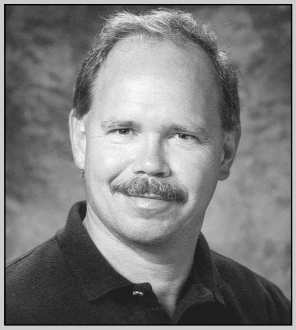
Manage Documents with More Efficiency and Less Effort Using the SAP Business Document Service and SAP Desktop Office Integration

Philip Bremner



Philip Bremner is a contract programmer at Visual SAP, where he specializes in SAP DOI, BDS, ALV Grid Control, and Control Framework programming. He has 10 years of programming experience in ABAP, Visual C++, and Visual Basic, and is a Microsoft Certified Professional holding nine Microsoft certifications.

(complete bio appears on page 62)

I don't think anyone would argue the importance of managing organizational documents effectively. All too often, however, companies rely on a shared network drive or database to serve as a document management solution. While such a solution will do the job to some extent, it obviously has some shortfalls — there is no easy way to guarantee that files are faithfully copied to the drive or database, there is no efficient or reliable way to track and coordinate changes to documents, and searching for documents can be cumbersome and time-consuming. Things can quickly become even more complicated once someone forgets to copy a critical document to the database or drive, not to mention when people inevitably start emailing around various copies of documents or saving them to their local hard drives, making it impossible to tell which version is the most recent, or even where that version is, for that matter.

Making it easy to store and access documents in a coordinated and centralized way is an effort that will deliver very real rewards for your organization. Ideally, you want to link your SAP environment with an integrated document management solution that allows users to check documents in and out of a single, central repository, easily edit and update documents, and create and retrieve particular versions of a document, all from within the same screen, so they don't have to keep switching between different applications.

While you can create a homegrown solution or purchase an off-the-shelf product like Cypress or SAPERION to address these needs, if you are running SAP Release 4.6A or higher, you already have a robust and flexible document management solution at your fingertips. By adding only a few lines of code to a simple custom ABAP program, you will be able to:

- Centrally store and manage any type of electronic content — e.g., Microsoft Office files, images, programs, and web pages — as "documents"
- Link documents to SAP R/3 business objects, like customers and materials, so that end users can access them from within any compatible R/3 applications[1]
- Classify documents using tailored sets of metadata[2]
- Coordinate changes and avoid collisions through document check-in/check-out functionality
- Enable users to search for documents via metadata or indexed content
- Store and manage multiple versions and variants[3] of documents
- Allow users to instantly retrieve the most up-to-date document version available in a given language or format through SAP's "dynamic context resolution" feature
- Ensure that references within and between documents — e.g., the link between a document and its template, which itself may have many versions and variants — remain current

This article introduces programmers and functional users to SAP's document management infrastructure — the SAP Knowledge Provider — and, using a downloadable demo program (available at **www.SAPpro.com**) as an example, goes beyond the existing documentation to show you how to leverage this infrastructure to create a comprehensive storage solution for your own environment. You will learn how to enable Knowledge Provider capabilities using its Business Document Service (BDS) interface, and, through standard SAP Desktop Office Integration (DOI) functionality, how to enable users to create, retrieve, and update documents and versions of documents, all from within a single, custom SAPGUI screen. While the discussions in this article are limited to enabling these particular Knowledge Provider functionalities, the lessons learned can be easily applied to enabling additional functionalities.

Before jumping right into the programming details, however, it will be helpful to review some of the key concepts involved in the Knowledge Provider document management infrastructure, and how the BDS and DOI functionalities enable and enhance it.

A Complete Document Management Solution — The Knowledge Provider Infrastructure, the BDS Interface, and DOI Functionality

Figure 1 is a graphical overview of the components involved in SAP document management, which at its core consists of the SAP Knowledge Provider, a generic document management infrastructure within the Basis system, and various repositories. The Knowledge Provider's services provide its generic content management, indexing, and document retrieval functions. Repositories (such as an R/3 database, a content server,[4] or some kind of external storage system like Open Text Livelink, for example) house the physical documents.

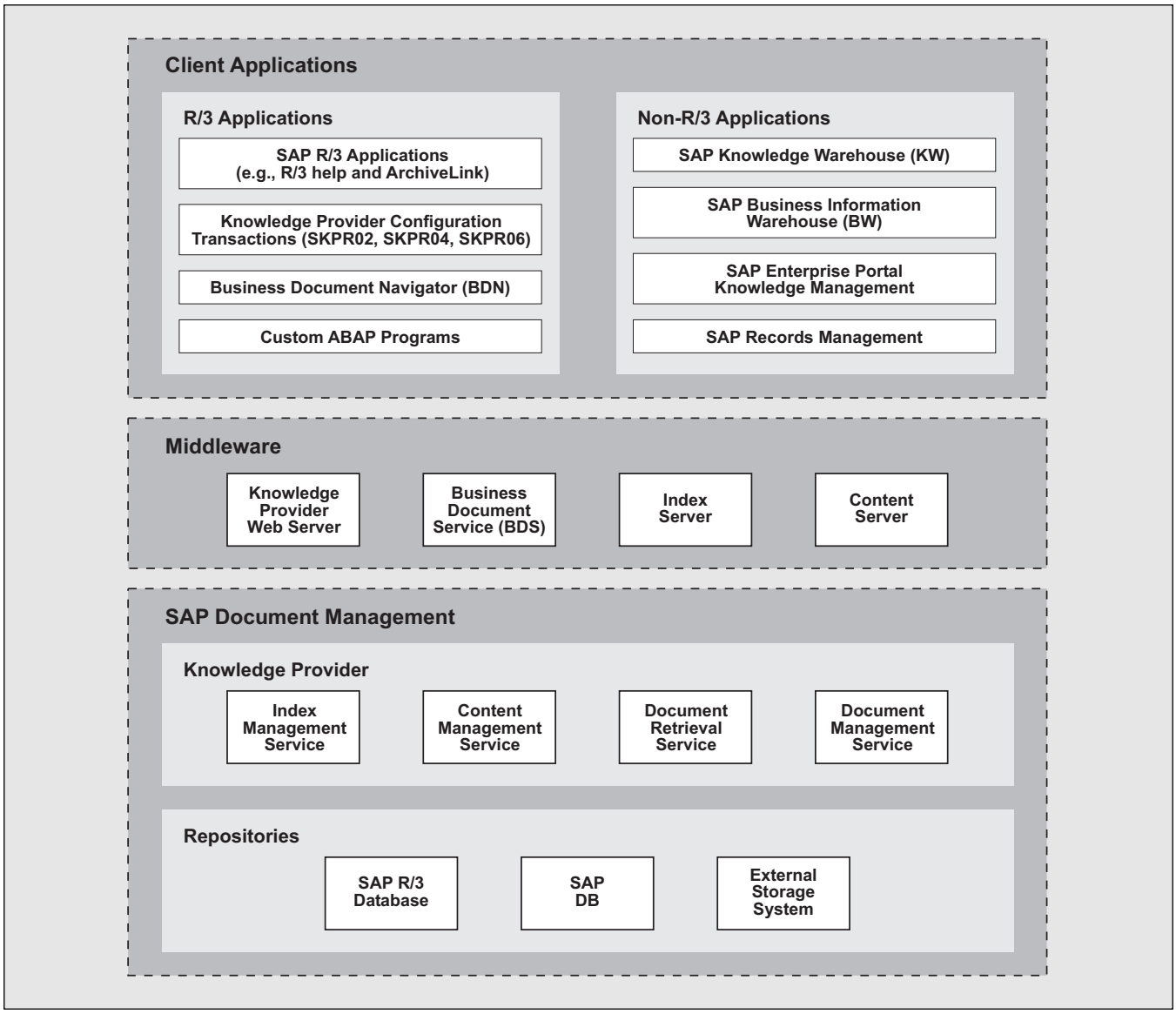
[1] Many important R/3 applications offer convenient, context-sensitive access to documents (e.g., help text and training materials) or support for document-based business processes via SAP Business Workflow.

[2] Transaction *DMWB* (the Document Modeling Workbench) provides an interface to the Knowledge Provider "content model," which enables you to define metadata fields specific to a type of document — for example, "photographer" and "player" fields for a document containing sports images, or "topic" and "journalist" fields for a newspaper article.

[3] For example, a French copy of an English original, or a PDF version of a Microsoft Word DOC file.

[4] The link between documents and their storage repositories is stored in table BDS_LOCL. The SAP Content Server is distributed with SAP R/3 Release 4.6A+, but it is installed separately (usually on its own NT server).

Figure 1 An Overview of the SAP Document Management Architecture



✓ ***Note!***

*SAP R/3 also contains a component called the document management system (DMS), which provides flexible status management and document structuring functionality for engineering and logistics processes. Introduced in Release 3.0, the DMS can be incorporated into the Knowledge Provider infrastructure as of Release 4.6. To configure the DMS to use the Knowledge Provider infrastructure, in Customizing select Document management → General data → Settings for storage systems, and then activate the "Storage via Knowledge Provider" indicator. For more information on the DMS, go to the 4.6C online help (**http://help.sap.com**) and navigate to SAP Library → Cross-Application Components → Document Management.*

A Brief Overview of SAP Desktop Office Integration (DOI)

SAP Desktop Office Integration is an object-oriented ABAP API. Adding DOI classes to an ABAP program makes it possible to run desktop applications that are compatible with Microsoft's OLE (object linking and embedding) technology — like Microsoft Office and Lotus SmartSuite applications, for example — from within an R/3 screen, instead of having to launch the applications separately. SAP DOI also integrates desktop applications with R/3 data processing, so that data such as internal tables, variables, and events can be transmitted from the ABAP program to the application and vice versa.

The desktop application runs within the R/3 screen via an SAP control. SAP controls, which are automatically installed with the SAPGUI, are prebuilt software components used to design user interfaces. Some of the most commonly used controls are the SAP tree control (used to display data in a tree structure in an R/3 screen), the HTML viewer control (used to display HTML pages and graphics in R/3 transactions), the ALV grid control (used to display tables in an R/3 screen), and the SAP custom container control (one of several types of container controls used to "contain" the controls in use). The custom container control is linked to an area of the SAPGUI screen via the Screen Painter (transaction SE51). In a DOI scenario, a custom container control contains the desktop application. The SAP Control Framework resides on the application server and provides the infrastructure for the communication between the ABAP program running on the backend and the controls running on the frontend. Method calls from the frontend execute the corresponding backend OCX (custom OLE controls, also known as ActiveX controls) methods that control the desktop application.

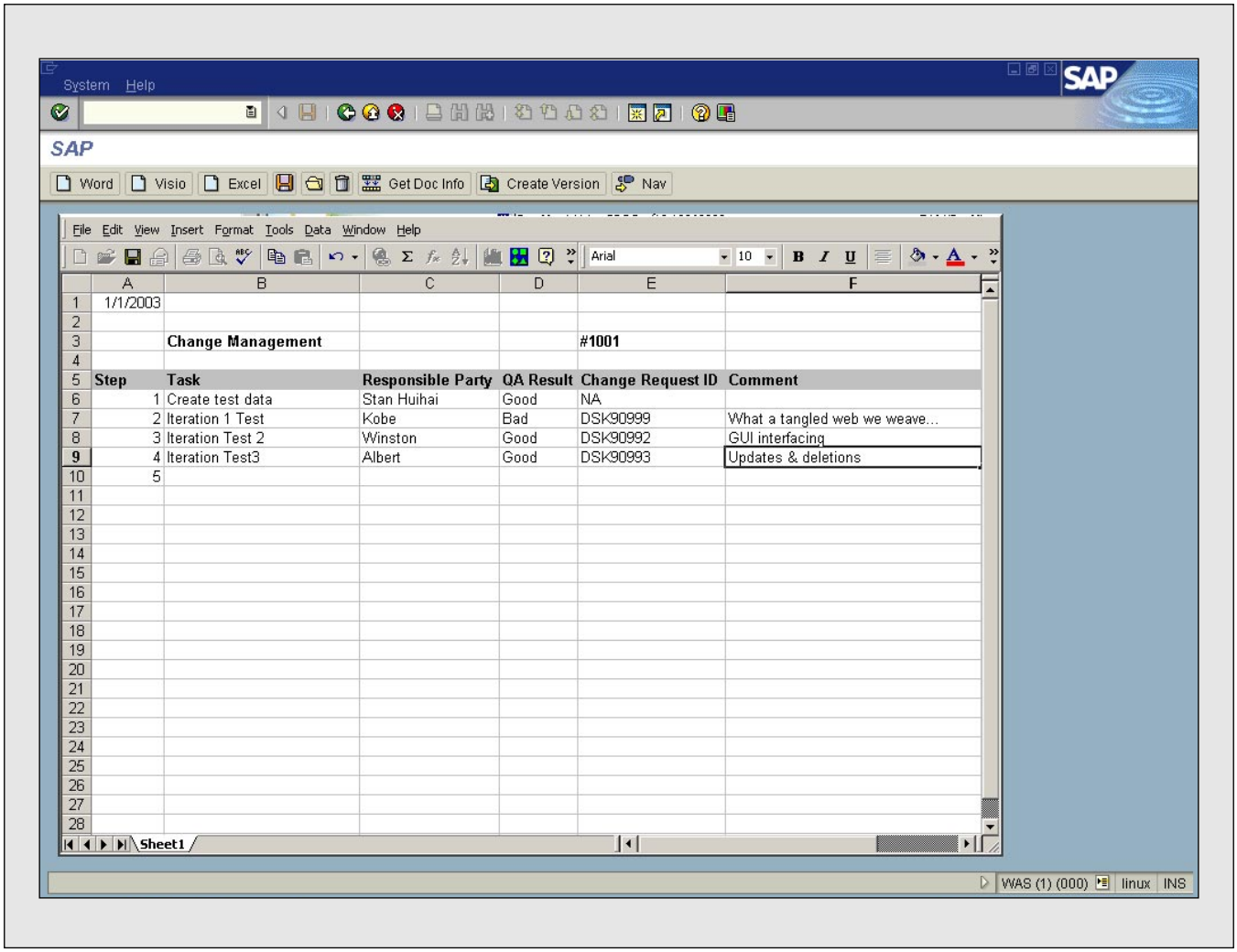
More information on the SAP Control Framework and DOI is available in the 4.6C online help (**http://help.sap.com**) under *SAP Library → Basis Components → Controls and Control Framework → Desktop Office Integration*. See also the articles cited in the "Resources" sidebar on page 62.

The Knowledge Provider itself has no graphical user interface, but instead relies on business applications like R/3 transactions or custom ABAP programs to deliver its content and functionality to end users, which it accesses via an interface like the Business Document Service (BDS) middleware component.

✓ ***Note!***

*While this article focuses on the BDS interface, it is also possible to use the Knowledge Provider Web Server, an index server, or a content server (either the SAP Content Server or a third-party product). I prefer the BDS because of its simplicity and the ease with which it integrates with DOI. For details on using other interfaces, visit the 4.6C online help (**http://help.sap.com**) and navigate to Implementation Guide for R/3 Customizing (IMG) → R/3 Basis Customizing → Basis Services → SAP Knowledge Provider.*

Figure 2 A Microsoft Excel Document Displayed Within an SAPGUI Screen

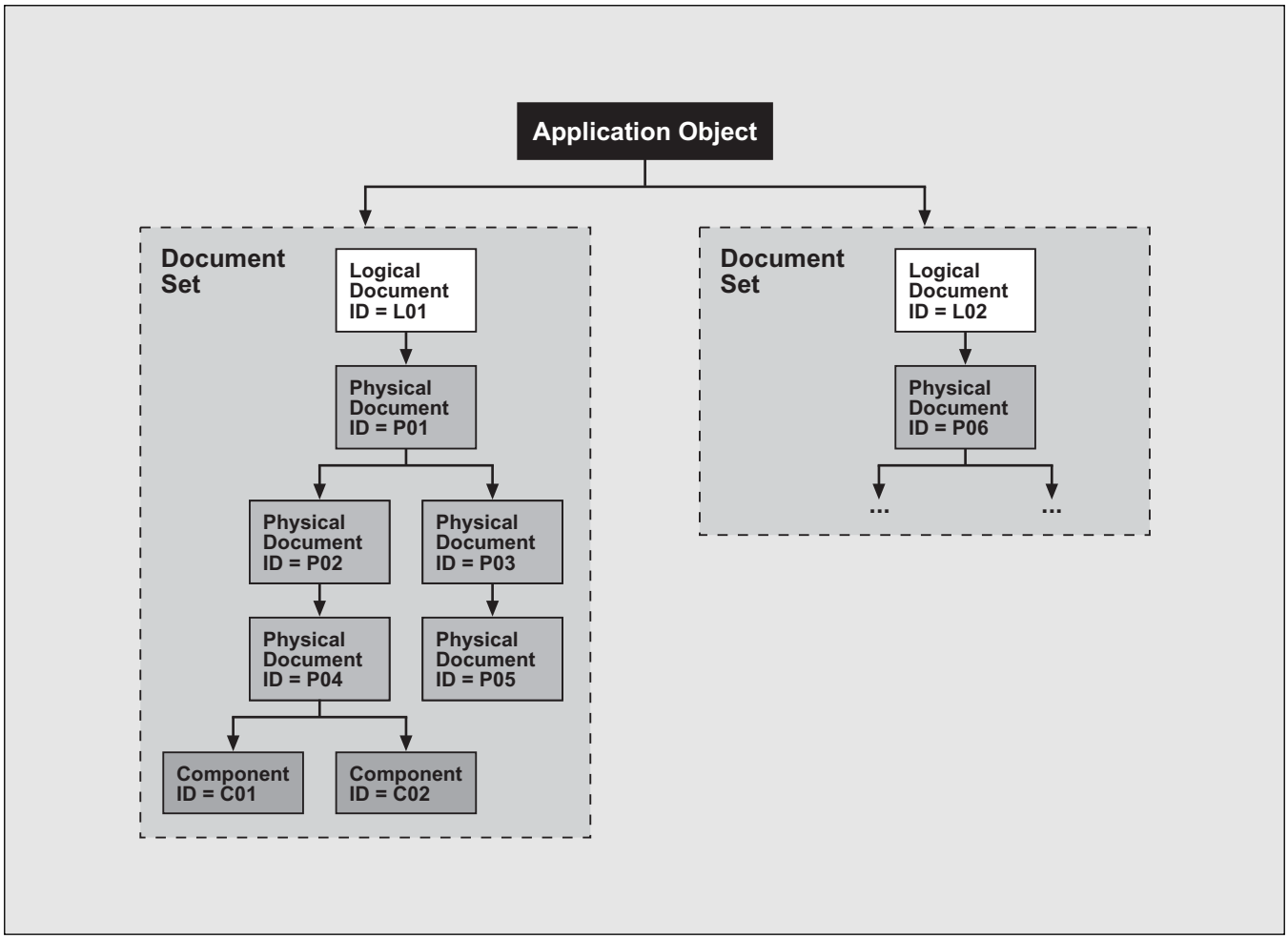


While its name might imply a complex configuration, the BDS is actually an easy-to-use ABAP class[5] (CL_BDS_DOCUMENT_SET) whose methods you can call from an ABAP program to expose Knowledge Provider functions like creating, retrieving, or updating documents, versions, and variants.

With the BDS functionality, however, users still have to launch an external desktop application to view or edit a document and then manually upload documents to the designated repository. It would be much more efficient for users to manage documents within an SAPGUI screen and for the check-in process to be automated. Standard SAP Desktop Office Integration (DOI) classes can be used in combination with the BDS class to load a document into a pane within an SAPGUI screen and automatically upload it to a repository upon saving (for a refresher on key DOI concepts, see the sidebar on the previous page). You can even save users additional time by preloading a blank document or template into the application pane on the screen. **Figure 2** shows an example of a

[5] Coding with the BDS and DOI classes requires some familiarity with ABAP Objects. If you are new to ABAP Objects, don't worry — you should be able to pick up what you need to know from the examples in this article.

Figure 3 *The Object Model*



Microsoft Excel document displayed within an SAPGUI screen via DOI.

To learn to use the BDS interface effectively, you need to understand some important Knowledge Provider elements, how its object model is organized, and how the objects relate to one another. We'll examine these concepts next.

Understanding the Knowledge Provider Object Model

The following discussion and figures will save you from hours of reading through SAP documentation. They provide an overview of the concepts that underlie the Knowledge Provider and how the key objects within the object model relate to one another.

The most important objects in the Knowledge Provider infrastructure are application objects, logical documents, physical documents, components, and document sets (see **Figure 3**):

- **Application objects** represent either business objects (like customers) or technical objects (like ABAP classes). You assign a document to an application object when you "create"

(i.e., register) a document in the Knowledge Provider database. An object is identified by its class type, class name, and object key, where the class type must be "BO" (a Business Object Repository class), "CL" (an ABAP Objects Class Library class), or "OT" (some other type of class). Application objects represent the highest level in the object model, and indirectly determine the repository in which a document is stored.

> ✓ ***Tip***
>
> *The SAP help doesn't clearly explain how the Knowledge Provider determines where to store a document. Here's how it works: Each document is assigned to an application object via its class type, class name, and object key. Using the class type and class name, each application object is assigned to a physical document class in configuration table BDS_LOCL, and then each physical document class is assigned to a storage category hard-coded in its private STORAGE_CATEGORY attribute via transaction DMWB (the Document Modeling Workbench). Finally, each storage category is mapped to a repository in a configuration table. The implication is that all documents of a particular type are always placed in the same kind of storage (e.g., in R/3 or in a content server). All of these settings are defined by the particular Knowledge Provider template — i.e., the content model defined in transaction DMWB — that is in use.*

- **Logical documents** represent the intent (function) of the document (e.g., "this is a material master training manual"). The logical document concept provides a way to refer to a document without having to link directly to any particular *instance* of that document (which may subsequently get updated, translated, or deleted). Logical documents provide a link to the template used to store documents — the Knowledge Provider content model (more on this in a moment).

- **Physical documents** represent actual instances of a logical document that a user has checked into the Knowledge Provider. Despite the name, physical documents do *not* represent actual files. Instead, physical documents are made up of one or more document "components," which represent the actual files (more on these in the next bullet item). This abstraction accommodates cases where multiple files collectively form a single document. Think of a book composed of 10 chapters, each of which is a separate file — each chapter represents a component, and the book represents the physical document.

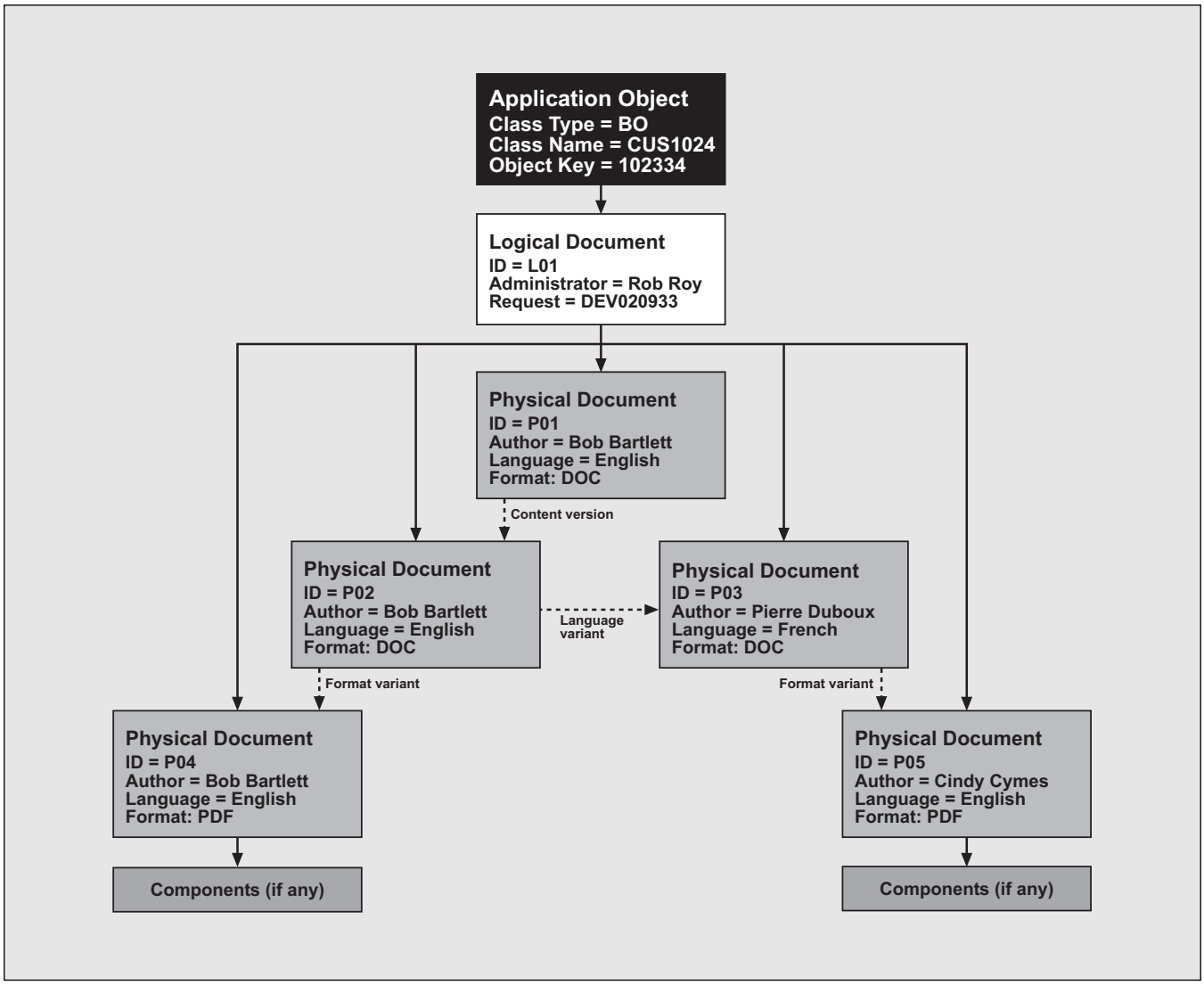
- **Components** are divisions of physical documents. While they represent actual files, it is more useful to think of them as "stubs" for files rather than the files themselves, which may be physically stored in an external system (refer to the previous bullet item).

- **Document sets** enable you to refer to a group of physical documents and their components collectively. Some BDS methods can process (i.e., retrieve, delete, etc.) more than one document at a time — you specify how many or how few to process using the method call's import parameters.

The configuration of application objects, logical documents, physical documents, and components of an application, including their attributes and permissible relationships, is specified in a "content model," which is defined via the Document Modeling Workbench (transaction *DMWB*) as of Release 4.6D or function group SDCL in earlier releases.[6] An

[6] Note that I provide only a high-level view of the Knowledge Provider content model. The details are beyond the scope of this article. For more information, go to the 4.6C online help (**http://help.sap.com**) and navigate to *SAP Library → Basis Components → Basis Services/ Communication Interfaces → SAP Knowledge Provider → Document Management Service → Content Models*.

Figure 4 *The Content Model*



example Knowledge Provider content model is shown in graphical form in **Figure 4** (note that Figure 4 is a detailed look at the left-hand document set in Figure 3). The key elements of the content model are attributes and relationships:

- **Attributes** store key administrative or user-defined document metadata. You assign "global" attributes (i.e., those not specific to a particular physical document) at the logical document level and instance-specific attributes at the physical document level. For example, the logical document in Figure 4 has the global attributes *Administrator* and *Request*, which are common to all of its physical documents. In contrast, attributes *Author*, *Language*, and *Format* vary from instance to instance, so they are defined at the physical document level.

- **Relationships** tie documents together. The Knowledge Provider supports many different types of relationships, including versions,

Figure 5 Using the DCR Feature to Retrieve Documents

Example Code	Description
`href=…ID=L01, English, DOC`	Returns the newest English version of logical document L01 in DOC format, if available.

`href=…ID=P03`
or
`href=…ID=L01, Version=1, Variant=1`
	Returns physical document instance P03, which is a variant of logical document L01.

variants (a language translation or file format conversion of a document), links between a document and its template, and hyperlinks between documents. The BDS includes methods you can use to define additional relationships and retrieve all documents related to a specific document.

Last, but far from least, take a look at **Figure 5**. Using the example objects in Figure 4, Figure 5 illustrates a powerful Knowledge Provider feature called "dynamic context resolution" (DCR) for retrieving documents. DCR is a mechanism by which the system tries to automatically locate the physical document (or component) that is the closest match for the set of specified attribute values. In practice, DCR simplifies BDS programs — you can simply "ask" the system for what you want (a particular document ID, language, format, etc.) and let the system worry about finding it.[7]

> ✓ ***Note!***
>
> *More details on the Knowledge Provider can be found in the 4.6C online help (**http://help.sap.com**) under SAP Library → Basis Components → Basis Services/Communication Interfaces → SAP Knowledge Provider.*

[7] A detailed discussion of dynamic context resolution is beyond the scope of this article. For more information, see the 4.6C online help (**http://help.sap.com**) under *SAP Library → Basis Components → Basis Services/Communication Interfaces → SAP Knowledge Provider → Document Management Service → Context Resolution*.

Now that you've got a solid understanding of the basic Knowledge Provider concepts and where the BDS and DOI functionalities fit in, we're ready to dive into the heart of the matter — adding this document management functionality to an ABAP program. So let's get started!

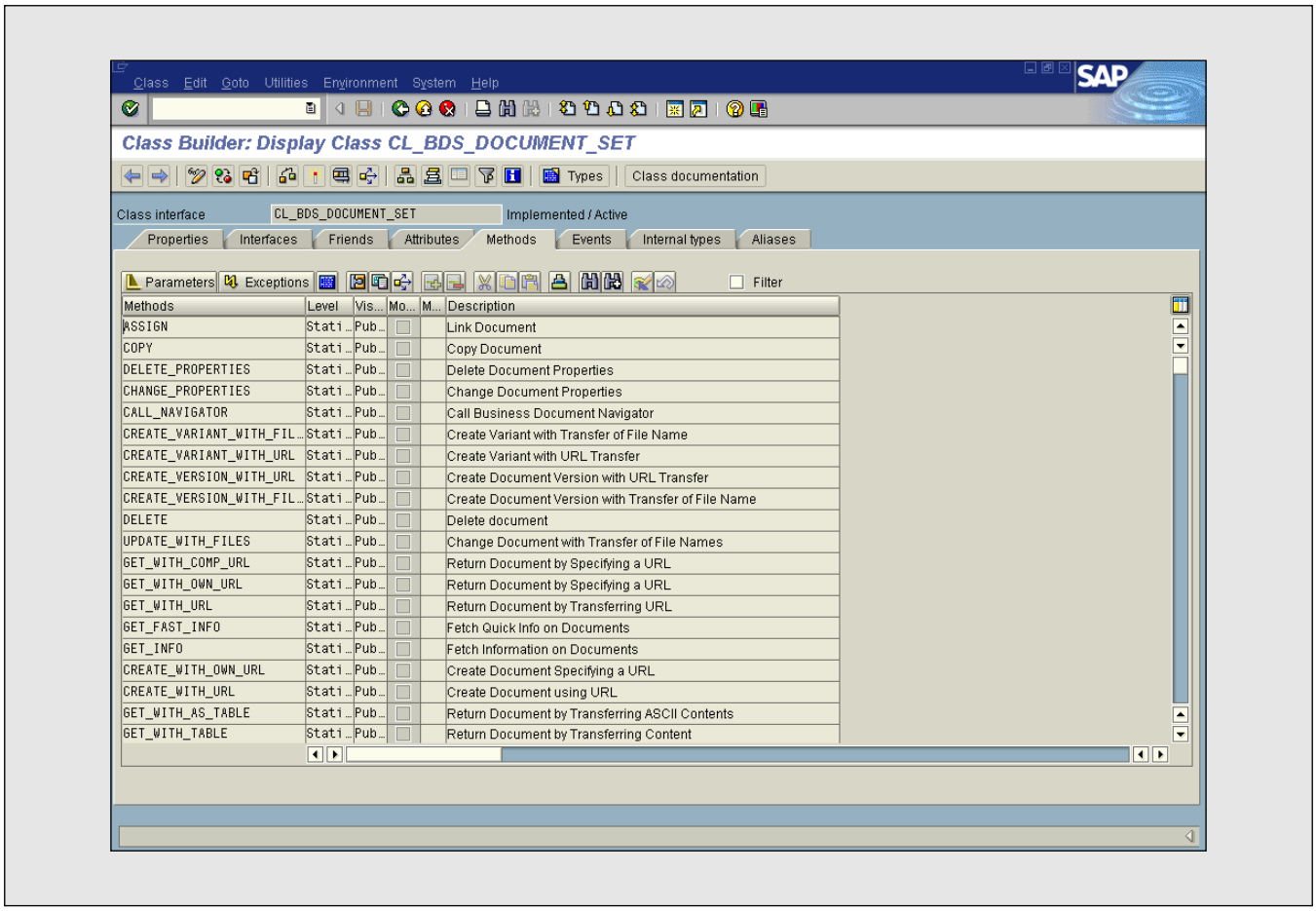
Programming with the Business Document Service (BDS) Interface

The Business Document Service (BDS) interface is an ABAP API for the document management functionality offered by the SAP Knowledge Provider. It is simply a public class — CL_BDS_DOCUMENT_SET — that provides a set of methods with which you can create, retrieve, edit, and delete documents, and versions and variants of documents. Using a demo program, I will show you in detail how to use this class and its methods to enable this functionality, and how to incorporate some Desktop Office Integration (DOI) classes that will allow users to retrieve, edit, and update Word, Excel, and Visio documents directly from within an SAPGUI screen.

I've often found that the best way to get to know this or any unfamiliar class is to review its methods and interfaces[8] in the SAP Class Builder (transaction

[8] An interface is a named collection of class method definitions (without implementations).

Figure 6 BDS Class CL_BDS_DOCUMENT_SET in the Class Builder (SE24)



SE24). **Figure 6** shows the methods available in the CL_BDS_DOCUMENT_SET class. To see the parameters associated with a method, place your cursor on the method and click on the *Parameters* button at the top of the *Methods* tab.

> ✓ ***Tip***
>
> *You can try out each method listed in the Class Builder by switching to "test" mode (press F8 or click on the icon), highlighting a method, and then clicking on the icon, which will appear to the right of the selected method.*

While the list of methods can be overwhelming at first, there are only a handful you'll use repeatedly. Further, you'll see that several methods come in "sets" — for example, there is a set for creating documents and a set for retrieving documents. The sidebar on the next page provides an overview of the most frequently used BDS methods.

So let's say that we are members of Company X's development team and that we need to document a change request, which for our purposes includes creating a functional design document in Microsoft Word, a technical specification document in Microsoft Visio, and a test plan in Microsoft Excel. In the past, we have always created and stored such documents on a team-shared drive. Unfortunately, over time, the number of documents on the shared drive has grown to an

Frequently Used Methods of the BDS Class CL_BDS_DOCUMENT_SET

The CL_BDS_DOCUMENT_SET methods used most frequently in BDS programming are those for creating, retrieving, updating, and deleting documents, and for creating versions and variants of documents.

Creating Documents

- *create_with_files* — Use this method when you want to check one or more files into the Knowledge Provider. It immediately commissions the SAPGUI to upload the document, stores it in the repository mapped to the application object (e.g., R/3 database table BDS_CONTX, where X is a number), and registers the document in the Knowledge Provider database.
- *create_with_URL* — Use this method when you want to create and save a new document using the DOI methods *create_document* and *save_to_URL*. It will return a strange-looking "URL" (a symbolic string unique to BDS)* that *save_to_URL* will recognize. The *save_to_URL* method physically uploads the working copy of the document from its temporary location on the user's PC to the R/3 application server and passes it to the BDS. The BDS then uses the URL to physically store the document's content in the repository mapped to the application object to which the document has been assigned.

  After saving the document with the *save_to_URL* method, make sure you call the BDS method *confirm_create* to complete the process. The *confirm_create* method inspects the newly uploaded content and updates the document's Knowledge Provider record with critical metadata like file size.

> ***Warning!***
>
> *If you don't call "confirm_create" after "create_with_URL," your document's metadata will be incomplete. You will need to delete it either with the BDS "delete" method or using the BDS's Business Document Navigator (BDN), and then re-create the document. See the sidebar on pages 50-51 for more on the BDN.*

---

* This "URL" appears in the following format:

SAPR3-BDS-<class name>-<class type>-<objectID>-<file name>-<version number>-<variant number>

It is not the type of URL that most people are familiar with — e.g., *http://* or *ftp://* — but is instead a symbolic name that represents a file stored in the BDS system. BDS-specific URLs only make sense to the BDS and DOI methods; they will not work in a web browser.

(continued on next page)

unmanageable size, and it is difficult to keep documents effectively grouped by change request. Switching back and forth between the SAP system and the desktop application, as well as searching for documents using Windows Explorer, is also quite tedious.

We can easily address these problems by creating our own document management program — a small custom ABAP program that includes BDS and DOI functionality and runs in a custom SAPGUI screen. With this application we can now create the new documents we need (using a Microsoft Office window embedded within the SAPGUI screen), store them in an R/3 database, link them to a change request, and subsequently view or edit them within the same SAPGUI screen. And that's not all — we can get this program up and running in two days, and so can you.

(continued from previous page)

- *create_with_own_URL* — Use this method to upload any existing files that are accessible via a traditional URL like *http://*, *file://*, or *ftp://*. Similar to *create_with_files*, it retrieves the file immediately, routes it to the appropriate repository, and registers a physical document in the Knowledge Provider.

- *create_with_table* — Use this method when you want to save a document currently loaded in an internal table. This function is used less frequently than the others.

Retrieving Documents

- *get_with_files* — This method extracts both the content and metadata of one or more documents in the Knowledge Provider. The content is extracted from its repository and materializes as files at the path (or paths) you specify in the "FILES" table parameter through fields "DIRECTORY" and "FILENAME" (the paths are PC paths).

  As you will see in the demo program discussed in the article, you control which documents this and the other *get_with_* methods retrieve by specifying an application object (through the required fields class type, class name, and object key), and optionally the "signatures" of one or more documents (i.e., a document ID, plus a version number or variant ID if you wish to retrieve a single document instance). Most of the time you'll specify both an application object and full document information to retrieve the specific document and version you want.

- *get_with_URL* — Like *create_with_URL*, this method returns a BDS-specific URL that is passed to a DOI method for processing; in this case, DOI method *open_document*. It does not, by itself, retrieve the document content, but instead relies on the DOI method to do this.

- *get_with_table* — This method returns both the content and metadata of one or more documents in table parameters. This function is used less frequently than the others.

> ✓ ***Tip***
>
> *The "get_info" method is an efficient method to use when you only need document metadata.*

Updating Documents

- *update_with_URL/files/table* — Use one of these methods to update existing documents. Note that, technically, these methods allow you to update the metadata or content of any version, no matter

> ✓ ***Note!***
>
> *I've chosen to use an R/3 database to store documents in the example because it promotes centralized management, reduces the risk of data loss (due to database backups), and enables versioning. Depending on where you prefer to store your physical document content, you could also use an HTTP content server or an external storage system, for example. Also note that while the demo does not include details on how to link into the change request transport layer, this capability can be easily added to the program.*

how old. However, you will probably wish to design your programs such that users can only modify the current version (to preserve an accurate version history or audit trail, for example).

Also, like its colleagues, *update_with_URL* simply returns a BDS-style URL that is passed to the DOI method *save_document_to_URL* rather than performing the update itself. You must also call *confirm_update* after completing the save.

Deleting Documents

- *delete* — This method deletes all document instances in the specified document set. The system deletes both the instance's metadata and its corresponding content in the repository.

Creating Versions and Variants of Documents

- *create_version_with_URL/files/table* — Use this method instead of *update_with_URL* when you want to check in an updated instance of a document as a new version. The old instance is preserved and linked to the new one via a version relationship.
- *create_variant_with_URL/files/table* — Use this method to check in a translated instance of a document, or an instance that has been converted to another format (e.g., a Word DOC original converted to PDF or HTML format).

> ✓ ***Note!***
>
> *Once you've created a new version or variant with these methods, you can update or delete them using the standard "update_with_" and "delete" methods — these methods accept a version or variant as input.*

The CL_BDS_DOCUMENT_SET class provides additional methods for querying documents, defining and inspecting relationships between documents, locking and unlocking documents,** copying document sets, and generating the entries needed to transport documents (note that the objects still must be written into a change request in such cases). I'll leave you to research these methods on your own since they are used less frequently and are easy to figure out once you know how to use the core methods.

** The BDS methods *document_enqueue* and *document_dequeue* enable you to prevent user collisions by locking specific document instances (via document ID, version number, and variant ID). To implement a check-in/check-out scheme in your programs, call the enqueue function immediately after retrieving a document for editing. Dequeue the document immediately after inserting the updated version.

Over the course of the rest of this article, we will walk through the following tasks:

- Creating the SAPGUI screen and the ABAP program
- Creating a document
- Updating a saved document
- Searching for a document and testing the method calls
- Retrieving an existing document
- Creating a new version of a document
- Running the full program

Preparing the Demo Environment

Before trying to develop the demo or install the code (available for download at **www.SAPpro.com**), you need to do a few simple things. You'll find detailed instructions in the demo ZIP file.

Setup is minimized by the fact that the Knowledge Provider content model can be used out of the box without any customization of its definitions. By default, the Knowledge Provider stores all documents in R/3 database table BDS_CONT1, and the BDS registers any unfamiliar application object IDs in configuration table BDS_LOCL.

One last thing — be sure to get the assistance, or at least the approval, of your R/3 administrator and/or management before storing documents in the R/3 system. If you'll be using this document management solution productively, come prepared with an estimate of how much storage space you'll need now and how much you are likely to need in the future.

Now would be a good time to download and install the demo program available in the "Download Files" section at **www.SAPpro.com** so you can follow along. The sidebar above outlines some prerequisites to performing these tasks in your own environment.

✓ ***Note!***

The demo created for this article was developed with a 6.10 SAP Mini Basis system running on a SUSE Linux 8.0 operating system. The code is simplified to facilitate the discussions. For clarity, the demo program and the discussions focus only on the code needed to create and manipulate documents with the BDS. By simply adding a few more lines of code, you can easily enhance the demo program. For example, you could activate a menu or screen exit within the Transport Organizer (SE10/SE11) to make the functionality available where it is most likely to be used.

Creating the SAPGUI Screen and the ABAP Program

Before we can start coding the program, we have to create a new report[9] containing a single screen 100. As in all DOI applications,[10] we then use the Screen Painter (transaction *SE51*) to add a custom container control to the screen — the editing application (in this example, Microsoft Office) will be launched within this container control. We'll call this control "CONTAINER" (see **Figure 7**).

Now let's focus on the code. The first thing we have to do is initialize both the BDS and the DOI interfaces in the PBO screen event. Assume all variables have been declared globally at the top of our report. **Figure 8** shows the code we need to start with.

[9] I've created the demo program as a report because it easily supports the simple upload of a single file, but you could also create it as a module pool. The decision is arbitrary, but the code would look slightly different (e.g., there would be no "call screen 100" command in the start-of-selection event).

[10] See the resources cited on page 62 and the sidebar on page 30 for more on DOI concepts like controls, containers, etc.

Figure 7 Add a Custom Container Control

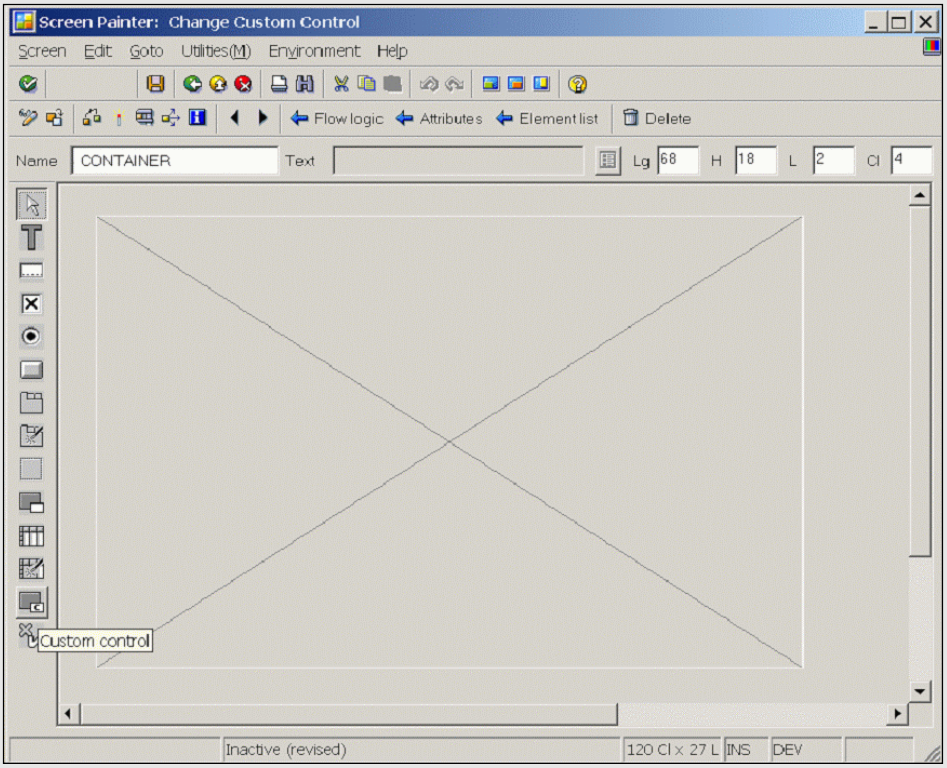


Figure 8 Initialize the BDS and DOI Interfaces

```
 1 * BDS initialization
 2   if obj_bds is initial.
 3     create object obj_bds.
 4   endif.
 5
 6 * DOI initialization
 7   if h_control is initial.
 8     call method c_oi_container_control_creator=>get_container_control
 9       importing control = h_control
10                 retcode = s_retcode.
11     call method c_oi_errors=>raise_message exporting type = 'E'.
12   endif.
13
14   if h_container is initial.
15     create object h_container
```

(continued on next page)

Figure 8 (continued)

```
16       exporting container_name = 'CONTAINER'.
17
18     call method h_control->init_control
19       exporting r3_application_name =
20                                 'BDS-DOI'
21         inplace_enabled = 'X'
22         inplace_scroll_documents = 'X'
23         parent = h_container
24       importing retcode = s_retcode.
25     call method c_oi_errors=>raise_message exporting type = 'E'.
26   endif.
```

(Note that the code listed in Figure 8, and all subsequent code samples in the article, are excerpted from the downloadable code.)

The first (BDS) segment (lines 2-4) creates an instance of BDS class CL_BDS_DOCUMENT_SET and stores the reference in variable *obj_bds*. We will use this variable to call all of the BDS methods we'll need. This is the only step needed to initialize and work with the BDS.

The second (DOI) segment (lines 7-12) asks the SAP Control Framework[11] to create a new DOI control object in memory and place a handle to it in variable *h_control*. The SAP Control Framework provides a convenient method — *c_oi_errors=>raise_message* — that reduces the amount of code we have to write (i.e., no *if-else* method is required to check the return code). The *c_oi_errors=>raise_message* method automatically checks for errors and returns to the main processing if none occur. If an error does occur, it is sent to static class *c_oi_errors*, which resides in global memory, and is retrieved through the method call listed.

The final (DOI) segment (lines 14-26) consists of two steps. First, we instantiate a DOI container object (*h_container*) to represent the custom container control created on screen 100. We specify its name (CONTAINER) exactly as we did in Figure 7. Then, we initialize the SAP Control Framework with *init_control*. By passing the handle to our *h_container* control, we permanently bind *h_control* to the container on our screen.

> ✓ ***Note!***
>
> *I've preceded each call in Figure 8 with a check to ensure it is executed only once.*

That's it! We've now got the foundation of a document management program, and we're ready to start adding the document management functionality. Let's start by enabling the program to create a blank document (in this case, a Word document) when users first launch the desktop application.

[11] The SAP Control Framework is a Basis component that enables and manages communication between the R/3 server and Dynpro controls that run locally on a user's PC (e.g., an SAP tree control or table control). DOI is built on top of the SAP Control Framework. (Refer to the sidebar on page 30 for a brief overview of SAP DOI and also to the resources listed on page 62.)

Figure 9 Retrieve a Document Proxy

```
CALL METHOD h_control->get_document_proxy
  EXPORTING
    document_type   = 'Word.Document'
    document_format = 'OLE'
  IMPORTING
    document_proxy  = h_proxy
    retcode         = s_retcode.
```

Figure 10 Create an Application Instance and Open a New Document

```
CALL METHOD h_proxy->create_document
  EXPORTING
    open_inplace = 'X'
  IMPORTING
    retcode      = s_retcode.
```

Creating a Document

Creating a new document involves adding a simple sequence of five method calls:

1. A call to DOI method *get_document_proxy* to retrieve a document proxy
2. A call to DOI method *create_document* to create an application instance and open a new document
3. A call to BDS method *create_with_URL* to create a document entry in the Knowledge Provider database
4. A call to DOI method *save_document_to_URL* to save the document to the repository
5. A call to BDS method *confirm_create* to add missing document metadata to the Knowledge Provider database

Call #1: A Call to DOI Method get_document_proxy to Retrieve a Document Proxy

In **Figure 9**, we ask the SAP Control Framework (using the handle *h_control*) to return a handle to a new document proxy (*h_proxy*). The document proxy handle represents the DOI document class interface. *Word.Document* is a predefined DOI keyword that identifies Microsoft Word as the application with which to display or edit the document.

Call #2: A Call to DOI Method create_document to Create an Application Instance and Open a New Document

The DOI call in **Figure 10** creates an instance of the Microsoft Word application and opens a new document. The *create_document* call creates a

Figure 11 Create an Entry in the Knowledge Provider Database

```
CALL METHOD h_bds->create_with_URL
  EXPORTING
    classname  = 'CM_DEMO1'
    classtype  = 'OT'
  CHANGING
    object_key = 'CM100000'
    components = i_bds_components
    signature  = i_bds_signature
    uris       = i_bds_uris
  EXCEPTIONS
    …
```

document in memory. At this point, no physical document is created or opened on the PC. The *open_inplace* parameter tells the DOI interface to launch Microsoft Word within the SAPGUI context. It knows to open the application inside the container control on the SAPGUI screen because we specified that control during the initialization in Figure 8.

Call #3: A Call to BDS Method create_with_URL to Create a Document Entry in the Knowledge Provider Database

In **Figure 11**, we ask the BDS to do two things:

1. Insert a preliminary entry in the Knowledge Provider database for our document.

2. Return a BDS-specific URL that we will pass to *save_to_URL* when the user is ready to save the document.

> ✓ ***Note!***
>
> *The "with_URL" BDS methods were specifically designed for use with DOI to integrate document creation and editing functionality into SAPGUI screens.*

(Note that this does not physically save the document content to the database. The save is performed via the call to DOI method *save_document_to_URL*, which is discussed in the next section.)

Figure 11 uses a few important parameter settings that require some discussion before we go any further:

- *classname* and *classtype* — You may recall that in the Knowledge Provider, the class name and class type fields collectively define a type of application object. For illustration purposes, I've specified arbitrary values (class name *CM_DEMO1* and class type *OT*) to uniquely identify our document management program.

- *object_key* — An object key is a logical collection of related documents within the program. To keep the demo simple, I've specified the static value *CM100000* to identify the new test document. You can also have the method generate a unique ID by providing an initialized variable.

- *components* — This parameter identifies the actual files you want to store under this document entry in the Knowledge Provider. Initially, we wish to save a single Word document (after the user enters content) so we load the components

Figure 12 Values Entered in the Components Table

```
wa_bds_components-doc_count = 1.
wa_bds_components-comp_count = 1.
wa_bds_components-mimetype = 'application/msword'.
wa_bds_components-comp_id = 'CMFUNCT_DESIGN.doc'.
append wa_bds_components TO i_bds_components.
```

table as shown in **Figure 12**. (Note that in practice, you would specify both the *comp_id* and *mimetype* parameters in Figure 12 using variables, instead of hard-coding the values as I've done here in the example.)

Field *doc_count* is a counter field that lets us group physical documents that may consist of one or more components. In this case, we are creating[12] only one document with a single component, but you may wish to create multiple documents, each with one or more components. Fields *comp_id* and *mimetype* are used to specify the document's file name and type,[13] respectively.

[12] The word "create" can be confusing in this context. The document file already exists. All we are doing is "registering" it with the Knowledge Provider, which creates an entry for it, stores its metadata, and (if storing the document in R/3) saves the file to the R/3 database.

[13] MIME (Multipurpose Internet Mail Extension) types are keywords that universally identify different documents (e.g., *image/jpeg* for a JPEG image or *application/pdf* for a PDF document). MIME types are defined by a cross-industry body (i.e., not SAP). Use the DOI static method call *c_oi_container_control_creator=>ole_to_mime* to dynamically determine a document's MIME type.

> ✓ ***Note!***
>
> *DOI functions like "get_document_proxy" expect document types to be specified using OLE keywords like "Word.Document," while BDS functions use cross-industry MIME types.*

- *signature* — Use this parameter to attach properties to the document (or documents) being stored. You can then use the BDS *query_documents* method to search for documents by these values. (More on searching for documents in an upcoming section.)

  To define properties, place standard keywords[14] in the *prop_name* field and values in the *prop_value* field in the signature table, as shown in **Figure 13**.

[14] A listing of standard BDS attributes can be found in the 4.6C online help (**http://help.sap.com**) under *SAP Library → Basis Components → Basis Services/Communication Interfaces → Business Document Service → Concepts*.

Figure 13 Values Entered in the Signature Table

```
wa_bds_signature-doc_count = '1'.
wa_bds_signature-prop_name = 'BDS_KEYWORD'.
wa_bds_signature-prop_value = 'Key1'.
APPEND wa_bds_signature TO i_bds_signature.
```

(continued on next page)

Figure 13 (continued)

```
wa_bds_signature-prop_name = 'BDS_DOCUMENTCLASS'.
wa_bds_signature-prop_value = 'MS Word Application'.
APPEND wa_bds_signature TO i_bds_signature.

wa_bds_signature-prop_name = 'DESCRIPTION'.
wa_bds_signature-prop_value = 'CM document description'.
APPEND wa_bds_signature TO i_bds_signature.

wa_bds_signature-prop_name = 'BDS_DOCUMENTTYPE'.
wa_bds_signature-prop_value = 'BDS_ATTACH'.
APPEND wa_bds_signature TO i_bds_signature.
```

The *doc_count* field links the components and signature tables, so make sure the value you specify here matches the document's corresponding value in the components table. For our example, I've chosen to load *BDS_KEYWORD*, *BDS_DOCUMENTCLASS*, *BDS_DESCRIPTION*, and *BDS_DOCUMENTTYPE*, since I've found these most useful when searching for documents.

- *uris* — After calling the *create_with_URL* method, table parameter *uris* will contain a BDS-specific URL for each component specified in the components table. Be sure to clear this table before calling the method.

Call #4: A Call to DOI Method save_document_to_URL to Save the Document to the Repository

Having provided a URL, the BDS expects the document to be uploaded via the *save_document_to_URL* method, which is what we do in **Figure 14**. The s*ave_document_to_URL* method will recognize the URL as a BDS-specific URL and will pass the uploaded document to the Knowledge Provider, which saves it to the repository.

Call #5: A Call to BDS Method confirm_create to Add Missing Document Metadata to the Knowledge Provider Database

The final call (**Figure 15**) updates the Knowledge Provider database with any missing document metadata (e.g., file size or last changed date).

> ✓ ***Note!***
>
> *If you don't call the BDS method "confirm_create" after calling "create _with_URL," your document's metadata will be incomplete, and thus the document cannot be retrieved with the BDS API. You will need to delete the document either using the BDS "delete" method in the program code or graphically from within the Business Document Navigator (BDN), and then re-create the document. See the sidebar on pages 50-51 for more on the BDN.*

To recap, so far we've initialized the BDS and the DOI interfaces, generated a new Word document in memory, opened it in our Dynpro container control, saved it to the repository, and registered it in the

Figure 14 Save the Document to the Repository

```
CALL METHOD h_proxy->save_document_to_URL
  EXPORTING
    URL     = i_bds_uris-uri
  IMPORTING
    retcode = s_retcode.
```

Figure 15 Add Missing Metadata to the Knowledge Provider Database

```
CALL METHOD obj_bds->confirm_create
  EXPORTING
    classname       = 'CM_DEMO1'
    classtype       = 'OT'
    object_key      = 'CM100000'
    x_force_confirm = 'X'
  CHANGING
    signature       = i_bds_signature
    uris            = i_bds_uris
    components      = i_bds_components
  EXCEPTIONS
    …
```

Knowledge Provider database. Let's now examine how we can enable users to update an existing document.

Updating a Saved Document

We only need to add three calls to the program to enable users to make changes to a previously saved document:

1. A call to BDS method *update_with_URL* to update the document metadata and retrieve its URL
2. A call to DOI method *save_document_to_URL* to save the updated document to the repository
3. A call to BDS method *confirm_update* to update the document metadata

Call #1: A Call to BDS Method update_with_URL to Update the Document Metadata and Retrieve Its URL

As described earlier, *update_with_URL* does not actually save a document. Instead, it generates a fresh URL for the document (or a set of URLs if there is more than one document) that the DOI method *save_document_to_URL* will use to store

Figure 16 Retrieve the Document's URL

```
CALL METHOD obj_bds->update_with_URL
  EXPORTING
    classname      = 'CM_DEMO1'
    classtype      = 'OT'
    object_key     = 'CM100000'
    doc_id         = wa_bds_signature-doc_id
    doc_ver_no     = wa_bds_signature-doc_ver_no
    doc_var_id     = wa_bds_signature-doc_var_id
    x_force_update = 'X'
  CHANGING
    components    = i_bds_components
    uris          = i_bds_uris
    signature     = i_bds_signature
  EXCEPTIONS
    …
```

the document (or documents). As coded in **Figure 16**, we will find this URL in field *i_bds_uris[1]-uri*.

As before, we specify our target application in *classname* and *classtype*, and we specify the target document set with *object_key* — these values must match those of the existing document. Since we're updating a specific document within the document set (in this case, the only document within the set), we need to explicitly identify it using fields *doc_id*, *doc_ver_no*, and *doc_var_id*. Rather than hard-code 1s into these fields, I've reused the version values we placed in the signature table when creating the document.

Call #2: A Call to DOI Method save_document_to_URL to Save the Updated Document to the Repository

With the call shown in **Figure 17**, the system physically uploads the updated, cached version from the PC and overwrites the existing version in the database.

Call #3 : A Call to BDS Method confirm_update to Update the Document Metadata

Just as when we were creating a document, when working with URLs we need to confirm the save operation. The call to *confirm_update* (**Figure 18**) finalizes the document's metadata (e.g., file size or last changed date).

You've now seen how to enable users to create and update a Microsoft Word document. You can use almost the same code to enable users to create Microsoft Excel and Visio documents stored under application class *CM_DEMO1* with object key *CM100000*. Simply:

- Adjust the *document_type* to *Excel.Sheet* or *Visio.Drawing* in the DOI interface call to *get_document_proxy*
- Set the *wa_bds_components-mimetype* to *application/vnd.ms-excel* or *application/vnd.visio* in the BDS class method *create_with_URL*

Figure 17 Save the Updated Document to the Repository

```
CALL METHOD h_proxy->save_document_to_URL
  EXPORTING
    URL     = i_bds_uris-uri
  IMPORTING
    retcode = s_retcode.
```

Figure 18 Update the Document Metadata

```
CALL METHOD obj_bds->confirm_update
  EXPORTING
    classname       = 'CM_DEMO1'
    classtype       = 'OT'
    object_key      = 'CM100000'
    x_force_confirm = 'X'
  CHANGING
    signature       = i_bds_signature
    uris            = i_bds_uris
    components      = i_bds_components
  EXCEPTIONS
    …
```

Before adding any more code, let's make sure our program is working properly, which will also allow us to explore different ways of searching for documents.

Searching for a Document and Testing the Method Calls

Assume we have created and saved three documents with our program (a Word functional design document, a Visio technical specification document, and an Excel test plan document). Let's now retrieve them to make sure the program is working as it should.

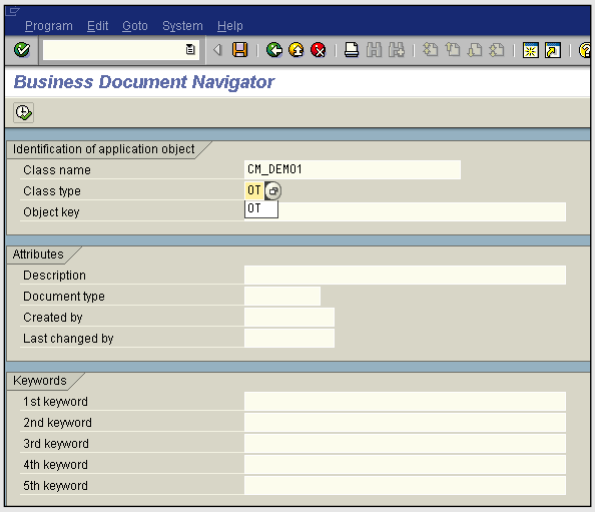
There are a few ways we can search for Knowledge Provider documents:

- By using the BDS method *query_documents*, which you can use to search by properties we attached to the document using the signature table (this method is extremely easy to use, so I won't belabor the details here)
- By using the Business Document Navigator (see the sidebar on the next page for details on how to use the BDN navigation tool)
- By running the BDS method *get_info* in test mode in the Class Builder (transaction *SE24*)

The method you choose will depend upon the results you are looking for. For our purposes, running the *get_info* method will allow us to verify the entries in the components and signature tables.

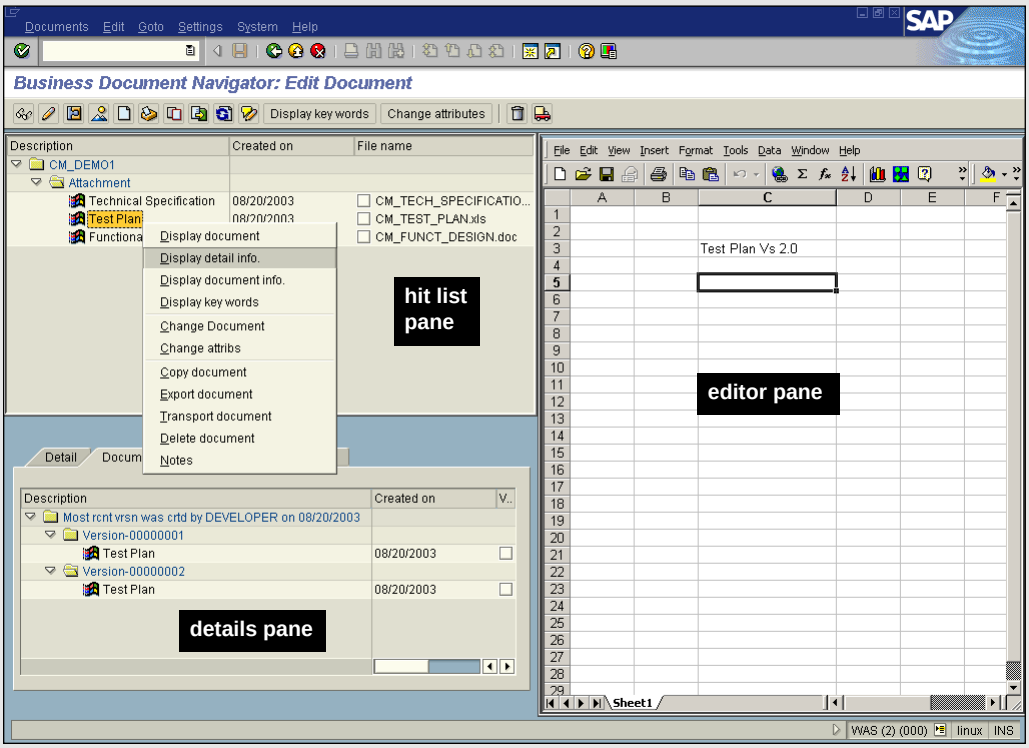
Managing Documents with the Business Document Navigator (BDN)

The Business Document Navigator (BDN), which is part of the BDS, is a central tool for navigating and managing Knowledge Provider documents. You can access it via transaction OAOR or the menu path *Office* → *Business Documents* → *Documents* → *Find*.



The first screen of the BDN is a document selection screen (shown to the right). While entries on the screen will depend on what you want to see, there are two ways to search for a document. If you want to browse all documents, just enter a class name and class type (note that these are required fields). If you are looking for documents tied to a particular object, like a customer number or equipment record, enter that number in the object key field. You can also search for documents and limit search results using keywords.

After you execute the search (by clicking on the icon), the main BDN screen appears (shown below).



The main BDN screen is organized into three panes:

- The **hit list pane** is a tree view of documents organized first by application class and then by document type.*

  To display or edit document content or metadata, drill down to the document level and right-click on the document's node to open a context menu. Most of the BDN's functions are provided in this context menu (note that the content of context menus in the BDN are node-specific, so there will be slight variations).

- The **details pane** displays document metadata.

  You can update most values directly by selecting a document node and either navigating to *Edit* → *Change Attributes* or right-clicking to open the context menu.

- The **editor pane** displays document contents and lets you edit documents in place via DOI (as you can see in the example screen, an Excel document is open within the editor pane).

  When you select a document (by clicking on a document node in the hit list pane), the Knowledge Provider retrieves the document from the repository. When you save changes to the document, the Knowledge Provider updates the document in the repository.

Key BDN functions include the following:

- **Creating new documents:** Creating a new document is not intuitive in the BDN. First, go to the *Create* tab in the details pane (clicking on the button in the application toolbar will also take you to this tab). From here, open the *Standard doc. Types* node and select a document type. Finally, right-click to open the context menu and either import an existing document or start a document authoring application (e.g., Microsoft Word, Excel, etc.).

- **Displaying an existing document:** To display a document shown in the hit list pane, right-click on the document node and select *Display document* from the context menu. The most effective way to locate documents is to use the parameters in the BDN selection screen. Manually searching though a large hit list can be very time-consuming.

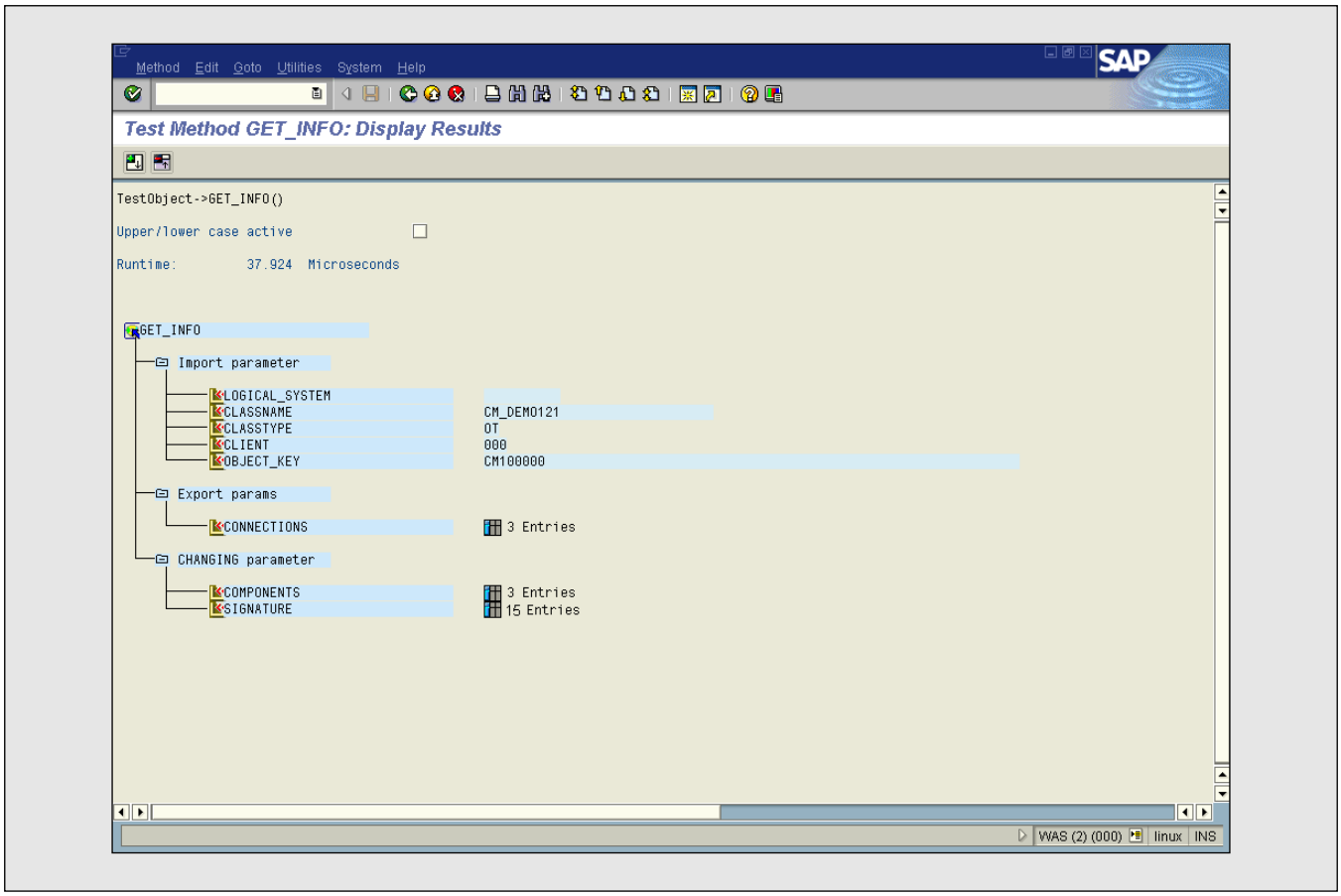
- **Saving an edited document:** Unfortunately, this involves more than just clicking on the save icon. As shown in the main screen, the save icon in the standard toolbar is "grayed out" in edit mode (SAP is aware of this issue). The workaround for now is to force the BDN to ask you to save by pressing *F3* to back out. A dialog will then ask you if you want to update the current version or create a new version or variant.

- **Displaying a document's versions and variants:** Document versions and variants are shown in the details pane on the *Document info* tab. To retrieve a particular version or variant, open the version folder, right-click on the document node, and select *Display document info* from the context menu.

---

* When storing a document with the BDS, you specify a document type using the attribute keyword BDS_DOCUMENTTYPE in the *signature* table parameter.

Figure 19 Executing BDS Method get_info in Test Mode



Open the Class Builder (transaction *SE24*), pull up class CL_BDS_DOCUMENT_SET, and switch to test mode by pressing *F8* or clicking on the icon in the application toolbar. Single-click on the icon to the right of the *get_info* method ( ) to open it, and enter the parameter values we've used:

- Class name *CM_DEMO1*
- Class type *OT*
- Your client ID
- Object key *CM100000*

Now run the *get_info* method (press *F8*). If the system locates the documents successfully, your screen will look like **Figure 19**. The components table should contain three entries (one for each document) and the signature table should contain 15 properties. For your reference, I've provided the details of each table's contents in **Figure 20**. Note that the BDS generated an ID for each document. Otherwise, the contents are self-explanatory and provide a good example of how the parameter values were used to perform the search.

Now that we've confirmed the program is working properly, let's look at how we can make it easy for users to retrieve an existing document.

Retrieving an Existing Document

To help users retrieve an existing document, we'll next generate a pop-up list of all registered documents that users can choose from. To do this, we'll add the

Figure 20 Key Contents of the Components and Signature Tables

Components Table

doc_count	comp_count	comp_id	mimetype	comp_size
1	1	CM_FUNCT_DESIGN.doc	application/msword	20480
2	1	CM_TECH_SPEC.vsd	application/vnd.visio	38400
3	1	CM_TEST_PLAN.xls	application/vnd.ms-excel	5120

Signature Table

doc_count	doc_id	…	prop_name	prop_value
1	BDS_LOC1 DDEA043F…	…	BDS_DOCUMENTTYPE	BDS_ATTACHMENT
1	BDS_LOC1 DDEA043F…	…	CREATED_BY	DEVELOPER
1	BDS_LOC1 DDEA043F…	…	DESCRIPTION	Functional Design
1	BDS_LOC1 DDEA043F…	…	STATE	2
1	BDS_LOC1 DDEA043F…	…	STORAGE_CATEGORY	BDS_DB
2	BDS_LOC1 DDEA043F…	…	BDS_DOCUMENTTYPE	BDS_ATTACHMENT
2	BDS_LOC1 DDEA043F…	…	CREATED_BY	DEVELOPER
2	BDS_LOC1 DDEA043F…	…	DESCRIPTION	Technical Specification
2	BDS_LOC1 DDEA043F…	…	STATE	2
2	BDS_LOC1 DDEA043F…	…	STORAGE_CATEGORY	BDS_DB
3	BDS_LOC1 DDEA043F…	…	BDS_DOCUMENTTYPE	BDS_ATTACHMENT
3	BDS_LOC1 DDEA043F…	…	CREATED_BY	DEVELOPER
3	BDS_LOC1 DDEA043F…	…	DESCRIPTION	Test Plan
3	BDS_LOC1 DDEA043F…	…	STATE	2
3	BDS_LOC1 DDEA043F…	…	STORAGE_CATEGORY	BDS_DB

following five method calls to the program (one of which, #3, is optional):

1. A call to BDS method *get_info* to retrieve the list of components
2. A call to BDS method *get_with_URL* to retrieve the URL of the selected document
3. A call to BDS method *get_info* to load only the selected document's records (optional)
4. A call to DOI method *get_document_proxy* to retrieve a fresh document proxy
5. A call to DOI method *open_document* to open the document

Figure 21 Retrieve a List of Components

```
CLEAR: i_bds_components, i_bds_signature.
  CALL METHOD obj_BDS->get_info
    EXPORTING
      classname  = 'CM_DEMO1'
      classtype  = 'OT'
      object_key = 'CM100000'
    CHANGING
      components = i_bds_components
      signature  = i_bds_signature
    EXCEPTIONS
      …
```

Call #1: A Call to BDS Method get_info to Retrieve a List of Components

The call shown in **Figure 21** retrieves the list of components (and their descriptions) for our pop-up list. Notice we've asked the method to retrieve metadata for all documents associated with our demo application object (class name *CM_DEMO1*, class type *OT*, and object key *CM100000*), which in the example will be a Word document, a Visio document, and an Excel document.

> ✓ ***Note!***
>
> *To avoid getting sidetracked, let's assume that we've generated the pop-up list (to see how this was done, review the download code) and skip ahead to how the document is retrieved once selected from the list.*

Call #2: A Call to BDS Method get_with_URL to Retrieve the URL of the Selected Document

To enable a user to retrieve the Visio document by selecting it from the pop-up list, we use the *get_with_URL* method to retrieve the URL of the document (**Figure 22**). The *get_with_URL* method requires an entry in the signature table that identifies the requested document through the fields *doc_id*, *doc_ver_no*, and *doc_var_id*, so we supply the method call with a single signature table entry that identifies the target document (in the example, the Visio document).

Call #3: A Call to BDS Method get_info to Load Only the Selected Document's Records (Optional)

Figure 23 is an optional call that populates the components table with records only for the document selected by the user. Remember that in order to provide the user with a list of documents to choose from, the first call to *get_info* (shown in Figure 21) retrieved the full document set, which in turn loaded the components table with the records for all documents in the document set. The call in Figure 23, on the other hand, loads the components table only with records for the selected document — i.e., the three main table parameters (*components*, *signature*, and *uris*) associated with these method calls will be filled only with the records for the selected document (the Visio document in the example here).

Call #4: A Call to DOI Method get_document_proxy to Retrieve a Fresh Document Proxy

Before the document can be opened in Visio, we

Figure 22 Retrieve the URL of the Selected Document

```
" code to delete all entries from the signature table except for
" the selected item
READ TABLE i_bds_signature INTO wa_bds_signature INDEX user_choice.
CLEAR i_bds_signature.
APPEND wa_bds_signature TO i_bds_signature.

" retrieve the url for the selected document
CALL METHOD obj_BDS->get_with_URL
  EXPORTING
    classname       = 'CM_DEMO1'
    classtype       = 'OT'
    object_key      = 'CM100000'
  CHANGING
    uris            = i_bds_uris
    signature       = i_bds_signature
  EXCEPTIONS
    nothing_found   = 1
    error_kpro      = 2
    internal_error  = 3
    parameter_error = 4
    not_authorized  = 5
    not_allowed     = 6.
```

Figure 23 Load Only the Selected Document's Records

```
CLEAR i_bds_components.
  CALL METHOD obj_BDS->get_info
    EXPORTING
      classname  = 'CM_DEMO1'
      classtype  = 'OT'
      object_key = 'CM100000'
    CHANGING
      components = i_bds_components
      signature  = i_bds_signature
    EXCEPTIONS
      …
```

need to use *get_document_proxy* to retrieve a fresh proxy variable that refers to the document. For convenience, we'll reuse variable *h_proxy* after clearing it. But *get_document_proxy* requires us to specify the

Figure 24 Retrieve a Fresh Document Proxy

```
CLEAR h_proxy.

  CALL METHOD c_oi_container_control_creator=>mime_to_ole
    EXPORTING
      mimetype = i_bds_components-mimetype
    …
    IMPORTING
      oletypes = oletypes.

  CALL METHOD h_control->get_document_proxy
    EXPORTING
      document_type   = oletypes
      document_format = 'OLE'
    IMPORTING
      document_proxy  = h_proxy
      retcode         = s_retcode.
```

document's type using a set of reserved OLE keywords (e.g., *Visio.Drawing*). Since our program supports more than one type of document (i.e., Word and Excel documents in addition to Visio documents), we do not want to hard-code this value. So where will we get it?

The answer is to derive the OLE document type from the document's MIME type, which BDS provides in field *mimetype* of the components table. In anticipation of this need, SAP developed a static DOI method, *mime_to_ole*, to do this. So, as shown in **Figure 24**, we pass *i_bds_components-mimetype* to *mime_to_ole*, receive the converted type into *oletypes*, and pass this variable for the *document_type* parameter of *get_document_proxy.*

The result is a brand-new proxy that we can use to open the document.

Call #5: A Call to DOI Method open_document to Open the Document

The *open_document* call (**Figure 25**) opens the document in the appropriate Office application (in this case, Visio). Since we've specified *open_inplace = 'X'*, the application is embedded within the container control on the screen.

Let's see how we can add one more important functionality to our application — the ability to create different versions of a document.

Creating a New Version of a Document

When introducing the Knowledge Provider, I mentioned that it can help you define and manage multiple versions and variants of a document. This powerful feature gives business users the same sense of security you and I get from R/3 version management, so you'll want to include it in nearly all document management programs.

The process of creating a new version involves only one new BDS method:

Figure 25 *Open the Document*

```
CALL METHOD h_proxy->open_document
  EXPORTING
    document_URL  = s_doc_URL
    open_inplace  = 'X'
  IMPORTING
    retcode       = s_retcode.
```

create_version_with_URL.[15] BDS offers similar methods for variants, which will be self-explanatory once you understand how to use this method. We'll need to add the following three calls:

1. A call to BDS method *create_version_with_URL* to declare the version in the database

2. A call to DOI method *save_document_to_URL* to save the document to the repository

3. A call to BDS method *confirm_create* to update the document metadata

Call #1: A Call to BDS Method create_version_with_URL to Declare the Version in the Database

The *create_version_with_URL* method is similar to *create_document*, with two main exceptions. First, *create_version_with_URL* stores a new document version with an existing document instance (through fields *doc_id*, *doc_ver_no*, and *doc_var_id*). Behind the scenes, the BDS creates a new physical document and defines a version relationship between the two documents. The method returns a new version number if it succeeds. Second, we pass document properties using a parameter called *properties* instead of *signature*. The difference is purely semantic.

Typically, when a user is creating a new version of a document, an existing document is also being displayed, so most of these variables will already be in memory. The easiest way to specify the document ID, version, and variant parameters is usually to read the document's entry from the signature table into a work area (*wa_bds_signature*), as shown in **Figure 26**.

Structures *i_bds_uris* and *i_bds_properties* need to be cleared before making the call. Upon successful completion, a new components table entry is created, internal signature table entries associated with the new version are created, and the target URL of the new version is returned in *i_bds_uris*.

[15] As you might expect, we used the *with_URL* version — instead of *with_files* or *with_table* — since we are presenting the document through DOI and will need the URL to update further changes to the document version.

Figure 26 *Declare the Version in the Database*

```
READ TABLE i_bds_signature INTO wa_bds_signature INDEX 1.

CALL METHOD obj_bds->create_version_with_URL
  EXPORTING
    classname      = 'CM_DEMO1'
    classtype      =  'OT'
```

(continued on next page)

Figure 26 (continued)

```
      object_key     = 'CM100000'
      doc_id         = wa_bds_signature-doc_id
      doc_ver_no     = wa_bds_signature-doc_ver_no
      doc_var_id     = wa_bds_signature-doc_var_id
    IMPORTING
      new_doc_ver_no = new_ver_no
    CHANGING
      components     = i_bds_components
      uris           = i_bds_uris
      properties     = i_bds_properties
    EXCEPTIONS
      …
```

Figure 27 Save the Document to the Repository

```
CALL METHOD h_proxy->save_document_to_URL
   EXPORTING
     URL     = i_bds_uris-uri
   IMPORTING
     retcode = s_retcode.
```

Call #2: A Call to DOI Method save_document_to_URL to Save the Document to the Repository

The call shown in **Figure 27** saves the version to its repository in the same way that the new document and updated document were saved earlier, so I won't go into the details here.

Call #3: A Call to BDS Method confirm_create to Update the Document Metadata

Again, the call shown in **Figure 28** updates the version metadata in the same way that the metadata of the new document and the updated document was added/updated earlier.

That's it — we're done. Let's run the demo program to see how things should look.

Running the Program

Figure 29 shows the demo program in action — you can see the pop-up list of existing documents that users can choose from. The program's features can be accessed using the icons in the application toolbar. Users can create three different types of documents using the corresponding icons (a Word document, a Visio document, or an Excel document), and can also open and delete existing documents, as well as create versions. The *Get Doc Info* button executes BDS

Figure 28 Update the Document Metadata

```
CALL METHOD obj_bds->confirm_update
  EXPORTING
    classname       = 'CM_DEMO1'
    classtype       = 'OT'
    object_key      = 'CM100000'
    x_force_confirm = 'X'
  CHANGING
    signature       = i_bds_signature
    uris            = i_bds_uris
    components      = i_bds_components
  EXCEPTIONS
    …
```

Figure 29 List of Documents Available for the User to View and Edit

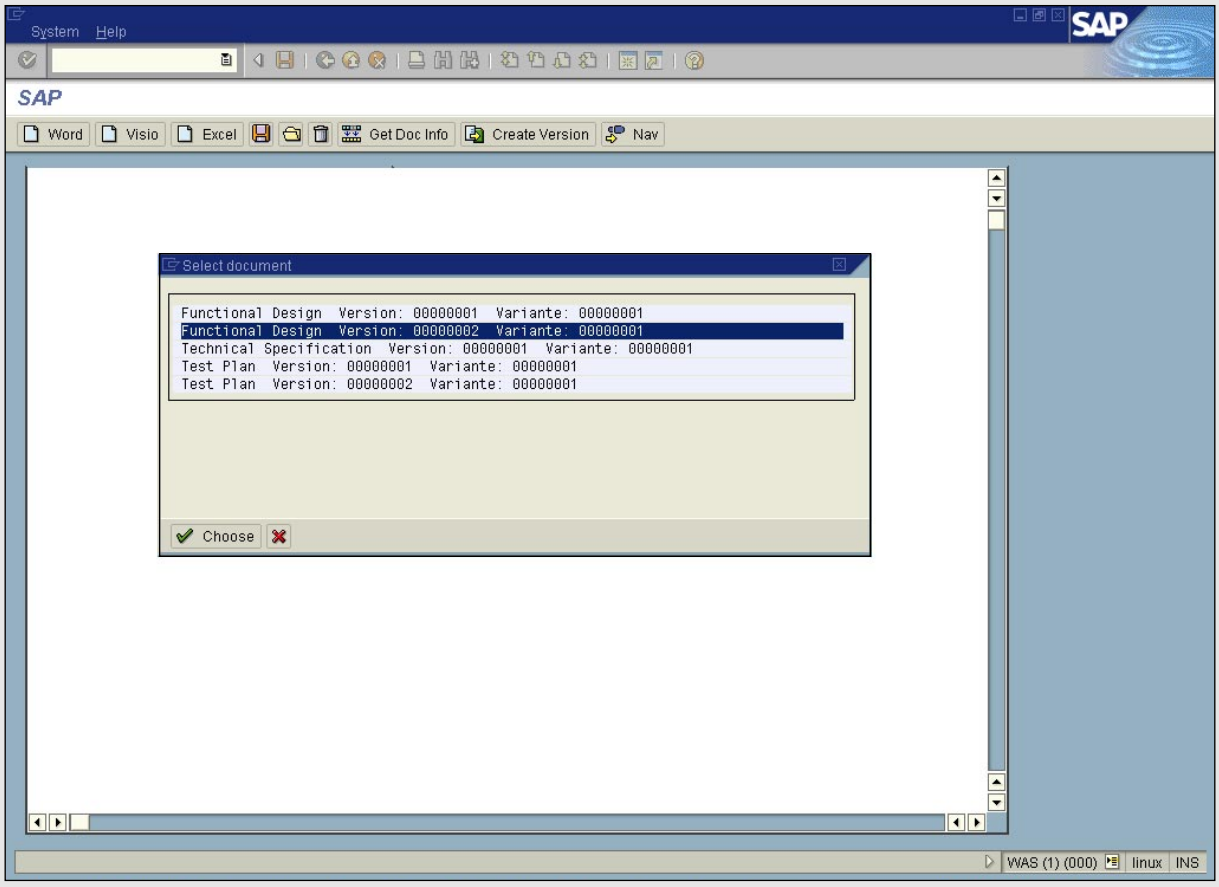
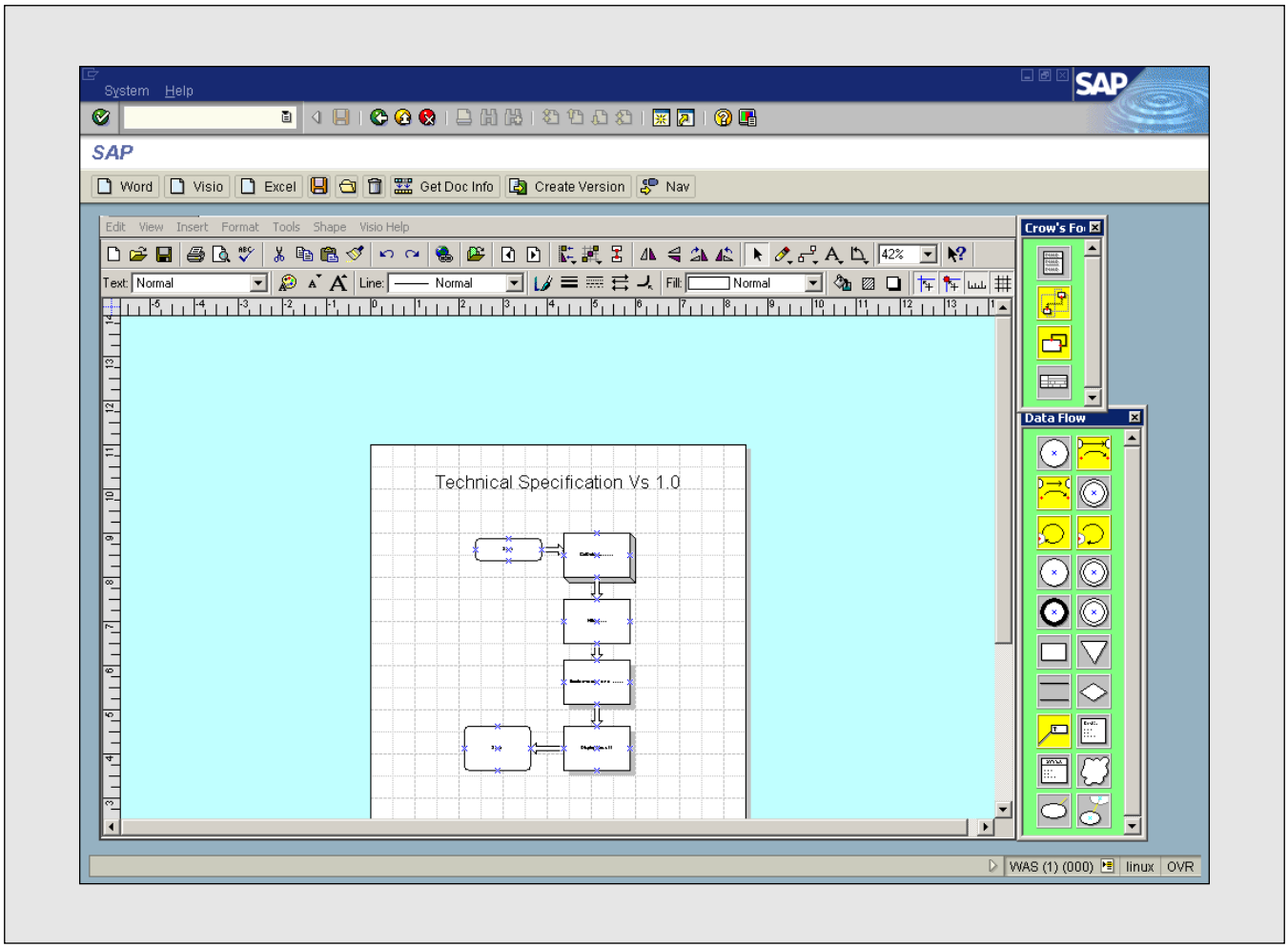


Figure 30 Visio Document Displayed in the Custom R/3 Screen



class method *get_info* and stops at a hard-coded breakpoint in the subroutine so you can examine the important BDS table structures.

Behind the scenes, when you click on the save icon, DOI method *save_document_to_URL* physically saves the file via the URL provided by the BDS. Similarly, existing documents are opened with *get_with_URL*, using the URL provided by the BDS. The retrieved URL is used by DOI to open the document through method *open_document*.

After running the demo and creating some documents (**Figure 30** shows a Visio document loaded in the Dynpro container control), review the documents in the Business Document Navigator (BDN) by clicking on the *Nav* toolbar button (refer back to the sidebar on pages 50-51 for details on using the BDN). I've included code with the download that demonstrates how to launch the BDN programmatically (using method *call_navigator*), or you can run it manually via transaction *OAOR (*or menu path *Office* → *Business Documents* → *Documents* → *Find)*. Either way, once the BDN is launched, just enter class name *CM_DEMO1*, class type *OT*, and execute by pressing *F8* or clicking on the icon.

Tips for Success

☑ Since the BDS interface is implemented using

ABAP Objects, you can easily customize or extend the BDS functionality by subclassing the CL_BDS_DOCUMENT_SET class and overriding any methods you wish.

☑ When storing documents in R/3, SAP strongly recommends using the BDS instead of directly writing binary document data to R/3 database tables yourself in ABAP. Documents stored through the BDS automatically benefit from BDS features like frontend caching.

☑ A good way to become comfortable with the Knowledge Provider functionality is to experiment with the Business Document Navigator (transaction *OAOR*). You can then use it to review document sets added programmatically. The *Document detail* tab in the BDN lists most document metadata (some attributes like storage category, state, and user-defined attributes are not listed).

☑ When programming with the BDS, use the Class Builder (transaction *SE24*) to test and analyze your BDS and DOI method calls.

☑ With all BDS method calls, pay special attention to the top three BDS table parameters: *signature*, *components*, and *uris*.

☑ SAP provides the following set of tools you can use to test your DOI infrastructure (you'll find them in the *C:\Program Files\SAPpc\SAPGui\TestTools* folder on any PC with a full SAPGUI installation):

- *Check_DOI.exe* evaluates the target PC's configuration for running DOI
- *DOI_WORD_Check.doc* verifies that the target PC can run Microsoft Word through DOI
- *DOI_XLS_Check.xls* verifies that the target PC can run Microsoft Excel through DOI

☑ Remember that the calls to BDS methods *create_with_URL* and *update_with_URL* must be followed by *confirm_create* or *confirm_update*. If they are not, your document's metadata will be incomplete, and you will need to delete the document (either with the BDS *delete* method or from within the BDN), and then re-create it.

☑ An alternative to using the BDS to add document management functions to ABAP programs is to launch the BDN directly from within the program using the following syntax:

```
CALL METHOD
  CL_BDS_DOCUMENT_SET=>CALL_NAVIGATOR.
```

Control will return to your program (at the line immediately following this command) when the user exits the tool. While this method is tempting, it complicates the user interface and usually presents users with more functionality than they need for the given task.

Conclusion

On its own, the Knowledge Provider's Business Document Service (BDS) provides you, the ABAP programmer, an interface with which to integrate document management functionality (e.g., to check documents in or out of a repository or manage document versions) into your ABAP programs. A more powerful technique, however, is to use the BDS in combination with SAP's Desktop Office Integration (DOI) classes to enable users to view, edit, or create documents *within a pane in an SAPGUI screen.* This saves users from having to launch an external desktop tool and subsequently check in documents manually. Both the BDS and DOI classes are standard in R/3 Release 4.6A and higher.

Take the time to design a document management solution that will be strategic in the long term. Key questions to answer include:

- Will you store documents in R/3 or in an external system like SAP Content Server or Open Text Livelink?
- Will you offer users web-based searching? For example, the Knowledge Provider was designed

Resources

- ☑ SAP provides a BDS-DOI demonstration program, SAPRDEMO_DOI_BDS, that wraps all the functionality discussed in this article into a single class. The program demonstrates simplifying programming interfaces and code reuse through object programming as well as BDS-DOI programming.
- ☑ For documentation on the BDS and DOI classes, visit **http://help.sap.com**. In the 4.6C library, you'll find BDS documentation under *Basis Components → Basis Services/Communication Interfaces → Business Document Service*. For documentation on DOI, go to *Basis Components → Controls and Control Framework → Desktop Office Integration*.
- ☑ Rainer Ehre's article "SAP Desktop Office Integration (SAP DOI) — An Easier Way for ABAP Programmers to Integrate Desktop Applications with R/3" (published in the March/April 2000 issue of *SAP Professional Journal*) provides an excellent overview of DOI and its architecture.
- ☑ I wrote an article that reviews how to use the DOI interfaces, "Data Downloads to Excel Made Simple with SAP's Desktop Office Integration (DOI) — A Programmer's Guide," which was in the May/June 2001 issue of this publication.
- ☑ The DOI interface can be extended to work with any ActiveX-compliant application. For details, see my article "Integrating Custom ActiveX Documents into SAPGUI-Based Programs Using SAP's Desktop Office Integration (DOI)" in the July/August 2002 issue of this publication.

to integrate with sophisticated document search engines such as Autonomy.

- Will you integrate your document management solution with other R/3 and non-R/3 document management systems you may have, like SAP Knowledge Warehouse, SAP Enterprise Portal Knowledge Management, or SAP Records Management?[16]

As you can see, there are many important decisions that have significant cost and strategic implications, so take the time to prepare a thoughtful plan for how you will best meet your own organization's document management needs.

[16] SAP Records Management is a new add-on to the SAP Web Application Server that offers additional document management functionality, such as the ability to integrate workflow and transactions with business documents. For more information, visit the SAP Service Marketplace at **http://service.sap.com/recordsmanagement**.

Philip Bremner is a contract programmer at Visual SAP, where he specializes in SAP DOI, BDS, ALV Grid Control, and Control Framework programming. He has 10 years of programming experience in ABAP, Visual C++, and Visual Basic, and is a Microsoft Certified Professional holding nine Microsoft certifications. Philip currently has a B.S. in industrial engineering from the State University of New York, Buffalo, and a B.S. in computer science from California State University, Bakersfield. He can be reached at Phil@VisualSAP.com.

