equipment, he or she might have to run a report, or first look up the equipment number for the customer and then run a service history report. From a user's perspective, the outside-in approach can require navigation to multiple reports or transactions, and several searches to locate the specific data relevant to the customer in question. From an SAP Records Management standpoint, you must use the techniques in this article to configure the system to apply the notification posting to the customer record.

In the inside-out approach, the same user would start by opening the customer's record in the Records Browser. The record would provide links to all of the information related to the customer (e.g., the equipment record for the customer, any prior service notification documents, etc.). Only one or two mouse-clicks (if any) would be required to access the background information,* and posting the notification would involve navigating to the service notification node, right-clicking on it, and choosing Create from the context menu. The Create Service Notification transaction would then appear as usual. Since the transaction was launched via the Records Browser, the service notification would instantly be added to the record — that is, you would not need the automated update configuration described in this article.

In practice, you'll probably choose to use both approaches for different business processes. For example, you might choose the outside-in approach for simple posting tasks, where postings via the Records Browser would take longer overall (e.g., line workers posting production confirmations). You might choose the inside-out approach for tasks that would be simplified by a centralized access to data.

* This is especially convenient if the user has the customer on the phone, and must very quickly navigate to his or her information.

- **Update the SAP Records Management configuration**, so the system knows the record in which to insert the service notification links and where in the record to insert them.

- **Create the event linkage** that binds the service notification event to the event receiver function module.

- **Write the event receiver function module** that inserts the service notification links in the record. The small bit of code involved is provided for download at www.SAPpro.com.

Let's begin by describing our overall approach in a bit more detail, in particular the business objects and BAPIs we'll need to accomplish our goal.

✓ *Technical Prerequisites*

The previous article used an SAP R/3 Enterprise (SAP R/3 4.7) system based on SAP Web AS 6.20 to create the example. Since that time, SAP Web AS 6.40 (included as part of SAP NetWeaver '04) has become generally available, but for consistency, and because it remains prevalent in the marketplace, we will continue to use SAP Web AS 6.20 for this article.

To carry out the configuration settings and implementation described in this article, you must have the authorizations of an SAP workflow administrator and ABAP developer, and you must have the role SAP_BC_RM_ADMINISTRATOR assigned to your SAP user.

> ✓ ***Note!***
>
> *An alternative approach to our example would be to add a link in the record to a report that lists the service notifications for a customer. The disadvantage of this technique is that the record and the notifications would not be visualized simultaneously in the Records Browser.*
>
> *Still another approach would be to develop a new service provider[6] that lists the notifications for a customer using the in-place visualization feature for service provider elements, which would display the record on the right-hand side of the Records Browser and the list of notifications on the left. However, developing a new service provider requires additional development and maintenance effort and detailed knowledge of the Service Provider Framework.*

Implementing Automated Record Updates

Whenever a new business object (e.g., a posting in FI, an order, a service notification) is posted in the SAP system, an event is triggered.[7] Using a technique called *event linkage*, you can bind a workflow or function module to the event, so that the system calls the workflow or function module each time the event occurs.[8]

In this article, we will leverage this eventing capability to insert links to newly created service notifications in our example customer record. Technically, we will bind a custom ABAP function module to the "created" event of the service notification business object. The function module will do three things:

- **Read the customer number from the service notification.** The customer number is defined as an attribute of the service notification.
- **Search for a customer record that matches this customer number.** We will update the example customer record's content model so that all customer records have the attribute CustomerNumber.[9] The function module will use this field to locate the record for the customer in question.
- **Insert a reference to the service notification at the correct position in the customer record.** The function module will call a standard SAP Records Management BAPI to update the record. To ensure the service notification entry is inserted at the proper location within the record, we will update the example customer record model to include a placeholder for service notifications.

Let's explore the business objects we'll use in a bit more depth.

> ✓ ***Note!***
>
> *In the context of event linkage, the function module is called the "event receiver."*

---

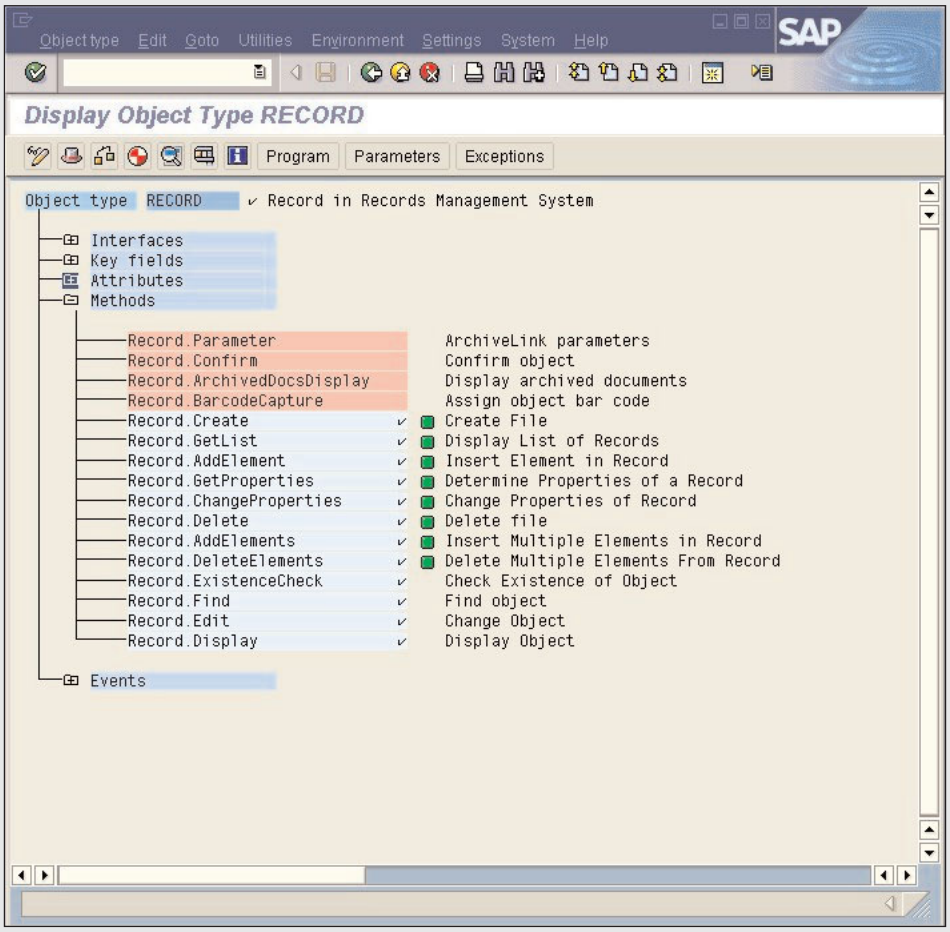
[6] Service providers are ABAP classes that handle the details of accessing the backend resources containing the actual information objects linked to the record (see the sidebar on pages 90-91). SAP provides predefined service providers that meet most needs, though you can also define your own. For more information on defining your own, visit the documentation area at http://service.sap.com/recordsmanagement.

[7] This eventing mechanism is a standard feature of all SAP Basis and SAP Web AS systems, and occurs behind the scenes when you create, change, or delete most business objects defined in the Business Object Repository.

[8] Starting a workflow is appropriate if additional user interaction is required. Keep in mind, however, that using a workflow is more resource-intensive than using a function module.

[9] The content model upon which an element is based defines the attributes that are bound to that element. Content models are the central modeling entities of the Knowledge Provider document management system underlying SAP Web AS (see the sidebar on pages 104-106) and are specified in the DOCUMENT_CLASS element type connection parameter for elements managed by the Knowledge Provider (records, record models, documents, document templates, and notes). As defined in the previous article, the example customer record uses the standard content model for records, SRM_REC00. In this article, we create a custom content model that the record will reference instead.

Figure 3 RECORD Business Object Displayed in the Business Object Repository



The Service Notification (BUS2080) and Record (RECORD) Business Objects

To implement the example, we'll need to leverage methods[10] and events for two SAP business objects (the attributes, methods, and events of SAP business objects can be viewed using the Business Object Builder[11]):

- Business object RECORD, which represents records in SAP Records Management
- Business object BUS2080, which represents service notifications

As you can see in the Business Object Repository display in **Figure 3**, the RECORD object provides a full line of methods for interacting with SAP Records Management objects, including methods to create, delete, read, and search for records, maintain record attributes, and insert content.

[10] The key methods we'll use in the example each have a corresponding BAPI that implements its functionality.

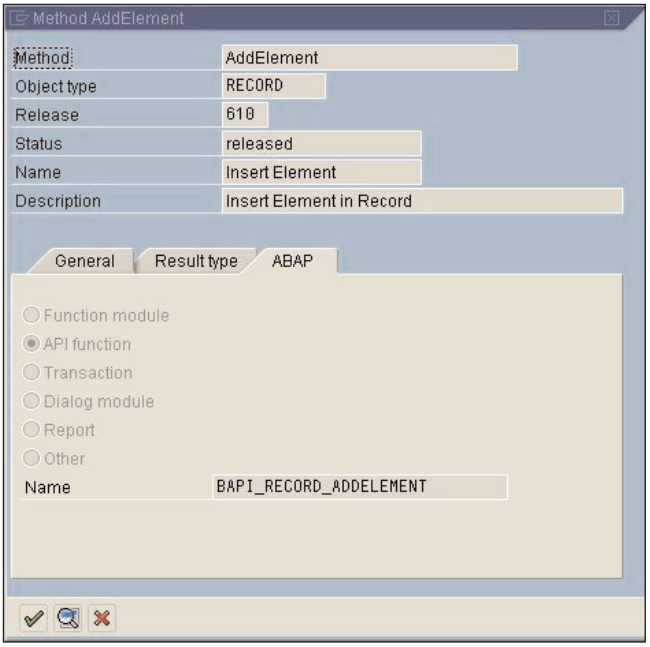
[11] The Business Object Builder is the modeling tool of the Business Object Repository (transaction SWO1).

> ✓ ***Note!***
>
> *Most of the RECORD business object methods are implemented as BAPIs; you can tell which because they are indicated with a green square icon, as shown in Figure 3. To identify the underlying BAPI, double-click on the method name in the Business Object Repository and navigate to the ABAP tab in the dialog box that appears (see* ***Figure 4****). As you can see, the underlying BAPI of the AddElement method is BAPI_RECORD_ADDELEMENT.*

Figure 5 summarizes the BAPIs associated with the key methods of the RECORD business object. In the example, we will use the AddElement method to add new service notification links to the example customer record. This method corresponds to the BAPI BAPI_RECORD_ADDELEMENT.[12]

[12] To add the service notification links to the example customer record, we will actually use another API instead of this BAPI to get around the BAPI's locking mechanism. We'll discuss this in more detail later in the article.

Figure 4 Detailed View of the AddElement Method of the RECORD Business Object

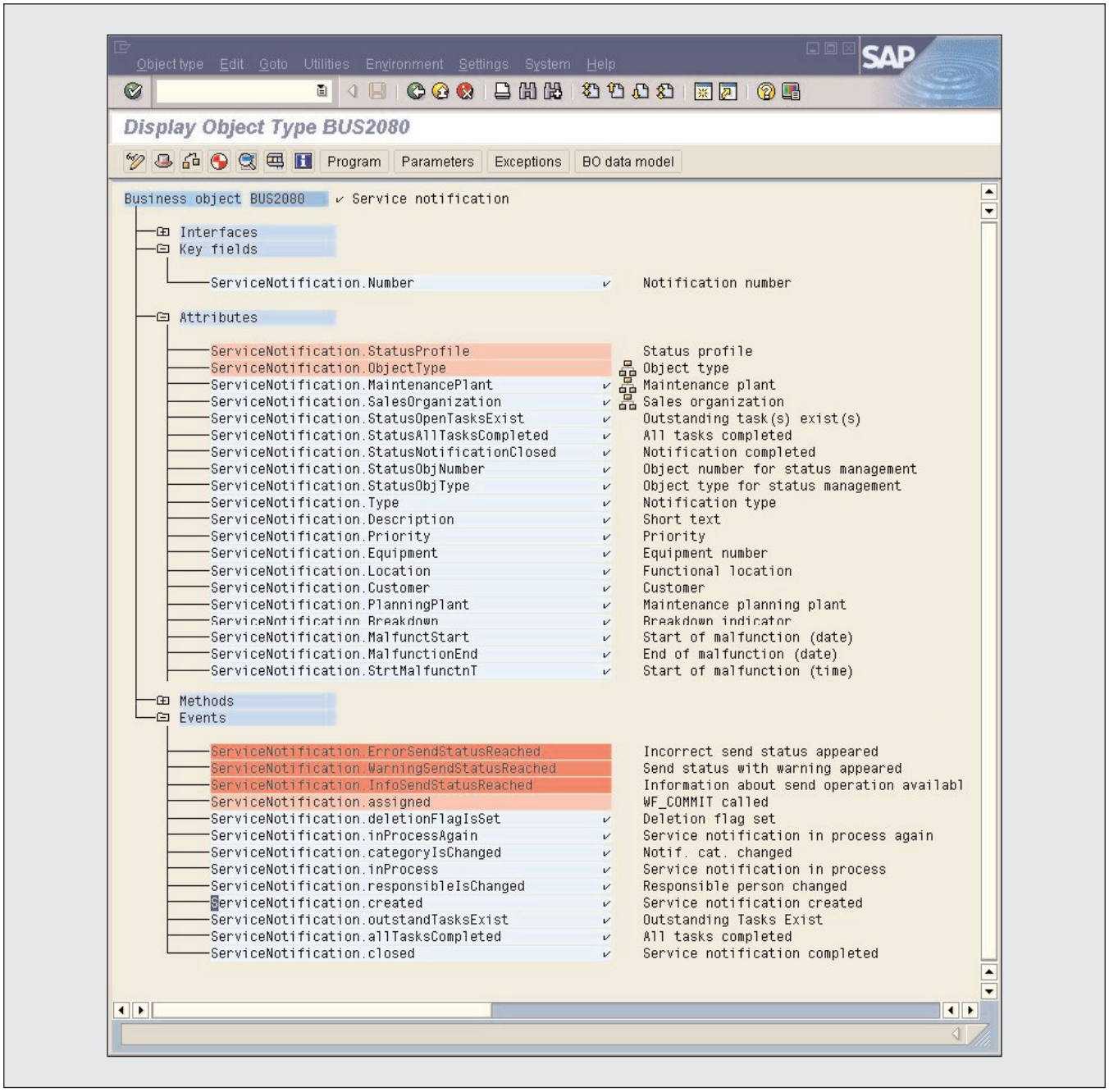


You can also view the details of the service notification object (BUS2080) in the Business Object

Figure 5 Key Methods of the RECORD Business Object and Their Uses

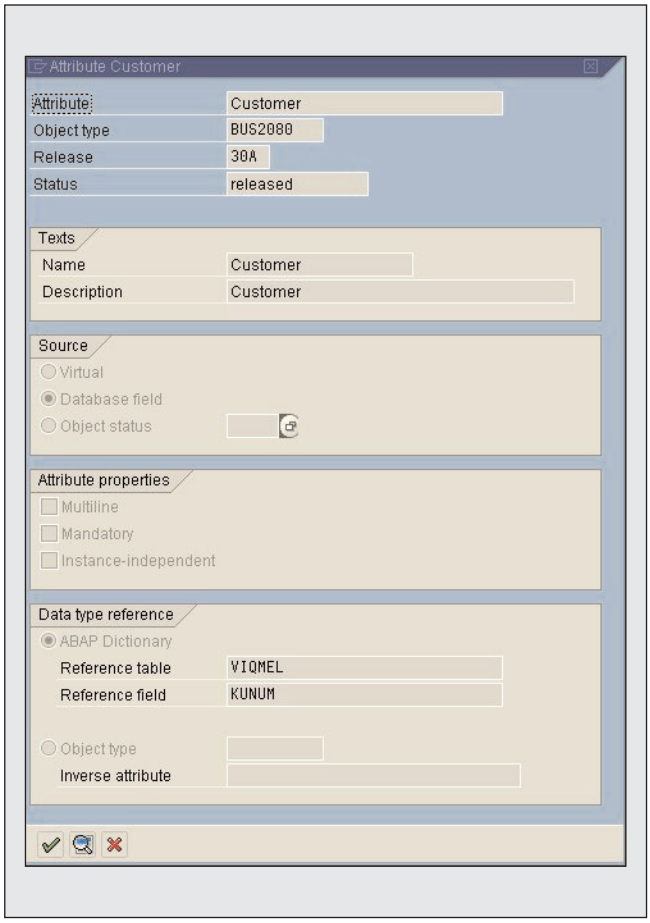
Method	Corresponding BAPI	Description
Record.Create	BAPI_RECORD_CREATE	Create a new record (based on a model configured in the system).
Record.GetList	BAPI_RECORD_GETLIST	Search for records via attributes. The result is a list of records. If an attribute is unique, the list will include exactly one hit.
Record.AddElement	BAPI_RECORD_ADDELEMENT	Insert a reference in a record pointing to another object.
Record.GetProperties	BAPI_RECORD_GETPROPERTIES	Read the attributes of a record.
Record.ChangeProperties	BAPI_RECORD_CHANGEPROPERTIES	Change the attributes of a record.
Record.Delete	BAPI_RECORD_DELETE	Delete a record.
Record.AddElements	BAPI_RECORD_ADDELEMENTS	Add several references in one step.
Record.DeleteElements	BAPI_RECORD_DELETEELEMENTS	Delete references included in the record.

Figure 6 BUS2080 Business Object Displayed in the Business Object Repository



Repository. **Figure 6** shows the event (created) that we will link to our function module, the attribute (Number) that is passed to our function module, and the attribute (Customer) that we'll need to retrieve for our notification. **Figure 7** shows the underlying details for the Customer attribute, accessed by double-clicking on the attribute name in Figure 6. We see at the bottom of the screen in Figure 7 that the customer number for the notification is stored in field KUNUM of table VIQMEL in the SAP database.

Figure 7 *Detailed View of the BUS2080 Business Object's Customer Attribute*



With these additional details in mind, it should be clear how our application comes together. To summarize: An SAP user creates and saves a new service notification. After saving the notification in the database, the system raises the event ServiceNotification.created and looks for any active event receivers (usually workflows or function modules) that need to be notified that the event occurred (i.e., event receivers that are *linked* to the event). Since we will have linked our custom function module to the ServiceNotification.created event, the system calls our function module, passing in an event object that contains — among other things — the number (key) of the new service notification. Our function module will then look up the customer number for the notification by calling BAPI_SERVNOT_GET_DETAIL, try to locate at least one customer record using BAPI_RECORD_GETLIST, and try to insert a new element (link) for the service notification in the record (we'll discuss this in more detail later in this article).

✓ *Tip*

Automatically adding new elements to records using BAPI_RECORD_ADDELEMENT is just the beginning of what you can do with these BAPIs. For example, you can even automate the creation of new customer records when new customers are added to the system. Simply write a function module that calls BAPI_RECORD_CREATE whenever a new customer (represented by business object KNA1) is created in the system. Remember to set values for those record attributes that can be set automatically — like the CustomerNumber attribute used in this article — to avoid manual work. Also, for convenience, be sure to insert a link to the customer master as an element in the record using BAPI_RECORD_ADDELEMENT.

✓ *Tip*

In addition to using these BAPIs to automate updates, consider leveraging them within your custom ABAP programs to add SAP Records Management functionality. For example, enable users to jump to a customer's record by double-clicking on the customer number in a report — code the report to call a BAPI like BAPI_RECORD_DISPLAY, which is associated with the Display method of the RECORD business object (see the system documentation for more details).

With our building blocks in hand, we're now ready to start building the application. Our first task is to update the records management configuration so that the system can identify the record in which to insert the service notification links and where in the record to insert them.

Updating the SAP Records Management Configuration

Before we can write the code to establish the linkage between the record and the service notifications, there are two preparatory steps we need to perform:

1. We need to provide the system with a way to easily identify the customer record in which the service notification links should be stored — let's say the record with customer number 1000. The best way to do this is to define a CustomerNumber attribute for customer records. The system can then find the right record simply by searching for the record with the CustomerNumber attribute set to 1000. To do this, we have to create a new content model, which defines all of the allowable metadata for the records, add a CustomerNumber attribute to it, and assign it to the example customer record, which currently references the standard content model SAP provides for records.

> ✓ ***Tip***
>
> *To make records easier to find, it is best to always include the key of the associated business object as an attribute of the record. While this gives users another field to maintain, they will find it well worth the effort when searching. During training, compare the task to labeling the spine of a binder, since this is an analogy users can relate to.*

> ✓ ***Tip***
>
> *To make users' lives even easier (and to make sure they don't forget), attribute maintenance can be automated with a technique similar to the one described in this article. If you automate the creation of customer records when new customer masters are added to the SAP system, you can set attributes like CustomerNumber automatically simply by calling the BAPI BAPI_RECORD_CHANGEPROPERTIES to update the attribute.*

2. We need to tell the system where in the record new notifications should be inserted. This position will be identified by giving a name to a position in the record model. Technically we will call the name of the position in the record an ANCHOR.

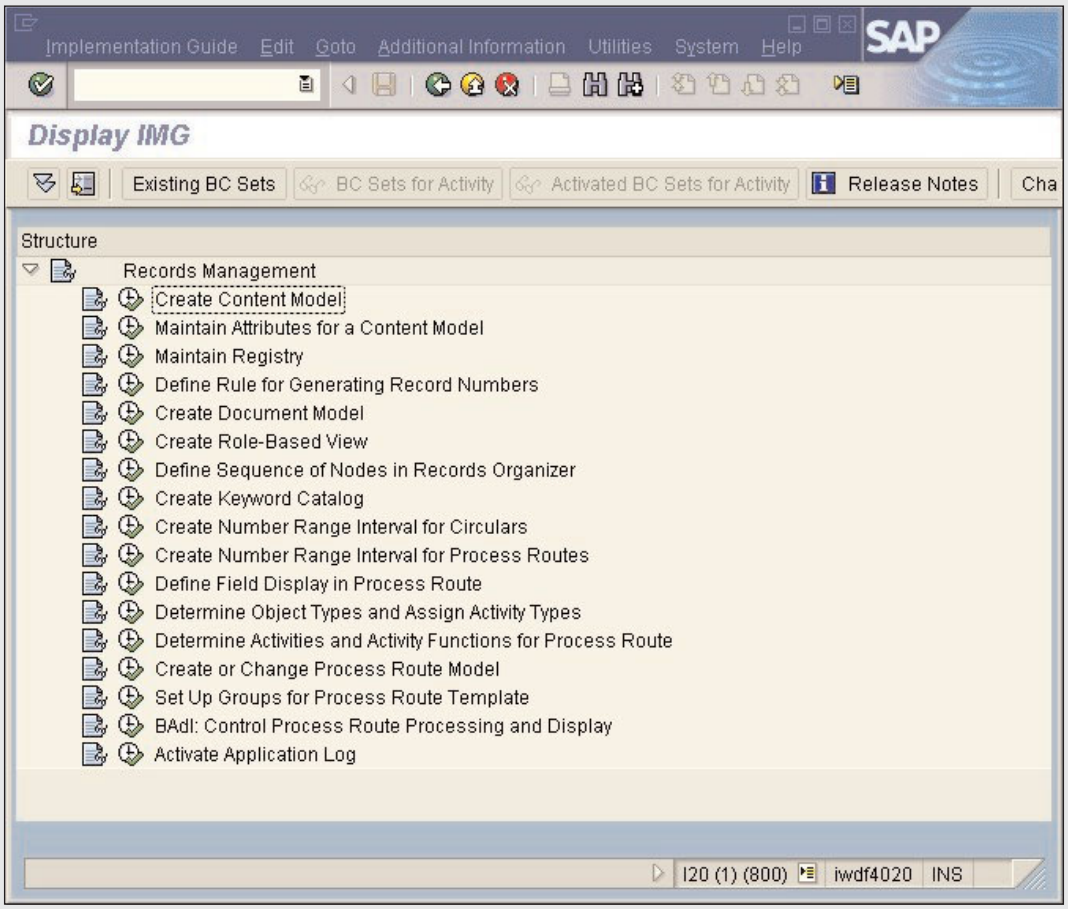
   Let's step through each of these tasks.

> ✓ ***Note!***
>
> *The procedures described in the following two steps are technically detailed and involve a lot of new concepts for many readers. To avoid getting sidetracked from the main objective (automating record updates), I recommend that you just scan through these steps for now, and then read them through in more detail later.*

Step 1: Define an Attribute in a Content Model to Identify the Record

The attributes associated with a digital record are known as a *content model*. In this section, we'll create a new content model for the example customer record and add a CustomerNumber attribute to it in order to facilitate the service notification automation.

Figure 8 Implementation Guide for SAP Records Management



> ✓ ***Note!***
>
> *The previous article employed an SAP standard-delivered content model for the customer record — content model SRM_REC00 (specified in the DOCUMENT_CLASS connection parameter for the customer record element type). We'll create a new content model in this article, in the customer namespace, so we can make the necessary modifications (i.e., add the CustomerNumber attribute).*

Both creating the content model and adding the attribute are initiated from the Implementation Guide for SAP Records Management (see **Figure 8**), accessed via transaction SPRO (Customizing), via menu path SAP Web Application Server → Basis Services → Records Management, or via transaction SRMCUSTOMIZING.[13] The activity names we're interested in are Create Content Model and Maintain Attributes for a Content Model.

[13] Transaction SRMCUSTOMIZING is a shortcut that opens the relevant section for SAP Records Management in the Implementation Guide started with transaction SPRO.

Creating a new content model is easy. All you have to do is launch the Create Content Model activity from the IMG for SAP Records Management (Figure 8), and a wizard appears to guide you through the process. Before launching the wizard, be sure to create a customizing transport request, since you'll need to enter this number on one of the screens. You'll also need to define a package (e.g., ZRM_DEMO) into which the wizard will place the objects it generates.

> ✓ ***Note!***
>
> *Content models are created and managed within a set of services included as part of SAP Web AS called the Knowledge Provider. The Knowledge Provider provides a toolset and a set of runtime components that allows client applications like SAP Records Management to manage multiple versions of information objects[14] in various languages and formats. In the context of this article, however, we will use it only to define the attribute set for the example record.[15]*
>
> *To avoid getting off track, the steps here will focus purely on the task at hand and not seek to explain much about the Knowledge Provider or its Document Modeling Workbench tool. See the sidebar on pages 104-106 for additional background information on the Knowledge Provider.*

Here's what each wizard step does and the key values to enter to create a content model for the example:

1. The first screen explains the purpose of the wizard. No entries are needed.

2. The second screen solicits basic input (see **Figure 9**). You have to enter a package name, a transport request number, and a prefix for the names of the objects that are generated. I recommend that you enter values similar to those shown in Figure 9. It is important to use a value starting with "Z" for the prefix of the object names, since the objects need to be created within the customer namespace.

3. The third screen asks which type of content model you want to create. To create a content model for a record, highlight Records, as shown in **Figure 10**.

4. The fourth screen asks whether you wish to use shared or separate tables to store the data. We recommend using separate tables to improve performance and simplify reporting.

5. The fifth screen asks whether you want to store the content in the SAP R/3 database or in an external location like the standard SAP Content Server or a third-party content server. For performance reasons, we strongly recommend that you store digital records in the database of your SAP application server.

6. The sixth screen indicates that you've successfully completed all input.

If all goes well, the wizard creates a new content model called Customer Record and its associated tables in the SAP R/3 database. You can verify that the process completed successfully by opening package ZRM_DEMO within the ABAP Workbench (or the Document Modeling Workbench, discussed next).

We're now ready to add an attribute to our new content model. Since SAP Records Management content models are defined and managed by the Knowledge Provider (see the sidebar on pages 104-106), we do this within the Knowledge Provider's central design tool, the Document Modeling Workbench.

---

[14] The term *information object* is used instead of *document* because the Knowledge Provider can handle more than just Microsoft Word and Excel documents, for example. Indeed, the content model we'll create here represents a record in SAP Records Management, which is surely not a document in the traditional sense. The Knowledge Provider then provides the functionality to coordinate multiple versions of these records.

[15] The Knowledge Provider probably comes across as overly complex for the task of defining a content model. Leveraging the Knowledge Provider provides many other technical benefits, however, that are beyond the scope of this article.

Figure 9 *Enter a Description for the Content Model*

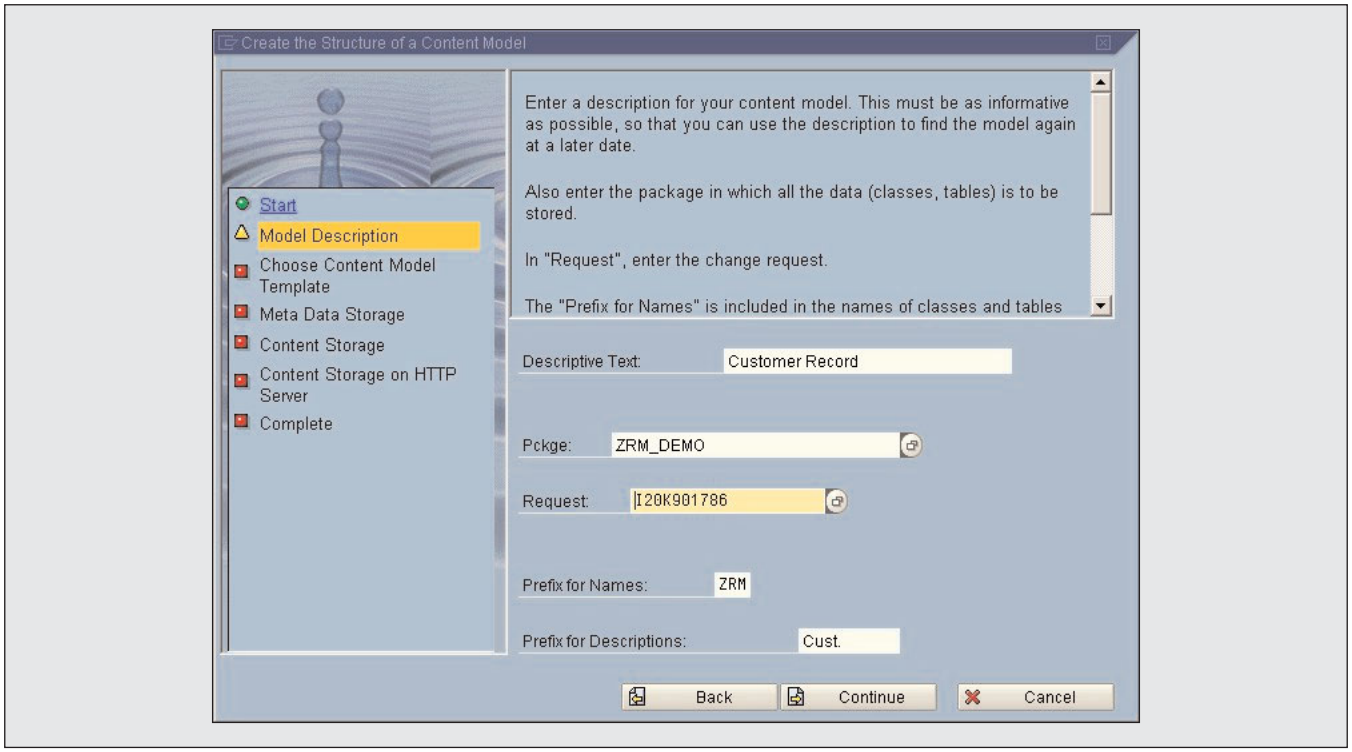
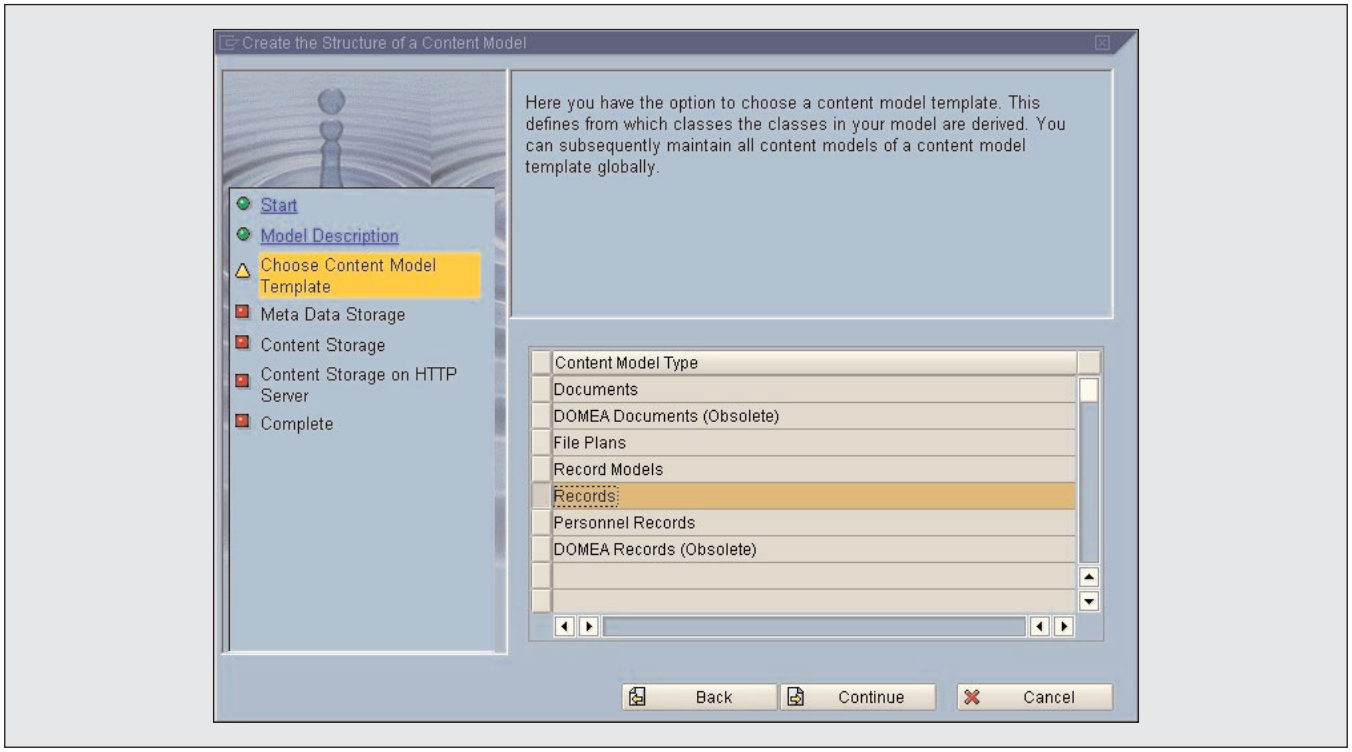


Figure 10 *Select a Type for the Content Model*

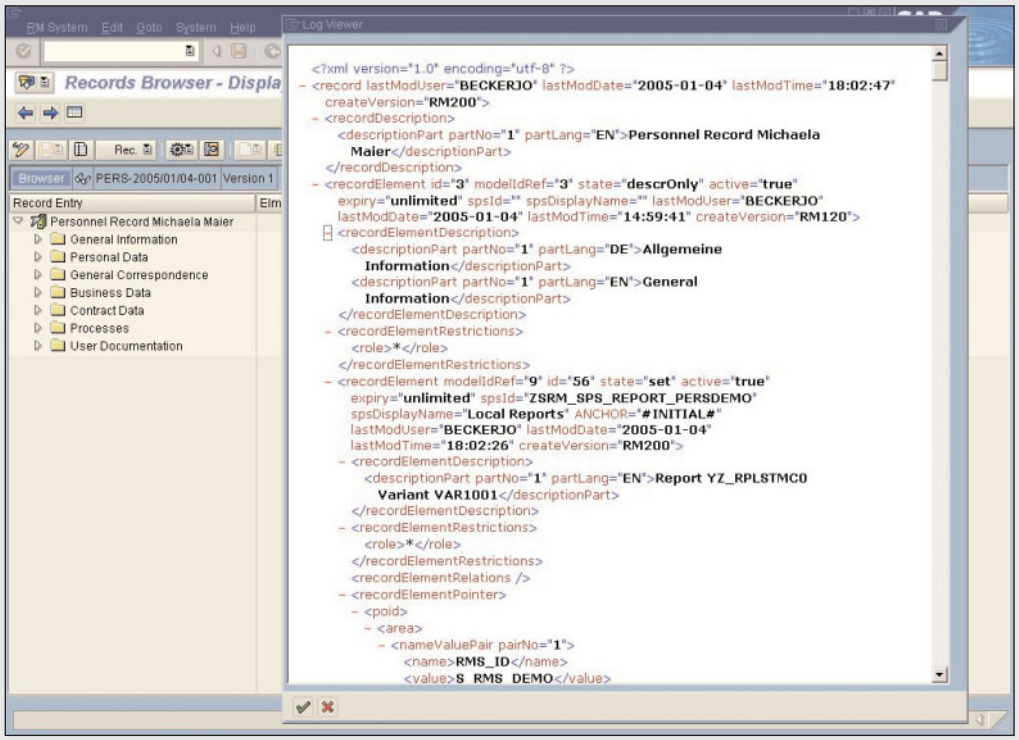


Understanding the Knowledge Provider

The Knowledge Provider offers a set of generic backend services that support the management of documents in the SAP system. These services are used in many different SAP applications to provide document management functionality — e.g., the SAP Document Management System (DMS), SAP Knowledge Warehouse, mySAP CRM Content Management, and SAP Records Management.* Similar to data dictionary services, Knowledge Provider services belong to the infrastructure layer of the SAP system, and in most cases you won't even know that the Knowledge Provider is running behind the application you are using. It is worth getting to know the basic principles of the Knowledge Provider, since it underlies so many SAP applications involving digital documents and adds a considerable number of functional enhancements to these applications.

What It Does

Have a look at the screenshot below.



The screenshot shows the XML representation of a digital record (in this case, a personnel record),

---

* The DMS centrally stores information records (stubs) for documents, so that they can be referenced easily from SAP transactions. SAP Knowledge Warehouse is a system that offers SAP training and system documentation and allows you to adapt this content to special customer situations. mySAP CRM Content Management stores and manages information (documents, graphics, multimedia files, etc.) related to CRM business objects (products, catalogs, business partners, etc.).

which is stored in the Knowledge Provider and is hidden from the end user.** As you can see, there is a one-to-one representation between the elements of a record structure and the elements included in its corresponding XML document — the Knowledge Provider stores and retrieves XML documents that SAP Records Management interprets as hierarchies of documents and other elements.

In addition to managing the digital record structure, the Knowledge Provider also manages documents (such as Microsoft Word documents and notes attachments) that are included in digital records. In fact, the Service Provider Framework includes a special service provider for such documents.***

The Knowledge Provider services provide SAP applications with independence from a document's storage repository (e.g., database, SAP Content Server, third-party archive system, file system), and independence from the document's format (e.g., .tif, .pdf, .doc). Furthermore, the Knowledge Provider supports versioning, multiple languages, and attribute definition, among others, for the applications it underlies. The services are implemented in ABAP and are a standard part of the SAP Basis system.

How It Works

There are two key terms you need to know to understand how the Knowledge Provider works: logical documents (also known as logical information objects) and physical documents (also known as physical information objects).

An easy way to understand logical and physical documents is to consider an example scenario: drafting a legal contract. When negotiations start, assume a first draft of the contract is written in Microsoft Word. The two parties check details and propose changes. During this process, several versions of the contract are created. Assume also that since Company A's headquarters is in Walldorf, Germany, and Company B's headquarters is in Redmond, Washington, the contract is translated into German and English. Finally, imagine that the final version of the document will be printed in PDF form (to protect it from changes) and that it will be published as an HTML page to the intranets of both companies.

In this example, the contract is a single *logical document,* since from a business perspective there is just one object — "contract." The contract document exists in multiple versions, in multiple languages (English and German), and in multiple formats (Microsoft Word, PDF, HTML, etc.). We would say that there are many (concrete) *physical documents* that represent the logical document. Each physical document corresponds to exactly one version of the document, is written down in exactly one language, and is stored in exactly one format.

This is the model that the Knowledge Provider uses. The logical documents, physical documents, and attributes that relate to a particular type of document (e.g., a contract) are organized in a content model, which is defined in the Knowledge Provider's Document Modeling Workbench (transaction DMWB).

** Users never maintain these XML documents directly; they use the Records Browser to make changes. You can think of the Records Browser as a highly specialized XML editor for SAP Records Management.

*** Both the Service Provider Framework and the Knowledge Provider are general infrastructure concepts that are used in SAP Records Management. While the Knowledge Provider solves problems that arise from the management of documents, the Service Provider Framework enables different information objects to take part in a single application. In other words, the Service Provider Framework offers integration; the Knowledge Provider offers document functionality.

(continued on next page)

(continued from previous page)

One great feature of this design is how easy it is to retrieve documents. Normally, users have to sift through tens or hundreds of documents on a file share to locate the needed version and format. When using applications that leverage the Knowledge Provider, the user only needs to identify the desired logical document and specify the preferred version, language, and format. Often, a few of these values — like language — can be specified by the program automatically based on the user's logon information or the business context (e.g., a workflow may only be interested in retrieving the most recent version). If a format or language isn't available, the Knowledge Provider can even propose the closest "fit" to the user's request. In this way, the Knowledge Provider provides a flexible framework that all SAP applications can freely use to manage documents with many versions, languages, and formats.

Launch the Document Modeling Workbench via transaction DMWB and follow these steps:

1. When you start the Document Modeling Workbench, you'll see a list of SAP applications that use the Knowledge Provider internally. In this article, we are only interested in SAP Records Management, so open the SRM node as shown in **Figure 11**. In the tree, you'll see subfolders for attributes (IO attributes), logical and physical information objects (LOIO classes and PHIO classes[16]), plus a few other objects we need not concern ourselves with. Our goal here is simply to define a new IO attribute called CustomerNumber and assign it to the LOIO class that represents the logical information objects.

2. Create a new IO attribute. Right-click on the IO attribute folder and choose Create from the context menu. In the pop-up that appears (see **Figure 12**), enter a technical name (e.g., ZRM_CUSTOMER) and a description (e.g., CustomerNumber) for the attribute, and then click on the Enter button ( ). A new attribute called ZRM_CUSTOMER will appear in the list of IO attributes. The attribute appears in blue to indicate that it is not yet active.

[16] IO attributes are input/output attributes, LOIO is short for logical information object, and PHIO stands for physical information object.

✓ ***Note!***

A logical information object is a conceptual document or record — such as a legal contract document or a customer record in SAP Records Management (refer to the contract example in the sidebar on pages 104-106). A physical information object is the actual version of the logical information object in a particular language and format (e.g., version 2 of the contract, in English, and in PDF form). Each content model contains a logical document class (LOIO class) and a physical document class (POIO class) that together form the content model. IO attributes are defined independently, but are assigned to a logical document class or a physical document class.

✓ ***Note!***

The terms "logical information object" and "physical information object" are used instead of logical document and physical document because the Knowledge Provider can handle more than just Microsoft Word and Excel documents, for example. Refer to the sidebar on pages 104-106.

Figure 11 The Document Modeling Workbench Initial Screen

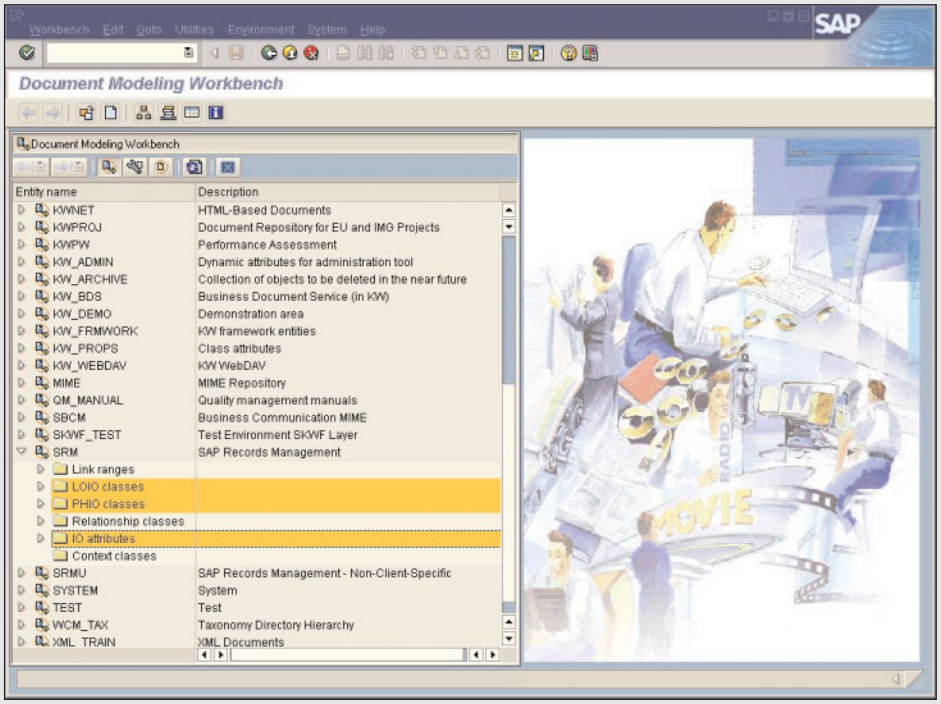


Figure 12 Creating a New IO Attribute

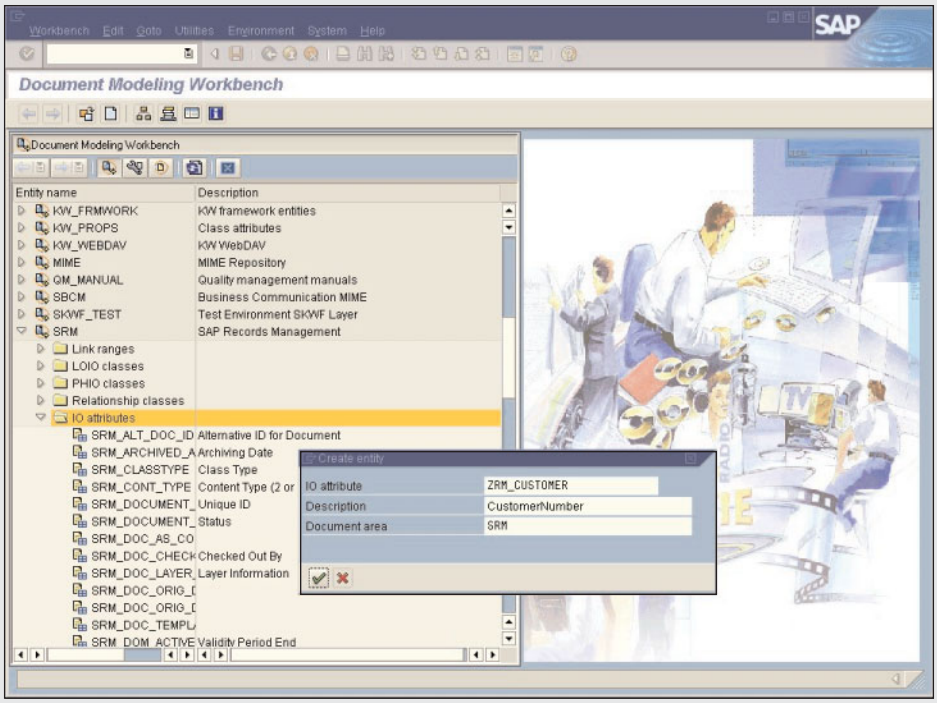
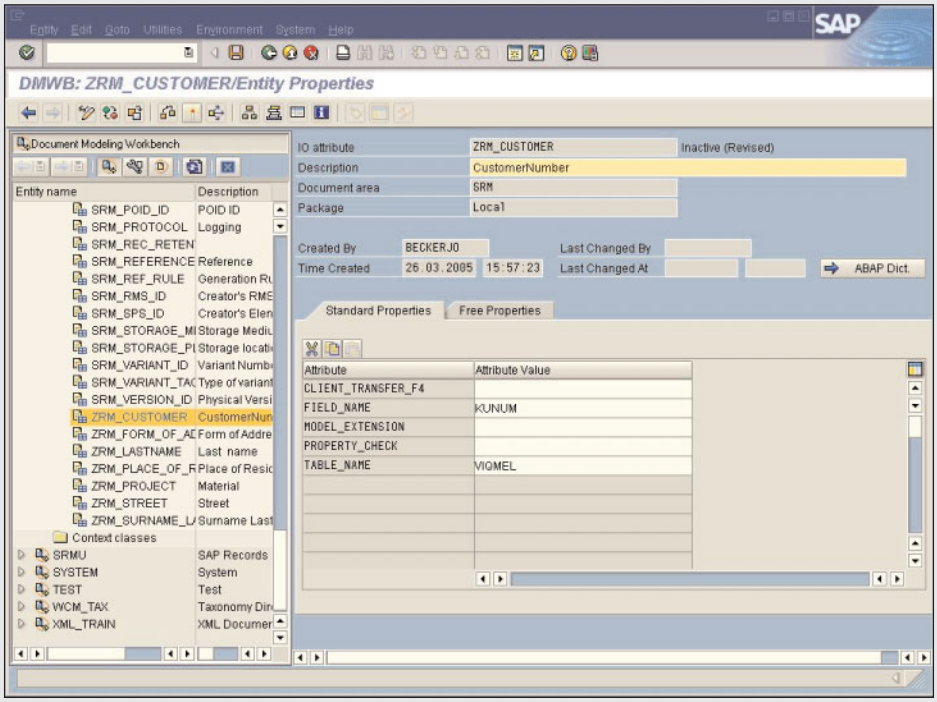


Figure 13 Defining the Details of the New IO Attribute



3. To maintain the details for this attribute, double-click on the attribute name and enter VIQMEL for the table name and KUNUM for the field name as shown in **Figure 13** (recall that in Figure 7 we noted VIQMEL-KUNUM as the table field underlying the customer number attribute of the service notification business object).

> ✓ ***Note!***
>
> *Entering these table and field values in the IO attributes provides a reference to the semantic description for VIQMEL-KUNUM in the data dictionary. When maintaining the record attributes, the help text and value help will be taken from this data dictionary description.*

4. Save and activate the new attribute by clicking on the Activate button ( ).

5. We next need to bind the attribute to the content model we defined for our record. Navigate back to the Document Modeling Workbench initial screen (Figure 11) and expand the LOIO classes node. Look for the record class ZRM_RECORD_V and double-click on the customer record subclass ZRM_REC01_V below it (see **Figure 14**). A screen like **Figure 15** will appear, on which you can add the attribute.

> ✓ ***Note!***
>
> *The actual compiled logical object class is ZRM_REC01, which appears under ZRM_REC01_V in the hierarchy. ZRM_REC01_V, which is a subclass of record class ZRM_RECORD_V, is a virtual working copy of the logical object class used for making any changes to the actual class. When you click on the Activate button after making changes, the system generates an updated version of the "real" class (ZRM_REC01).*

Figure 14 Hierarchy of Logical Document (LOIO) Classes

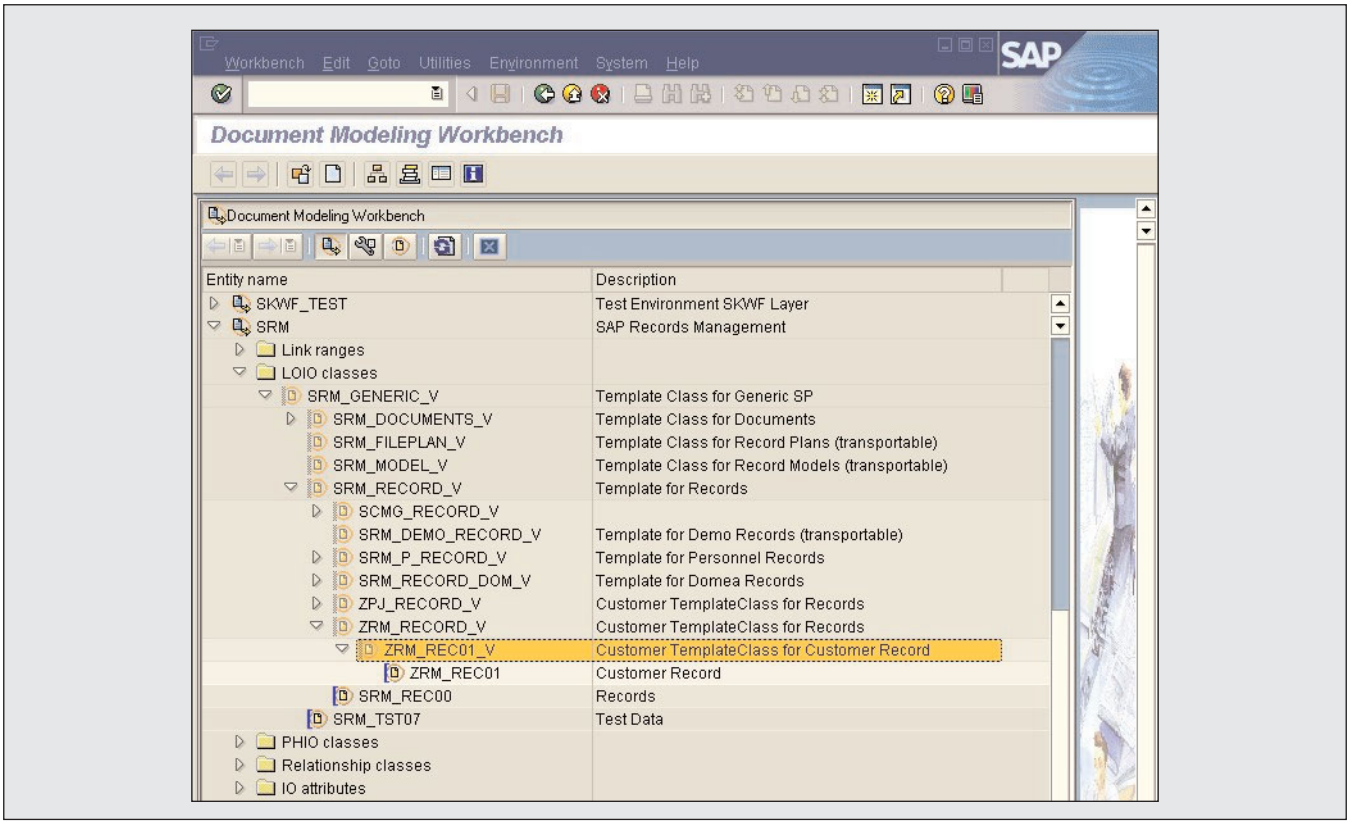


Figure 15 Details of the Logical Document (LOIO) Class

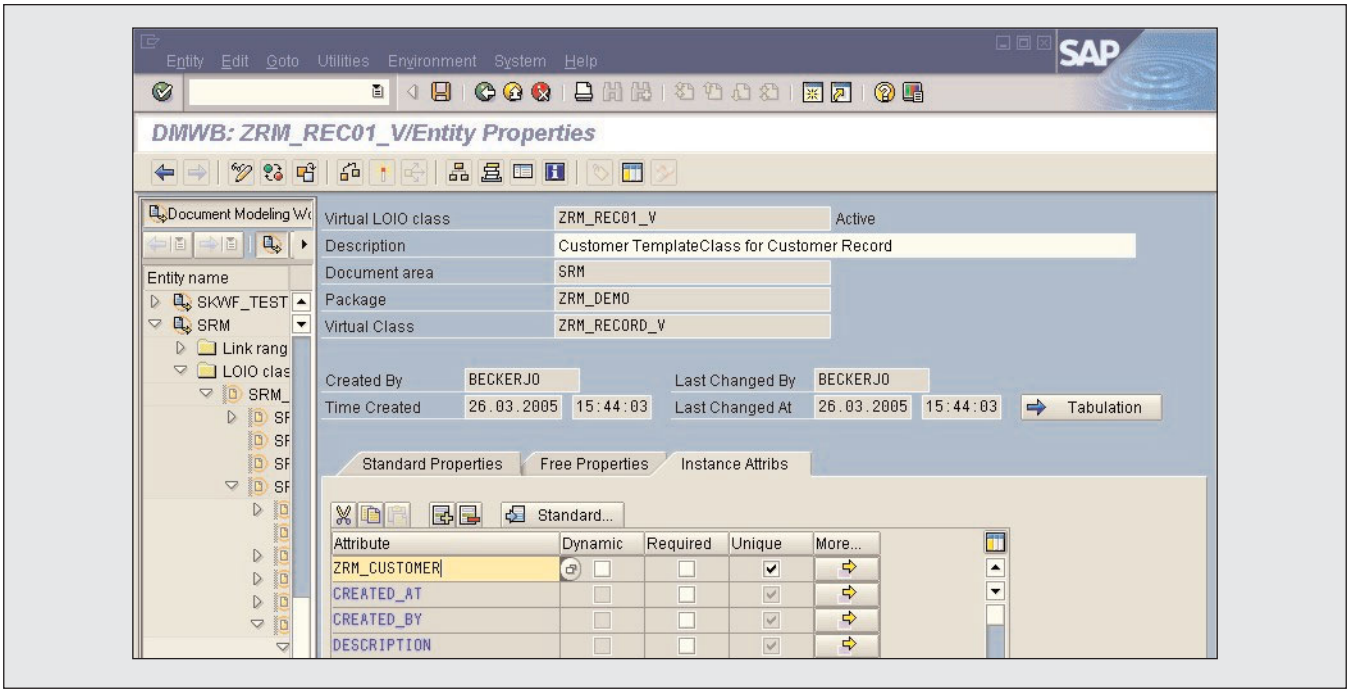


Figure 16 Navigating to the Registry of Service Providers and Element Types

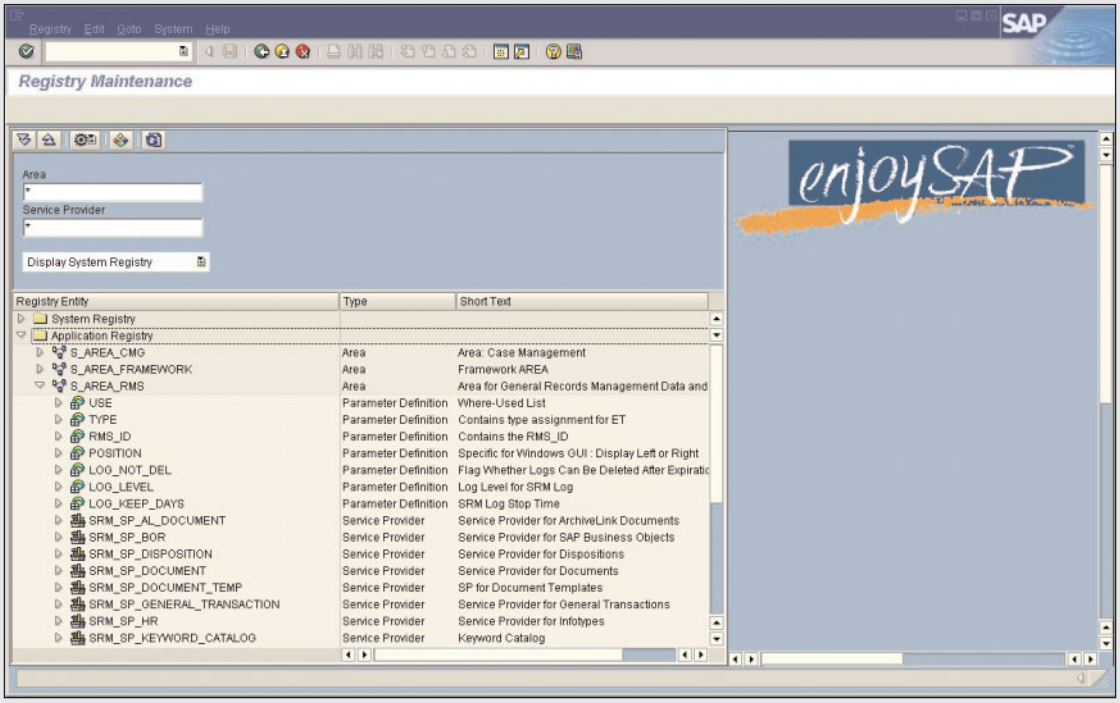
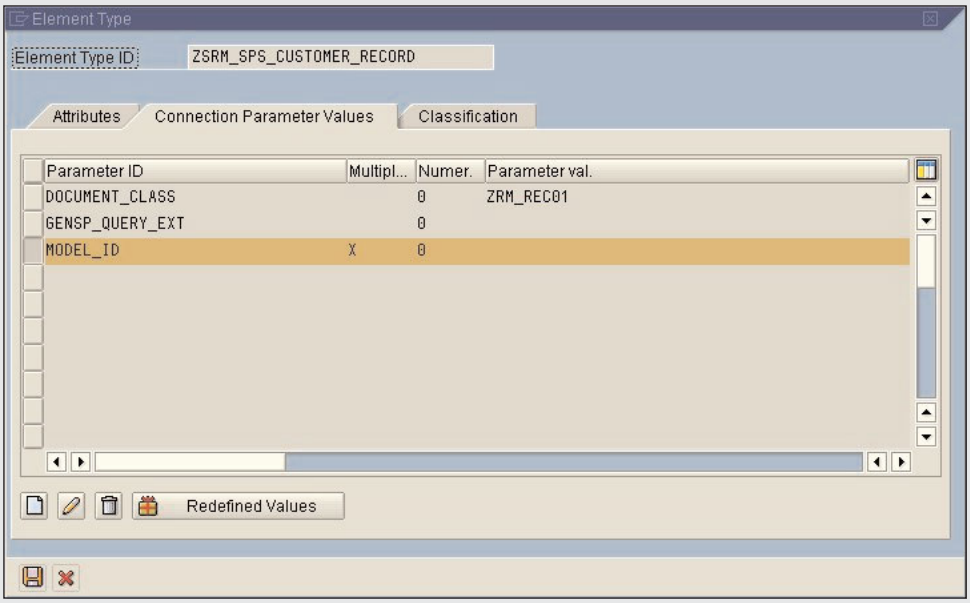


Figure 17 Updating the Content Model for Customer Records



> ✓ ***Note!***
>
> *Recall that we earlier specified "ZRM" (refer back to Figure 9) as the prefix the wizard should use when creating the content model. The wizard also automatically includes the word "RECORD" in generated SAP Records Management classes, which makes them easy to spot in the Document Modeling Workbench.*

> ✓ ***Note!***
>
> *You may be wondering why we're adding the attribute to the logical class instead of the physical class, and whether this decision is arbitrary. The short story is that, for SAP Records Management, the logical class should be used for version-independent attributes, while the physical class should be used for version-dependent attributes. Since the customer number forms the identity of our example customer record, and it is not expected to change across versions of a given customer record, we've chosen to add the CustomerNumber attribute at the logical class level.*

6. Once you've added the CustomerNumber attribute to the logical class, click on the Activate button ( ) to regenerate the class.

7. The final, critical step is to tell SAP Records Management to use the new content model as the basis for customer records. Navigate to the Registry[17] (transaction SRMREGEDIT) and expand the S_AREA_RMS item (which contains the objects and parameters associated with SAP Records Management), as shown in **Figure 16**. Then navigate to the SRM_SP_RECORD service provider, expand the list of element types, right-click on theZSRM_SPS_CUSTOMER_RECORD element type, and choose Change from the context menu. Finally, navigate to the Connection Parameter Values tab (see **Figure 17**), change the DOCUMENT_CLASS parameter from SRM_REC00 (the standard record content model currently specified for the example) to ZRM_REC01 (the newly created content model), and save to complete the update.

[17] See the January/February 2005 article for complete details on the Service Provider Framework and how to work with the underlying Registry.

> ✓ ***Note!***
>
> *We recommend that you delete any records in your test environment that are based on the element type ZSRM_SPS_CUSTOMER_RECORD to avoid potential data inconsistencies.*

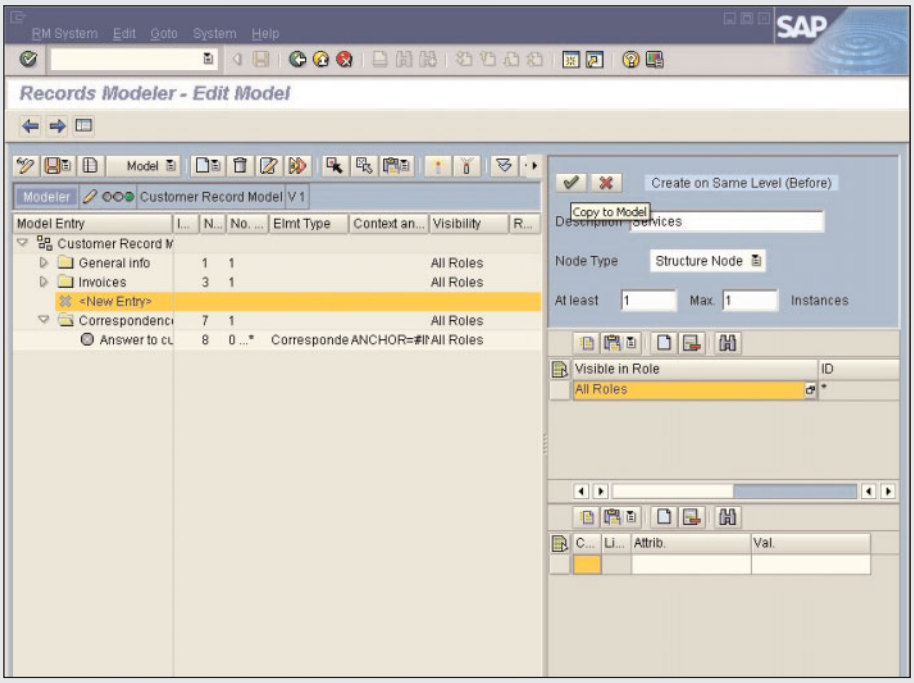
> ✓ ***Note!***
>
> *If the ZSRM_SPS_CUSTOMER_RECORD element type isn't present in your Registry, refer to the January/February 2005 article for details on how to create it.*

Step 2: Add a Service Notification Node to the Record Model

We now need to enhance the existing record model by adding a new node for service notifications. We also want to define this node as the position in the record where links to new notifications are automatically inserted.

Since elements within record models are based on element types, we must first define an element type for service notifications so that we can add this type of node to the model. This will be an element type for an information object that is derived from the service provider for business objects (SRM_SP_BOR).

Figure 18 Definition of the Services Structure Node



> ✓ ***Note!***
>
> *A record model is a template that defines the structure of a record and which content (in terms of element types) is allowed or needed in a digital record (see the sidebar on pages 90-91). Each record derives from a single record model. The January/February 2005 article describes in detail how a record model is defined using the Records Modeler.*

To create an element type for service notifications, follow these steps:

1. Open the Registry (transaction SRMREGEDIT) and expand the SAP Records Management area S_AREA_RMS.

2. Right-click on the entry for the service provider for business objects (SRM_SP_BOR) and choose Create from the context menu.

> ✓ ***Note!***
>
> *If it is difficult for you to carry out these steps, have a look at the January/February 2005 article, where the Registry was discussed in detail.*

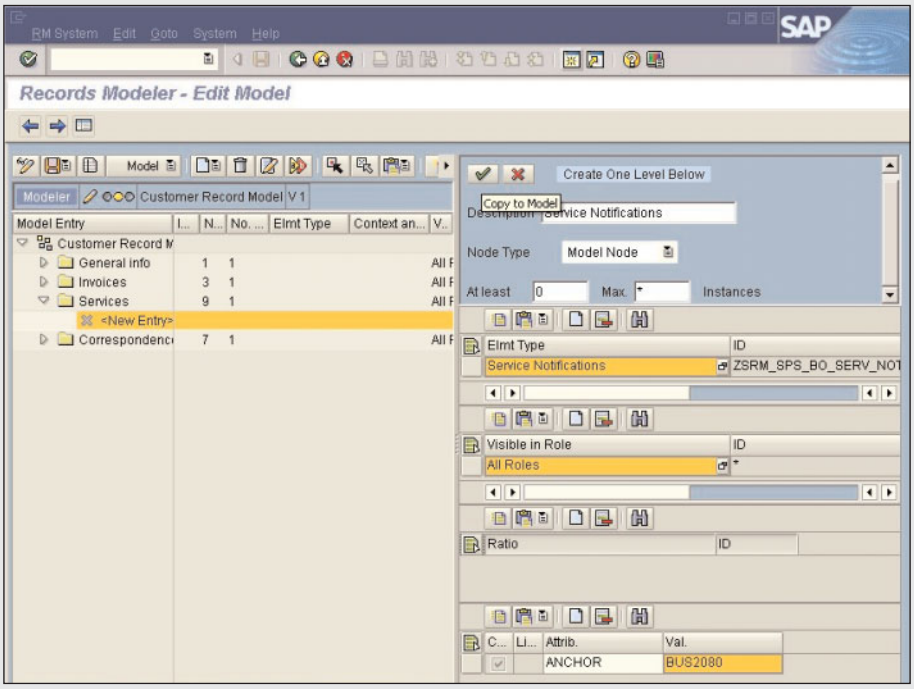
3. On the Attributes tab, enter the element type ID ZSRM_SPS_BO_SERV_NOTIFIC and the short description Service Notifications.

4. On the Connection Parameter Values tab, specify the following:

   - **BOR_OBJECT_TYPE:** BUS2080
   - **LOGICAL_SYSTEM:** NONE
   - **METHOD_BOR_OBJECT_DISPLAY:** DISPLAY

5. On the Classification tab, specify the following:

Figure 19 Definition of the Service Notifications Model Node



- **TYPE:** SRM_BUSINESSOBJECT

- **RMS_ID:** Z_DEMO_RMS_ID

We next need to define a new structure node (folder) and a new model node (object placeholder) for our notifications in the customer record model (see the sidebar on pages 90-91 for more on structure and model nodes). If you read the January/February 2005 article, most of the steps involved will be familiar. The one new concept is that of *anchors*, which are symbolic names that are assigned to nodes so that you can reference the nodes more easily. Without anchors, nodes in SAP Records Management are identified by a node number, which can be hard to remember. Plus, node numbers can sometimes change during record redesigns; anchors protect you from such changes.

Here's how to add the new node and anchor to the record model:

1. Start the Records Organizer (transaction ORGANIZER). Right-click on the element type Record Models and select Find from the context menu to search for the record model Customer Record Model. Switch to change mode (**)**.

2. Add a new folder called Services just after the Invoices node, but before the Correspondence node. Right-click on the Correspondence node and choose "Create on same level before" from the context menu. As shown in **Figure 18**, identify the node as a structure node named Services. Specify at least 1 instance with a maximum of 1 instance, which defines the cardinality of the node as 1:1 (meaning that exactly one folder named Services will appear in all records that are derived from this model). Set the Visible in Role field to All Roles, so that all SAP R/3 user roles can view the folder.

3. Define an object placeholder for the actual service notification links. Right-click on the newly created Services structure node and choose "Create one level below" from the context menu. As shown in **Figure 19**, identify the node as a model

node and name it Service Notifications. Specify at least 1 instance with a maximum of * instances, which defines the cardinality of the node as 1:*n* (meaning that at least one and up to an unlimited number of service notifications can be inserted); otherwise, the function module's call to BAPI_RECORD_ADDELEMENT, which we define in the next section, will fail after the first service notification is inserted. Set the Visible in Role field to All Roles, so that all SAP R/3 user roles can view the links. Restrict the type of objects that can be assigned to the node to service notifications by setting the Elmt Type field to the ZSRM_SPS_BO_SERV_NOTIFIC element type we defined earlier.

> ✓ ***Note!***
>
> *You can add nodes freely to existing record models, but the system will prevent you from deleting them. This protects existing records based on the model from corruption.*

4. Next, define an anchor by entering ANCHOR in the attribute field along with an anchor name in the value field. The name you choose must be a unique name for the record model. As you can see in Figure 19, we've chosen the name BUS2080, which is the name of the service notification object. Whatever name you choose, remember it because you'll need it when defining the function module (more on this in the next section).

> ✓ ***Tip***
>
> *To simplify coding maintenance, choose anchor names that clearly explain the type of content to be inserted. For example, if you want to insert business objects, consider specifying the name of the business object type to be inserted (e.g., we specified BUS2080, which is the name of the service notification object in the Business Object Repository).*

5. Finally, to apply the changes to the record model, click on the  button and save the model.

While it's taken quite a few steps, we've now made the necessary updates to our SAP Records Management configuration — i.e., all customer records will have a CustomerNumber attribute that users can (and should) complete, and all records will include a placeholder into which we can insert service notifications using the BAPIs explored earlier (refer back to Figure 5).

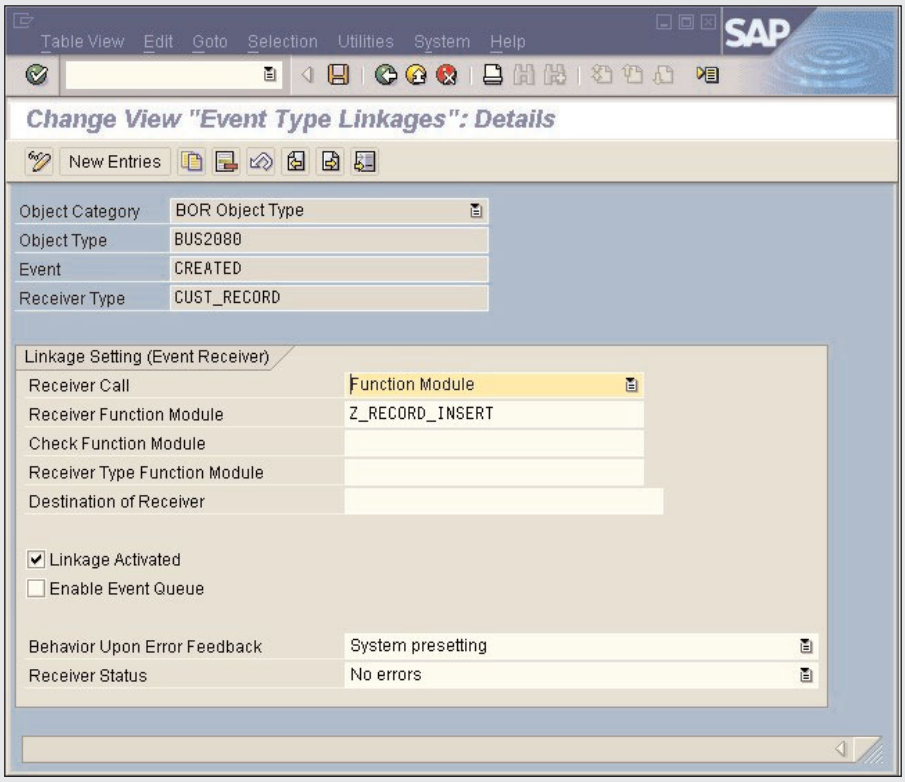
We're now (finally!) ready to write the function module to make the updates happen and configure the system to call the function module when new notifications are created in the SAP system.

Setting Up the Event Linkage

In this section, we look at how to use the event linkage concept to automatically add a link to a service notification in a customer record. The idea behind event linkages is to separate the occurrence of an event — e.g., the creation of a business object — from

> ✓ ***Note!***
>
> *In the SAP system, the event linkage concept is primarily designed to support the needs of SAP Business Workflow. Workflows often need to be started upon events occurring in business applications. The idea of a loose coupling instead of a hard-coded call is ideally suited for making the workflow integration into applications configurable.*
>
> *In the case of our example, we don't want to start a workflow when a notification is created; we just want to add the notification to a record, and we don't need any user interaction in order to do so. Because the event linkage concept has been developed in a generic way, we can use it for our purposes. The difference is that we are dealing with a different receiver type and that the receiver is started using a different receiver function module.*

Figure 20 *Event Type Linkage of BUS2080*



the applications that want to react to the event. This way, the creator of the event doesn't have to care about the applications. To be more precise, the application creating an event doesn't need to make any kind of call to potentially interested applications; it doesn't even have to know these applications exist. This separation frees developers from having to manage the specific ways that the applications communicate with each other and the specific interfaces that have to be agreed upon. Instead, there is just one generic method of communication: one application publishes an event and other applications may receive this event. You can think of the event publisher as a radio program broadcast and of the event receivers as radios tuned to that particular station.

Let's get started. To set up the event linkage in the system, we use transaction SWETYPV. **Figure 20**

✓ ***Note!***

It would also be possible to link a workflow to the "created" event of the notification and insert the notification into the customer record in a background step of the workflow. However, this would produce some unnecessary overhead without yielding any advantages. The situation would be different if inserting notifications into the record required user interaction. Consider a faxed document that has been scanned into the system and must be inserted into a record. In this scenario, user interaction might be required to classify the document and identify the record. Workflow is ideally suited to cases like this.

shows the configuration of the linkage we need to react to the creation of a service notification:

- The **Object Category**, **Object Type**, and **Event** fields identify the event we want to link to.
- The **Receiver Type** field allows you to enter an identifier (without system checks) that distinguishes our event linkage from other event linkages to the same event. In the example, we specify CUST_RECORD as the receiver type.
- The **Receiver Call** field specifies the type of call that is made when the event occurs and the **Receiver Function Module** field specifies the name of the function module we will implement (Z_RECORD_INSERT in the example).
- The **Check Function Module** field evaluates if the linkage should be executed at all and the **Receiver Type Function Module** field can dynamically determine the receiver type during runtime; neither of these are relevant for the example because we want the system to call the function module whenever an event is raised.
- The **Destination of Receiver** field specifies an RFC destination if the receiver runs in a remote system. We assume that the example takes place inside a local system, so this field is not relevant for our purposes here.
- The **Linkage Activated** option is used to switch the linkage on or off.
- The **Enable Event Queue** option activates a mechanism that can reduce system load if needed. This simple example does not require this functionality.
- The **Behavior Upon Error Feedback** field specifies if the linkage is deactivated, marked, or unchanged if errors occur within the event receiver.
- The **Receiver Status** specifies the current status of the linkage.

> ✓ ***Note!***
>
> *You may be wondering, "What if we can't locate a business object event to link to for a given scenario?" There are a few workarounds. First, you should check if you can use a customer exit or a similar concept instead, such as Business Add-Ins (BAdIs), Open FI, etc. These concepts all allow you to add new functionality without modifying SAP-provided development objects.*
>
> *Alternatively, you can also implement a custom event and register the event in the Business Object Repository. While this involves making a system modification, it's arguably a small one with a result that justifies some additional effort in upgrade projects.*

Now that we've bound the service notification event to the event receiver function module, we next need to implement the function module.

Writing the Event Receiver Function Module

The event receiver function module interface must be the same as the interface of function module SWW_WI_CREATE_VIA_EVENT to work properly, so for convenience, we can just copy this function module into our own example function module, Z_RECORD_INSERT. In the importing parameters, the values for the event name (CREATED), the receiver type (CUST_RECORD), the object type (BUS2080), and the automatically generated object key — i.e., the number of the notification — will be passed to our function module during runtime. The other parameters are of no relevance to our implementation.

The implementation consists of three major steps:

1. Determine the customer number associated with the service notification.

2. Find the record for the customer.

3. Insert the service notification into the record.

Figure 21 lists the complete code for the receiver function module. Let's take a closer look at how the code executes the implementation steps.

Step 1: Determine the Customer Number Associated with the Service Notification

For the first step (lines 31-43), we make use of a BAPI of the service notification business object type, `BAPI_SERVNOT_GET_DETAIL`. This BAPI returns, among other data, the service notification's header data for a given notification number, which includes the customer number in field `CUST_NO` of structure `BAPI2080_NOTHDRE`.

Step 2: Find the Record for the Customer

In the second step (lines 45-67), we use `BAPI_RECORD_GETLIST` of the record object type to find the record for which the attribute `ZRM_CUSTOMER` matches the value determined in the first step. There should only be one customer record per customer number. The record is identified by the document class and object ID fields.

Figure 21 Code for the Event Receiver Function Module

```
 1 FUNCTION z_record_insert.
 2 *"----------------------------------------------------------------------
 3 *"*"Local interface:
 4 *"  IMPORTING
 5 *"     VALUE(EVENT) LIKE  SWETYPECOU-EVENT
 6 *"     VALUE(RECTYPE) LIKE  SWETYPECOU-RECTYPE
 7 *"     VALUE(OBJTYPE) LIKE  SWETYPECOU-OBJTYPE
 8 *"     VALUE(OBJKEY) LIKE  SWEINSTCOU-OBJKEY
 9 *"     VALUE(EXCEPTIONS_ALLOWED) LIKE  SWEFLAGS-EXC_OK DEFAULT SPACE
10 *"  EXPORTING
11 *"     VALUE(REC_ID) LIKE  SWELOG-RECID
12 *"  TABLES
13 *"      EVENT_CONTAINER STRUCTURE  SWCONT
14 *"  EXCEPTIONS
15 *"      READ_FAILED
16 *"      CREATE_FAILED
17 *"----------------------------------------------------------------------
18
19
20   DATA: ls_notif_header        TYPE bapi2080_nothdre,
21         lt_property_selection  TYPE TABLE OF bapipropqy,
22         ls_property_selection  TYPE bapipropqy,
23         lt_resulting_list      TYPE TABLE OF bapidoctab,
```

(continued on next page)

Figure 21 (continued)

```
24         ls_cust_record_id      TYPE bapidoctab,
25         lv_anchor              TYPE bapisrmrec-anchor,
26         lt_element_sp_poid     TYPE TABLE OF bapiproptb,
27         ls_element_sp_poid     TYPE bapiproptb,
28         ls_return              TYPE bapiret2,
29         lv_element_description TYPE bapiedescr.
30
31 *** 1. get customer no. from service notification header
32   ls_notif_header-notif_no = objkey.
33
34   CALL FUNCTION 'BAPI_SERVNOT_GET_DETAIL'
35     EXPORTING
36       number      = ls_notif_header-notif_no
37     IMPORTING
38       notifheader = ls_notif_header.
39
40 * build display name of notification as element in the record
41   CONCATENATE ls_notif_header-notif_no ls_notif_header-short_text
42               INTO lv_element_description
43               SEPARATED BY space.
44
45 *** 2. find customer record
46 * define search criteria
47   ls_property_selection-propname = 'ZRM_CUSTOMER'.
48   ls_property_selection-option = 'EQ'.
49   ls_property_selection-sign = 'I'.
50   ls_property_selection-propval_lo = ls_notif_header-cust_no.
51   APPEND ls_property_selection TO lt_property_selection.
52
53 * perform search
54   CALL FUNCTION 'BAPI_RECORD_GETLIST'
55     EXPORTING
56       rms_id             = 'ZSRM_DEMO_RMS_ID'
57       sps_id             = 'ZSRM_SPS_CUSTOMER_RECORD'
58     IMPORTING
59       return             = ls_return
60     TABLES
61       property_selection = lt_property_selection
62       resulting_list     = lt_resulting_list.
63
64   READ TABLE lt_resulting_list INTO ls_cust_record_id  INDEX 1.
```

Step 3: Insert the Notification into the Record

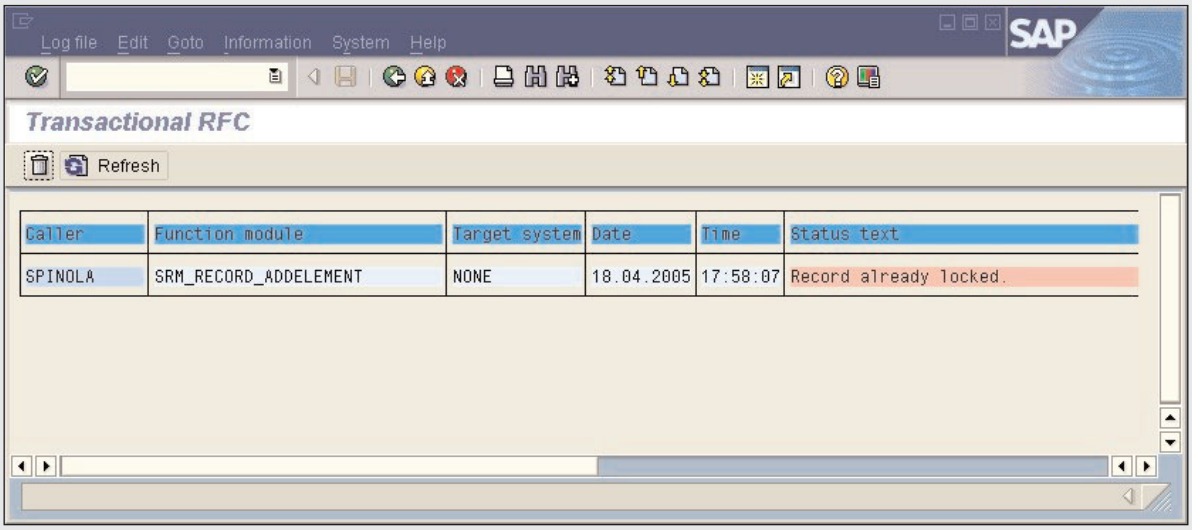
The third step (lines 69-103) is the one that requires the most thought. We want to add a new element to the record. Earlier we mentioned that this is easily accomplished by calling the BAPI `BAPI_RECORD_ADDELEMENT`. There is one issue

Figure 21 (continued)

```
  IF sy-subrc <> 0.
*   ...error: no customer record exists for this customer
  ENDIF.

*** 3. add new element to record
* set sp poid of BOR object
  ls_element_sp_poid-name = 'BOR_OBJECT_TYPE'.
  ls_element_sp_poid-value = objtype.
  APPEND ls_element_sp_poid TO lt_element_sp_poid.

  ls_element_sp_poid-name = 'BOR_OBJECT_ID'.
  ls_element_sp_poid-value = objkey.
  APPEND ls_element_sp_poid TO lt_element_sp_poid.

*
  CALL FUNCTION 'SRM_RECORD_ADDELEMENT'
    IN BACKGROUND TASK
    EXPORTING
      objectid              = ls_cust_record_id-objectid
      documentclass         = ls_cust_record_id-docclass
      sps_id                = 'ZSRM_SPS_BO_SERV_NOTIFIC'
      anchor                = 'BUS2080'
      description           = lv_element_description
    IMPORTING
      return                = ls_return
    TABLES
      element_sp_poid       = lt_element_sp_poid
    EXCEPTIONS
      anchor_not_found      = 1
      not_authorized        = 2
      parameter_error       = 3
      container_not_found   = 4
      container_is_locked   = 5
      max_number_of_elements = 6
      poid_is_wrong         = 7
      internal_error        = 8
      OTHERS                = 9.

  COMMIT WORK.

ENDFUNCTION.
```

we need to address that threatens the data integrity of our records, however: If the record is locked for editing by another user, the BAPI will be unable to insert the notification into the record. We can tell when this happens because `BAPI_RECORD_ADDELEMENT` will return a "record locked" error message in its `RETURN` exporting parameter. Usually, for this kind of error, you would retry the insertion at a later point

Figure 22 An Error Displayed in the tRFC Monitor



in time, which would require a fair amount of additional coding.

Fortunately, there is an easier way. SAP Basis supports a flavor of RFC called *transactional* RFC (tRFC) that automatically logs failed function calls for subsequent execution by an administrator. Function modules called as tRFCs are registered in a general database table for asynchronous execution. After `COMMIT WORK`, these function modules are executed in a separate work process. If an error occurs, all changes are rolled back and the function module can be restarted at a later point in time by an administrator using transaction SM58 (the tRFC Monitor). At any time, an administrator can use transaction SM58 to look up function modules that are waiting to be executed. The monitor shows the status of the function module call and, if in error status, the exception that has been raised (**Figure 22** shows an example). After resolution of the error, the call can be restarted by selecting Edit → Execute LUW from the tRFC Monitor application menu.

> ✓ ***Note!***
>
> *tRFC technology was designed to enable developers to build applications that need to commit data across multiple systems either all at once or not at all (e.g., a bank transaction that posts a debit to one system and a credit to another).*

> **Caution!**
>
> *To use the tRFC technique, a function module has to be RFC-enabled and it should return its errors as function module exceptions instead of in an exporting parameter such as RETURN (which would get lost because of the asynchronous call). SAP Records Management offers such function modules in addition to the BAPIs. In our case, we need the function module SRM_RECORD_ADDELEMENT.*

> **Note!**
>
> *The tRFC technique is still not an optimal way of handling the errors that can occur during background processing for record updates — there is no automated mechanism to restart the failed calls. Because of this, in an integration project with a large US customer, the SAP Records Management development team developed a tool that handles the calls between the application (in that case, SAP Supplier Relationship Management) and SAP Records Management in a more sophisticated way: A scheduled job picks up function calls that have been registered periodically for execution and repeats the execution after a configurable amount of time, and depending on the type of error that has occurred. SAP plans to make this tool available in the standard delivery of SAP Records Management (check http://service.sap.com/recordsmanagement for the latest news).*

Programmatically, invoking SAP function modules via tRFC is easy: just add `IN BACKGROUND TASK` to the `CALL FUNCTION` statement after the function module name.

For example, in the code listing for our event receiver function module (lines 80-81 in Figure 21), we've called the function module `SRM_RECORD_ADDELEMENT` as follows :

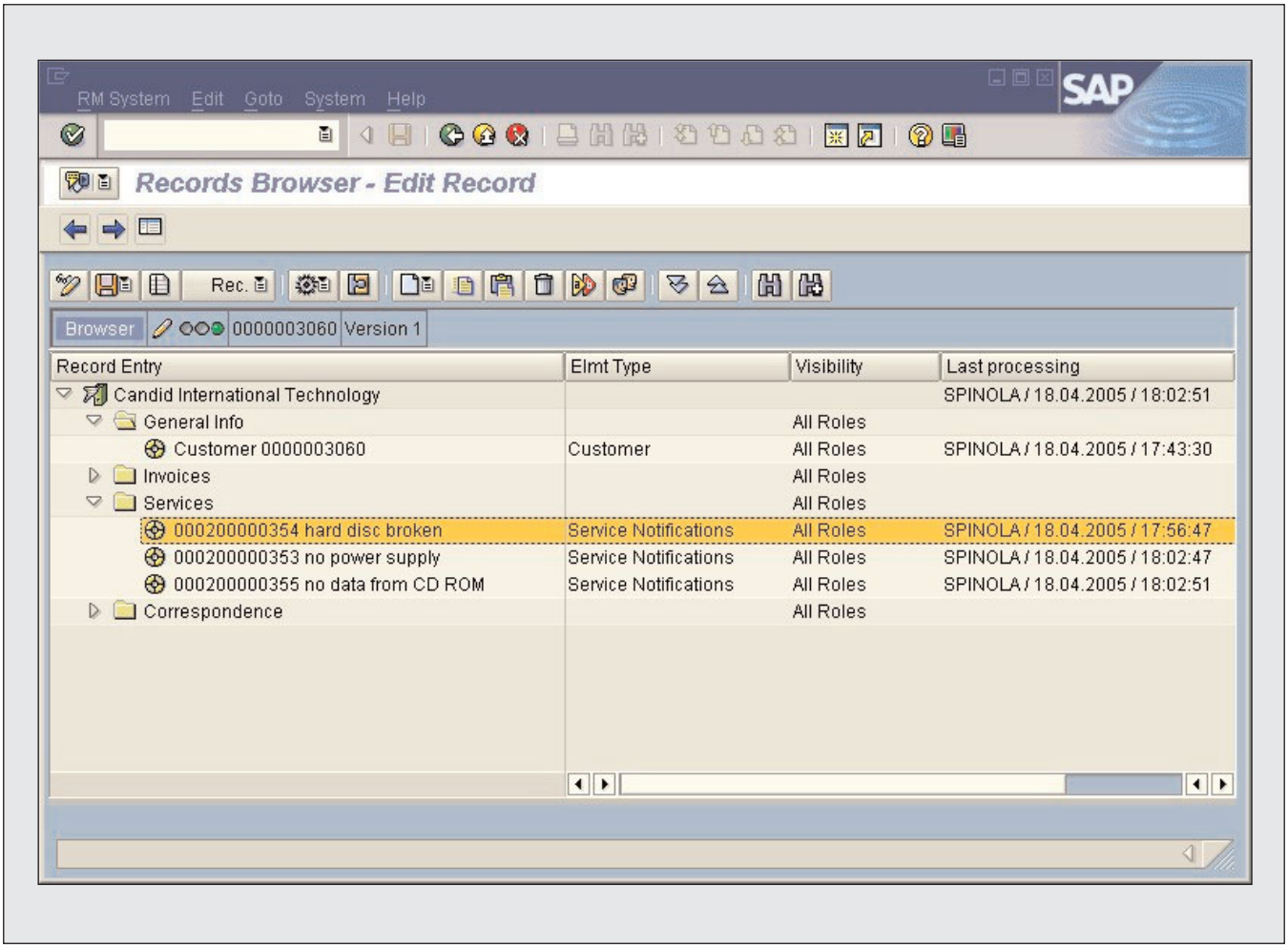
```
CALL FUNCTION 'SRM_RECORD_ADDELEMENT'
    IN BACKGROUND TASK
```

Let's now have a closer look at the `SRM_RECORD_ADDELEMENT` interface. The import parameters `OBJECTID` (line 83) and `DOCCLASS` (line 84) serve to identify the record. `SPS_ID` (line 85) is the element type of the element to be inserted into the record and `ANCHOR` (line 86) denotes the position at which it is inserted (as defined in the record model). `DESCRIPTION` (line 87) is the display text that the element will have in the record. The table parameter `ELEMENT_SP_POID` (line 91) is a list of name-value pairs of the `SP-POID` parameters of the element to be inserted — i.e., the parameters that identify an element of a service provider. For the BOR service provider, the `SP-POID` parameters are `BOR_OBJECT_TYPE` and `BOR_OBJECT_ID`.

The Completed Example

With the final piece of the puzzle — the function module — in place, the example is complete. When a new service notification is posted in the SAP system, a link to the new notification is automatically added to the customer record. The result is displayed

Figure 23 Service Notifications Added to the Record



in **Figure 23**, which shows service notifications inserted into the customer record at the defined location. Double-clicking on one of the service notification items in the record displays the details of the service notification (**Figure 24**).

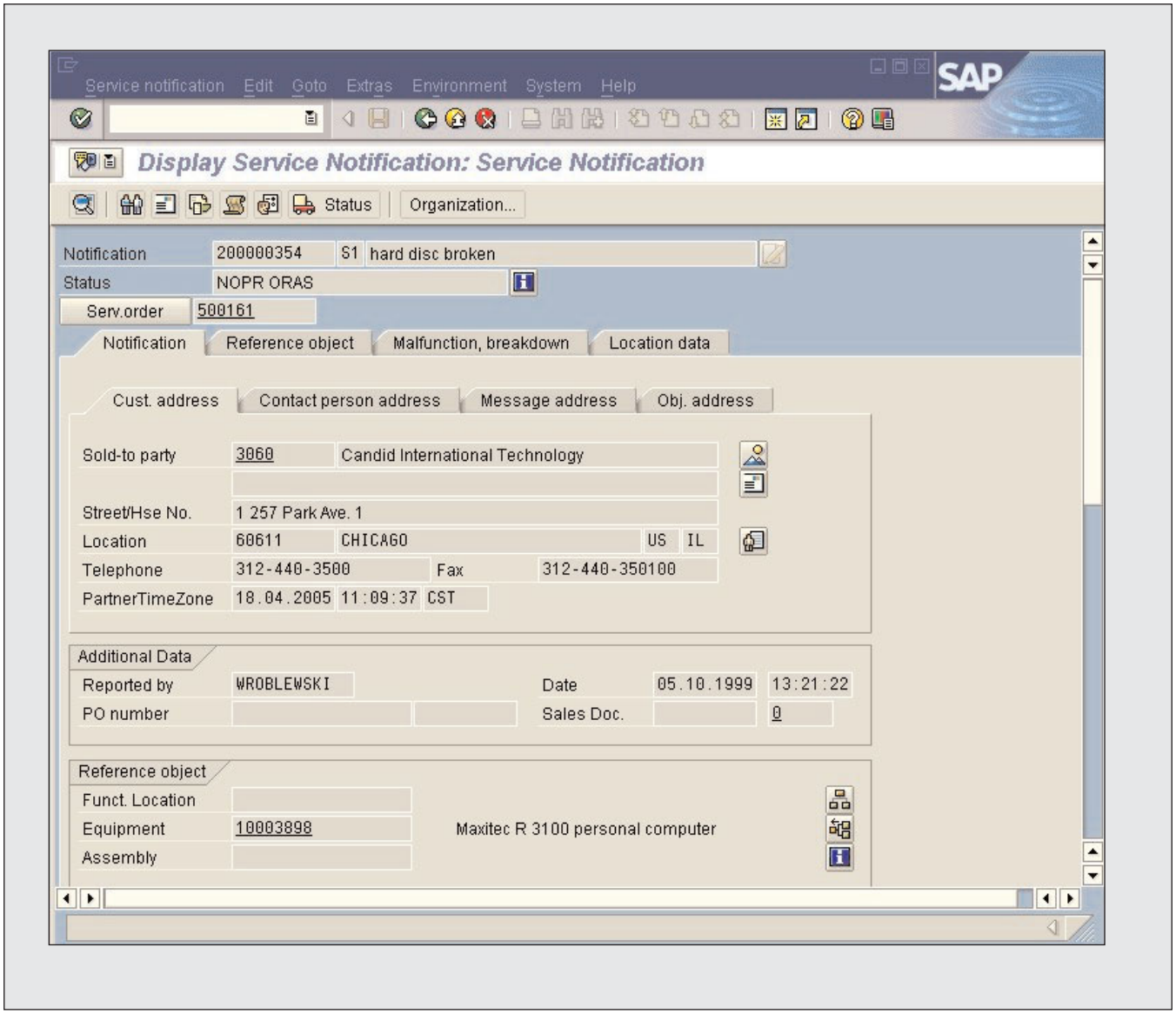
Using the Example

While you can productively use the solution presented in this article with ease, there are a few scenarios that could occur that you might be wondering how to handle — after all, users could potentially grow frustrated if the record includes links to deleted notifications, or if the service notifications are assigned to the wrong customer record. Here are a few possible scenarios and ways to easily address them:

- ☑ **Data, except for the customer number, in the service notification changes:** This is a potential problem that our example application addresses proactively. Because the customer record holds only a reference to the notification, not the notification itself, changes are inherently reflected.

- ☑ **The customer number in the service notification changes (e.g., because an agent accidentally assigned the wrong number to the notification):** Because this information is used to

Figure 24 Details of the Selected Service Notification Item



assign the notification to a record, the reference to the notification would have to be removed from one customer record and added to another customer record. Depending on the volume, you could do this manually, or you could set up an automatic update mechanism similar to the one described in this article.

☑ **The service notification is deleted:** This would require you to also delete the reference to the notification from the record. Again, depending on the volume, this can be done manually or you can use the method described in this article to delete the reference automatically.

Conclusion

This article has shown you how to use event linkage

to automatically update records in SAP Records Management when business events occur in an SAP system (e.g., when records are created, changed, or deleted). We encourage you to try your hand at this technique as it will greatly reduce the burden on SAP Records Management users and increase the ROI for your implementation.

As keeping references up to date is a universal requirement, SAP has developed a more permanent mechanism — as part of SAP Web Application Server 7.0 — that allows you to display record content dynamically. In this solution, the user can see folders and elements within a record that are not actually part of the record persistence. Nevertheless, the techniques presented in this article — event linkage and tRFC in particular — will remain useful to you throughout your SAP career.

Joachim Becker is Product Manager of SAP Records Management. He has a degree in mathematics from the University of Heidelberg, Germany. In his degree dissertation, Joachim discussed wavelet algorithms, which are used to analyze digital images, for example. He joined SAP in 1995 as a developer in the Financials area, and also worked as a trainer and consultant. Since 1999, Joachim has assumed various product management responsibilities in SAP's Technology Development department. He is an expert in Business Process Technology in general, and has focused on the areas of Workflow, Document Management, and Communication. Currently, Joachim works as a Process Architect on the definition of the newly announced Business Process Platform. He is also one of the authors of the book "SAP Records Management," which was published by SAP PRESS in March 2005 (currently available in German only). He may be contacted at joachim.becker@sap.com.

Ulrich Spinola has long-term international project and project management experience in the areas of SAP Records Management and Workflow. He is the architect of the process management components of SAP Records Management. In addition to his work in development and consulting, he speaks at conferences and publishes articles about SAP Records Management. He is also a coauthor of the book "SAP Records Management," published by SAP PRESS in March 2005. So far, only a German edition is available. Ulrich may be contacted at ulrich.spinola@sap.com.

