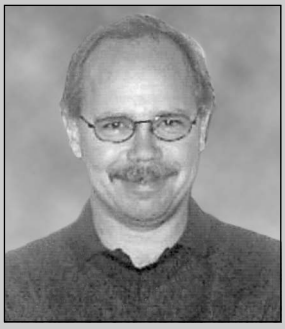
Data Downloads to Excel Made Simple with SAP's Desktop Office Integration (DOI) — A Programmer's Guide

Philip Bremner



Philip Bremner is a Microsoft Certified Professional holding nine Microsoft certifications including two Visual Basic™ certifications. Currently he is a senior programming analyst with Aera Energy LLC in Bakersfield, California, where he is acting as an Information Technology Lead in their Enterprise Architecture Implementation project.

(complete bio appears on page 46)

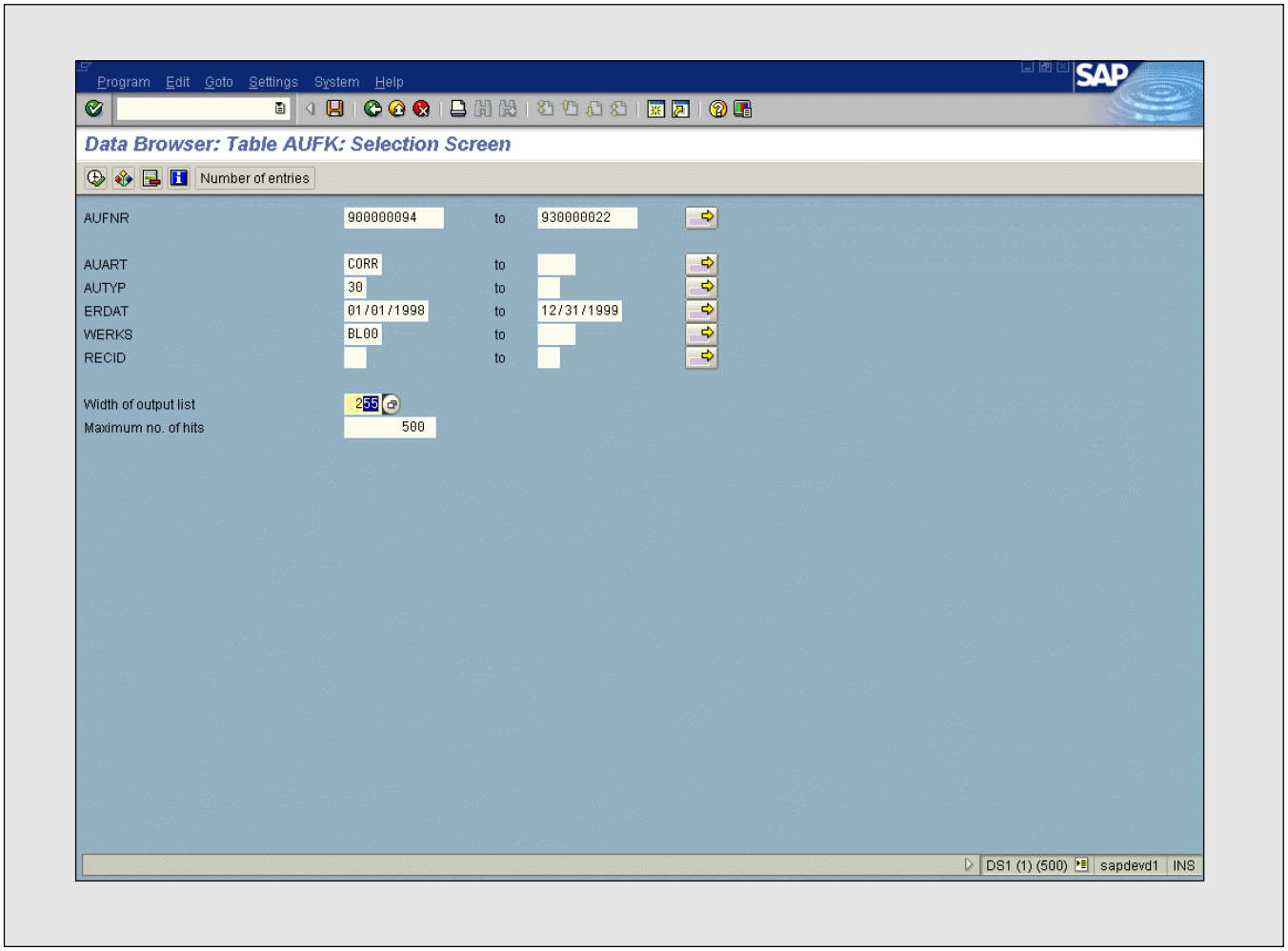
Downloading data from SAP is usually done by displaying the data in the Data Browser and using SAP's list-saving functionality to transfer the displayed data to a file on the client computer. Unfortunately, the SAP Data Browser display is limited to 255 characters.[1] This width limitation often boxes users into a corner, forcing them to be very selective as to which columns and/or how many columns of data they choose for display. It should also be noted that data downloads are limited strictly to the selected columns. But you can circumvent these restrictions by combining the Data Browser functionality with SAP's Desktop Office Integration (DOI) data transfer functionality.

The process is quite simple. First, you emulate SAP's Data Browser table selection and data querying functionality. Then, with the help of DOI, transfer all the selected data to a properly formatted Excel document on the client PC. Transaction SE16 provides the necessary table value search and selection help (F4) for a selection query.[2] By calling the underlying RFCs associated with transaction SE16, all the data browsing functionality users have become accustomed to is provided in your custom application. The user-selected data is captured in an internal table, then transferred from the SAP server to Excel using DOI functionality. DOI is used to create the Excel instance from the SAP server. In the created Excel object, the transferred data is loaded and the spreadsheet is formatted using Microsoft Visual Basic for Applications (VBA).

[1] In SAP Release 4.5B, with hot pack 35, the maximum display width is increased to 1,023 characters. The help documentation has not been updated and states the maximum display width is 255 characters.

[2] Single value, range value, and selection operator specification support is provided.

Figure 1 Data Browser (Transaction SE16) Selection Screen



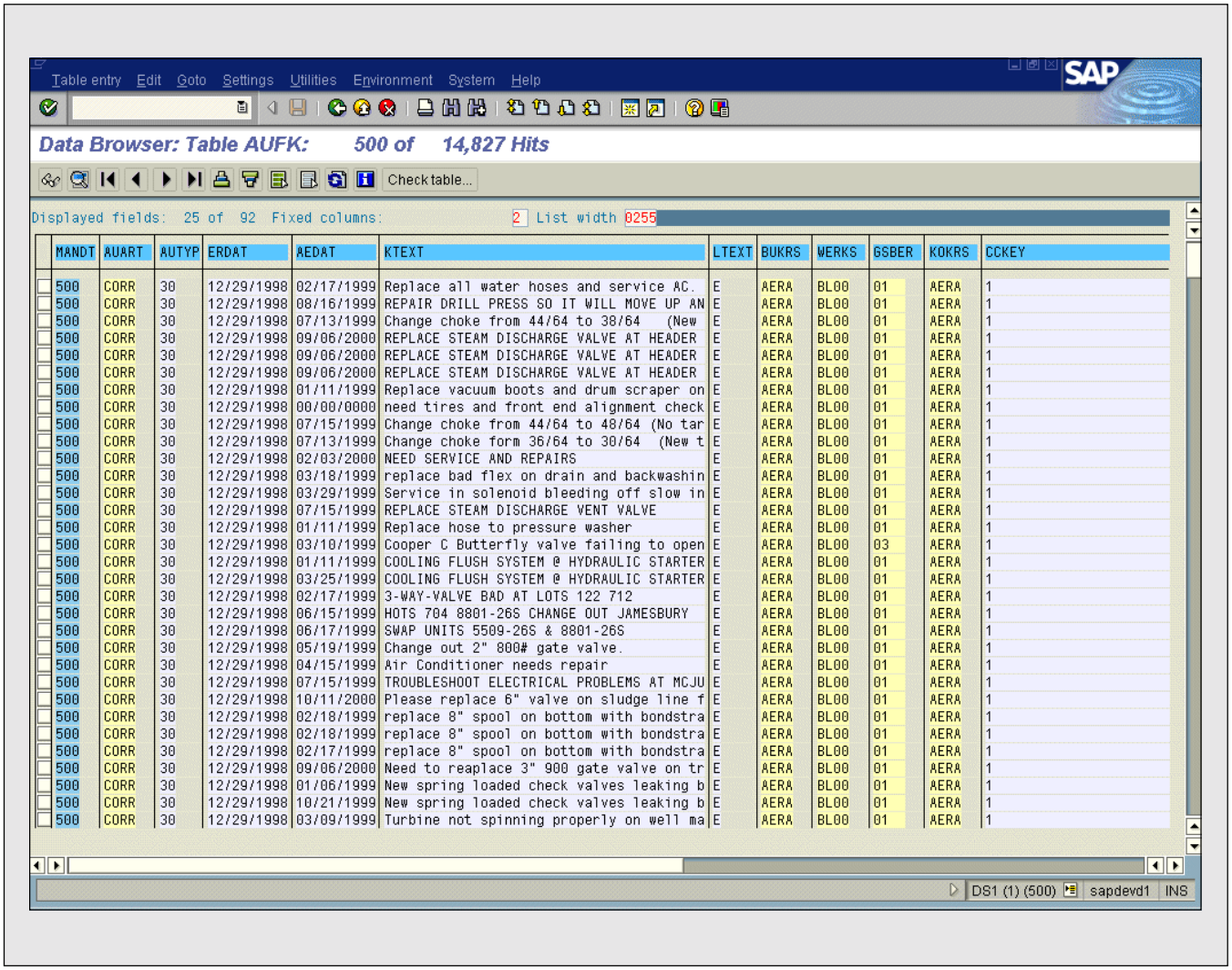
This article provides a hands-on example of DOI programming for readers who are interested in DOI, have ABAP programming knowledge, and have at least an understanding of Microsoft's Visual Basic for Applications (VBA).[3] In it, I will show you step by step how to:

1. Provide users with SAP Data Browser functionality to select *all* the data that they need through the `RS_TABLE_LIST_CREATE` function call.
2. Dynamically create an internal table definition and load the internal table with user-selected data.
3. Transport the data from the server to an Excel instance on the client computer via the DOI table collection.
4. Create the Microsoft Visual Basic code required to support data downloads.

Before delving into the details of these steps, let's take a glance at what this functionality looks like from the end user's perspective.

[3] This example, developed on an SAP Release 4.5B system, transfers data from SAP and displays it in a formatted Microsoft Excel spreadsheet. The installation requirements for this application are SAP GUI Release 4.x and Microsoft Excel 97 installed on your PC. Excel has limitations of 255 columns, 65,000 rows, and a recommended 3 MB file size limit.

Figure 2 Data Display Report — 25 of 92 Fields Are Displayed

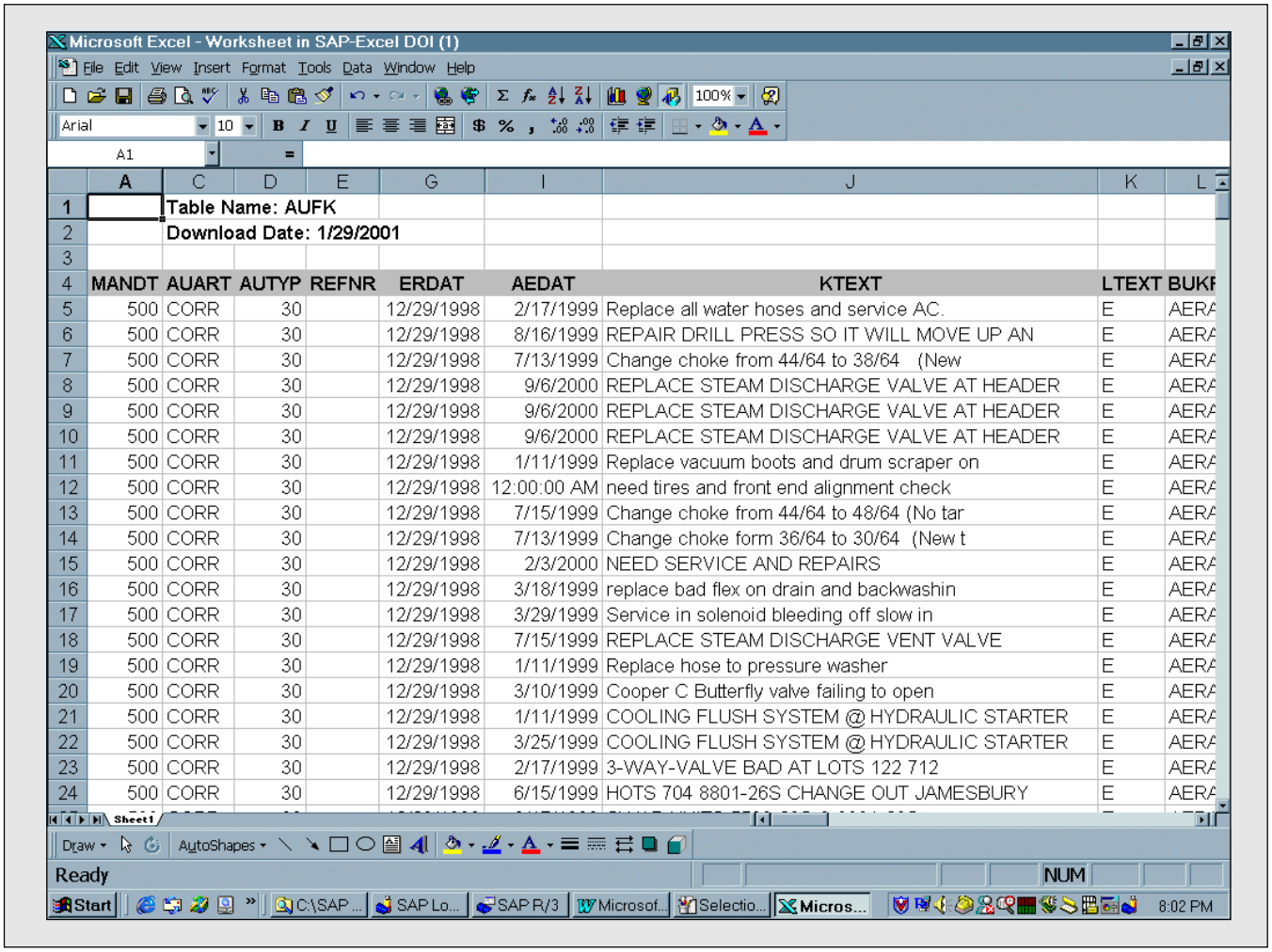


The Process at a Glance

Suppose a user — let's call him Bernd — wants to extract some purchase order data. He runs transaction SE16 and enters "AUFK" (order master data) in the initial "Table Name" screen. Bernd is then presented with a data selection screen (**Figure 1**), which is where the data to be returned by SAP is defined. He then populates the selection fields, such as "AUFNR" and "AUART," with selection criterion, as shown in Figure 1. The report width and the maximum number of records to return can also be configured here.

On program execution, the selected data is displayed in the form of an ABAP report (**Figure 2**). Usually, if Bernd wants to manipulate the data for further analysis, he has to download the data to his local PC through SAP's list-saving support — i.e., **System → List → Save → Local file**. The download supports a number of file formats, but regardless of which format is selected, only the data displayed in the report will be downloaded. The example in Figure 2 alerts us that only 25 of 92 fields are displayed. If Bernd wants all 92 columns displayed, he has to perform about four separate downloads and then merge the data back together. This process not only

Figure 3 Data Display in Excel — All Fields Are Present



Microsoft Excel - Worksheet in SAP-Excel DOI (1)

	A	C	D	E	G	I	J	K	L
1		Table Name: AUFK							
2		Download Date: 1/29/2001							
3									
4	MANDT	AUART	AUTYP	REFNR	ERDAT	AEDAT	KTEXT	LTEXT	BUKR
5	500	CORR	30		12/29/1998	2/17/1999	Replace all water hoses and service AC.	E	AERA
6	500	CORR	30		12/29/1998	8/16/1999	REPAIR DRILL PRESS SO IT WILL MOVE UP AN	E	AERA
7	500	CORR	30		12/29/1998	7/13/1999	Change choke from 44/64 to 38/64 (New	E	AERA
8	500	CORR	30		12/29/1998	9/6/2000	REPLACE STEAM DISCHARGE VALVE AT HEADER	E	AERA
9	500	CORR	30		12/29/1998	9/6/2000	REPLACE STEAM DISCHARGE VALVE AT HEADER	E	AERA
10	500	CORR	30		12/29/1998	9/6/2000	REPLACE STEAM DISCHARGE VALVE AT HEADER	E	AERA
11	500	CORR	30		12/29/1998	1/11/1999	Replace vacuum boots and drum scraper on	E	AERA
12	500	CORR	30		12/29/1998	12:00:00 AM	need tires and front end alignment check	E	AERA
13	500	CORR	30		12/29/1998	7/15/1999	Change choke from 44/64 to 48/64 (No tar	E	AERA
14	500	CORR	30		12/29/1998	7/13/1999	Change choke form 36/64 to 30/64 (New t	E	AERA
15	500	CORR	30		12/29/1998	2/3/2000	NEED SERVICE AND REPAIRS	E	AERA
16	500	CORR	30		12/29/1998	3/18/1999	replace bad flex on drain and backwashin	E	AERA
17	500	CORR	30		12/29/1998	3/29/1999	Service in solenoid bleeding off slow in	E	AERA
18	500	CORR	30		12/29/1998	7/15/1999	REPLACE STEAM DISCHARGE VENT VALVE	E	AERA
19	500	CORR	30		12/29/1998	1/11/1999	Replace hose to pressure washer	E	AERA
20	500	CORR	30		12/29/1998	3/10/1999	Cooper C Butterfly valve failing to open	E	AERA
21	500	CORR	30		12/29/1998	1/11/1999	COOLING FLUSH SYSTEM @ HYDRAULIC STARTER	E	AERA
22	500	CORR	30		12/29/1998	3/25/1999	COOLING FLUSH SYSTEM @ HYDRAULIC STARTER	E	AERA
23	500	CORR	30		12/29/1998	2/17/1999	3-WAY-VALVE BAD AT LOTS 122 712	E	AERA
24	500	CORR	30		12/29/1998	6/15/1999	HOTS 704 8801-26S CHANGE OUT JAMESBURY	E	AERA

is time-consuming, but also is error-prone because merging data can be tricky. It is not uncommon to misalign the separate uploads of data and consequently to have an upload with bad records.

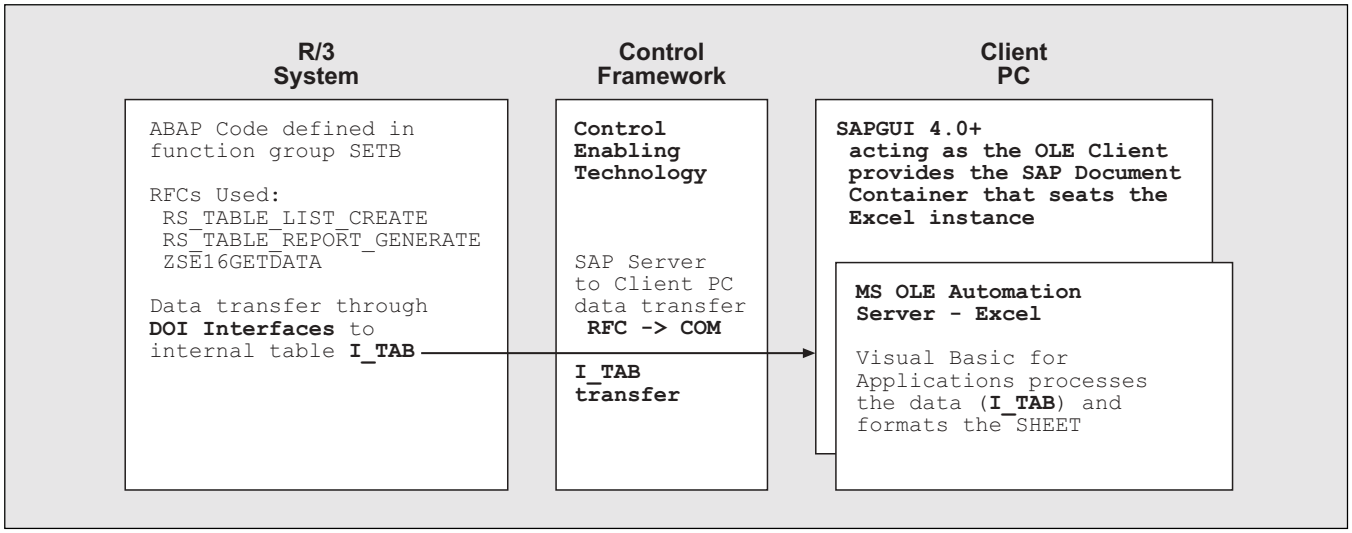
Using our DOI-supported program, Bernd can use the exact same process and selection screen, and he can have all the selected table data transferred to a formatted Excel spreadsheet, as shown in **Figure 3**. There is no need for performing or merging multiple downloads of data. Bernd doesn't need to search for the download file on his PC, or upload the data into Excel, or format the document. He can simply start using the data in his preferred editor.

So, how does this DOI program work?

The program architecture is shown in **Figure 4**. The program can be broken up into three main areas:

1. Data Browser support is captured by using the same RFC function calls as those called in transaction SE16. The needed function modules are located in SAP's function group SETB and can be accessed through the Repository Browser, transaction SE80. Simply call the RS_TABLE_LIST_CREATE function module and let it call whatever other functions (e.g., RS_TABLE_REPORT_GENERATE) are needed to support Data Browser functionality — this is done automatically as part of the function module's built-in logic.

Figure 4 *The Program Architecture*



2. The Data Browser hands off the selected data to DOI support through SAP's Control Enabling Technology for transfer to the client PC. DOI functionality creates the Excel instance on the user's local PC and hands off the data to Excel.
3. Excel VBA code examines the local DOI-populated objects and presents the data to the user.

Let's take a closer look at the steps involved in creating this DOI-supported program to enable your end users, like Bernd, to select and download all the data they need and easily display it in Excel.

Step 1: Enabling Users to Select All the Data They Need

SAP's Data Browser, transaction SE16, supports runtime table selection and user-defined selection options. This is accomplished by generating an on-the-fly data-displaying program that captures the user's data selections. Generated programs are stored for reuse. This functionality is encapsulated in function group SETB. The two core functions that provide the majority of the SE16 functionality are:

- RS_TABLE_LIST_CREATE
- RS_TABLE_REPORT_GENERATE

RS_TABLE_LIST_CREATE is the function that provides the user with dynamic table querying support. The user can select table fields to use in querying the database. F4 field-level search help is defined in the table definition, along with criterion selection operators (<>, <, >, etc.), to aid in limiting the returned data. Together, search help and criterion-limiting operators allow the user to define table-specific data selection queries. This is how, in the previous example, Bernd was able to get his order master data.

An ABAP program is generated that includes the user-defined selection fields. To generate a new program, the RS_TABLE_LIST_CREATE function calls the RS_TABLE_REPORT_GENERATE function. If the user changes the selection fields, the program is regenerated automatically; otherwise, the currently stored program runs.

As far as Bernd is concerned, he can change the fields on which he is setting data selection restrictions whenever he wants on any table he specifies. As programmers, we simply want to pick up on this dynamic selection functionality, which we do by letting the Data Browser associated function modules generate the "on-the-fly" programs whenever new programs are needed. Programmers can think of function group SETB as a Data

Listing 1: Calling a Custom Function to Transfer Data

```
35      CALL FUNCTION 'RS_TABLE_LIST_CREATE'
36         EXPORTING
37              TABLE_NAME         = DATABROWSE-TABLENAME
38              DATA_EXIT          = 'ZSE16GETDATA'
39         EXCEPTIONS
```

Listing 2: Custom Function ZSE16GETDATA

```
FUNCTION ZSE16GETDATA.
*"----------------------------------------------------------------------
*"*"Local interface:
*"       IMPORTING
*"              VALUE(TABNAME) TYPE  ANY
*"       TABLES
*"              DATA TYPE  TABLE
*"       EXCEPTIONS
*"              GENERAL_FAIL
*"              TABLE_IS_EMPTY
*"----------------------------------------------------------------------
DATA: MEM_ID(16),
      S_NAME LIKE SY-UNAME.

* Output data to memory for pickup by calling program.
  CONCATENATE SY-UNAME  'DOI' INTO MEM_ID.
  EXPORT DATA TO MEMORY ID MEM_ID.

ENDFUNCTION.
```

Browser object that supplies them with the user-selected data.

The RS_TABLE_LIST_CREATE function provides us with the ability to transfer the selected data to Excel through its data_exit parameter, as shown in **Listing 1** (the full code listing of ZBTABLELISTGENERATION, the example program, is available at **www.SAPpro.com**, with line numbers added for your reference). The data_exit parameter is an importing parameter that specifies a function to call from within the RS_TABLE_LIST_CREATE function. We can access all the data structures of the SETB function group through the data_exit parameter-specified custom function. Our custom function ZSE16GETDATA, shown in **Listing 2** (also available at **www.SAPpro.com**), exports the selected table data (which is stored in a structure called DATA) to memory.

All that function ZSE16GETDATA does is place the user-selected data in memory for later retrieval. The newly created RFC function, ZSE16GETDATA, is called from function RS_TABLE_LIST_CREATE. The data is copied to a memory location identified by MEM_ID.

Our custom Data Browser functionality is implemented in the example program

Listing 3: Defining a Dynamically Generated Program

```
 1 data: i_code(72) occurs 10,
 2       msg(120), lin(3), wrd(10), off(3), s_buff(72).
 3
 4   append 'PROGRAM SUBPOOL.' to i_code.
 5   concatenate 'DATA I_TAB like' databorwse-tablename 'OCCURS 0'
 6               Into s_buff separated by space.
 7   append s_buff to i_code.
 8   append 'INCLUDE ZIOIEXCEL.' to i_code.
 9   append 'FORM DYN1.' to i_code.
10   write 'IMPORT DATA TO I_TAB FROM MEMORY ID ''01'' ' to s_buff.
11   append s_buff to i_code.
12   append 'DATA h_oiexcel type ref to coiexcel.' to i_code.
13   append 'CREATE OBJECT h_oiexcel.' to i_code.
14   append 'CALL METHOD h_oiexcel->launchse16.' to i_code.
15   append 'ENDFORM.' to i_code.
```

ZBTABLELISTGENERATION. A copy of table selection screen 230 — "Enter table name" — is made from function group SETB. In screen 230's PAI USER_COMMAND module, we plug in the Data Browser functionality by calling the RS_TABLE_LIST_CREATE function. Since we defined the data_exit parameter (Listing 1), the custom function is run and selected data is exported to memory. Basically, all the functionality that is included in transaction SE16 is captured. Field selection, configuration support, user authority verification, table structure validation, support for table views, transparent tables, pooled tables, and cluster tables is provided. The ABAP report is not displayed because the data_exit parameter is filled.

> ✓ ***Tip***
>
> *You can simplify the program by skipping screen 230 and associated modules. Use a parameters statement, such as:*
>
> `p_tblnme like databrowse-tablename.`
>
> *All you will miss is the initial feel of SE16's table selection screen, the user parameters settings, and online manual help.*

> ✓ ***Tip***
>
> *Create the custom function module through transaction SE37.*

Step 2: Dynamic Internal Table Definition — Creating a Program Dynamically

With the data in memory, we now need to load an internal table with the data that is to be transported via DOI to Excel. The problem is that SAP doesn't support runtime table structure definition, and the user is selecting the table that is to be queried at runtime! We need to dynamically define an internal table into which the selected data can be loaded, and then transfer that table to our Excel instance.

Using ABAP dynamic subroutine pool creation commands, we can change the internal table definition to runtime and then call the required DOI-related functionality, passing in the newly defined and loaded internal table. For the sake of clarity and discussion, the example code in **Listing 3** is slightly different

Listing 4: Dynamic Program Generation

```
164   GENERATE SUBROUTINE POOL I_CODE NAME 'DOI_SE16'
165     MESSAGE msg
166       LINE lin
167         WORD wrd
168           OFFSET off.
```

from the actual program code.[4] Appending ABAP code into an internal table structure is intrinsically messy, mostly due to the complication of adding constants inside the ABAP command line. Look on line 10 of Listing 3 — we enclose the constant `01` in double apostrophes and use a write statement to load a temporary character buffer, `s_buff`, with our ABAP command before we append the line to `i_code`.

An internal table `i_code` is declared on line 1 in Listing 3. The program to be generated is loaded into the internal table as lines of program code. On line 4, we load the dynamic subroutine pool creation command into `i_code`. The structure to load the user-selected data is defined on lines 5-7 by using a data declaration "like" the user-selected table. By placing the user-selected data type internal table definition in a dynamically created form, we have essentially dynamically declared the internal table. All our DOI-required functionality is encapsulated in a class structure and kept in include file `ZIOIEXCEL`. Line 8 brings in this include. On line 9, we declare the form that will be called to start the DOI-supported data transfer. The data that was placed in memory through function `ZSE16GETDATA` is retrieved on line 10. A reference to our DOI encapsulating class, `COIEXCEL`, is defined on line 12. An instance of class `COIEXCEL` is created on line 13. With line 14, the DOI-supported functionality is started. In the actual code, the internal table loaded with user-selected data and a file path to a specific Excel file is exported to the `LAUNCHSE16` method of the `COIEXCEL` class. These parameters of method `LAUNCHSE16` are not shown here to keep the explanation simple.

We dynamically create the subroutine pool by calling the code shown in **Listing 4** and passing in the program code in the internal table `i_code`.[5] The generated pool is held in memory, providing us access to our dynamically created form, `FORM DYN1`, which was declared on line 9 of Listing 3. Note that SAP has tagged the `GENERATE SUBROUTINE POOL` command for internal use.

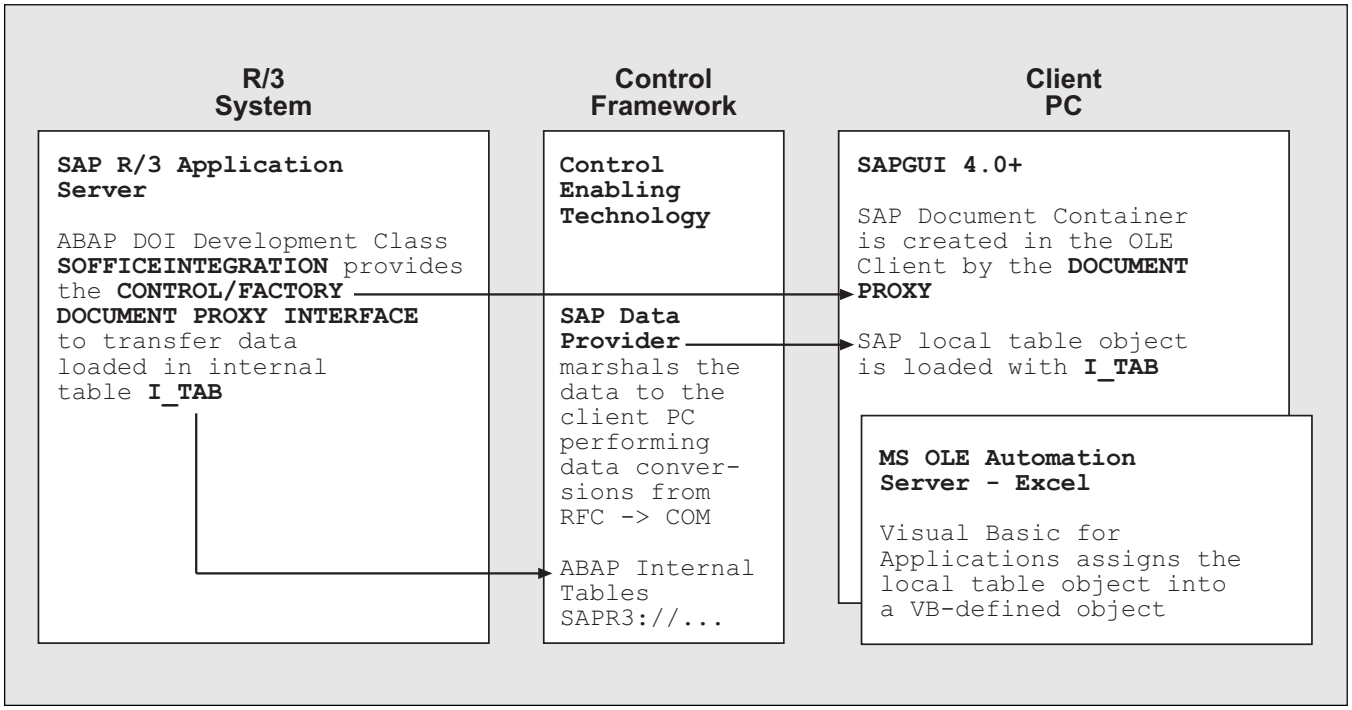
> ***✓ Tip***
>
> *Concatenating a constant into a character string requires the following syntax:*
>
> ```
> write 'XXX ''YY'' ' to s_buff
> ```

> ***✓ Tip***
>
> *Classes, as shown by class COIEXCEL, comprise the framework for object-oriented programming (OOP). A few of the larger principals supported by OOP are data encapsulation, modularization, and code reuse.*

[4] The actual code is listed in the `ZBTABLELISTGENERATION` program (available at **www.SAPpro.com**) in lines 137-172.

[5] This code is also excerpted from the `ZBTABLELISTGENERATION` program, available at **www.SAPpro.com**.

Figure 5 SAP Desktop Office Integration Architecture



Step 3: Transport the Data from the Server to an Excel Instance

Now that the data is loaded, we need to transport it to the Excel instance via DOI methods. SAP DOI methods are implemented as ABAP object-oriented interfaces and classes. The method calls are transmitted via Remote Function Call (RFC) to the client PC using the SAP Control Framework. The Control Framework provides the infrastructure for communication between the ABAP program running on the SAP application server and the OCX[6] controls running on the client PC. On the client PC, DOI is represented as a set of OCX files that are installed with the SAP GUI.

Figure 5 shows an overview of the relationship between the DOI interfaces, the Control Enabling Technology, and SAP's Component Object Model (COM) compliant controls. The DOI interfaces use the Control Enabling Technology to create an SAP Document Container Control that acts as a client hosting Microsoft's OLE Automation Server, Excel. After DOI has instantiated the Control Framework and the Container Control on the client PC, the data selected by our friend Bernd is passed through the framework to Excel using the SAP Data Provider OCX.

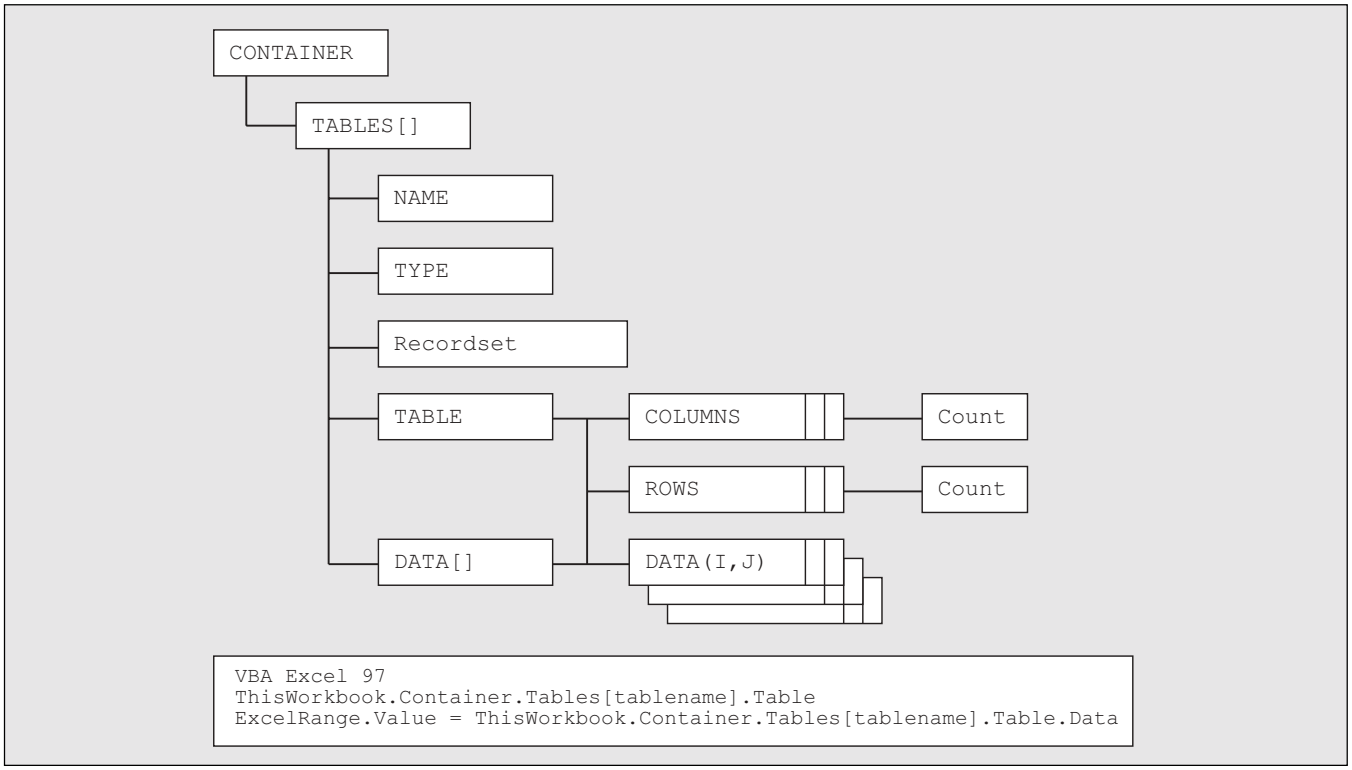
This article examines just one small part of the Desktop Office Integration functionality.[7] We are simply creating an external instance of Excel from SAP and transferring an internal table of selected data to the Excel instance. To do this, we need a reference to one of the DOI starter interfaces, a reference to the DOI table collection interface, and a reference to the DOI Excel document server interface.

To use Desktop Office Integration, you must start with one of the two globally available

[6] OCX is a Microsoft OLE Custom Control. It is a software module that's based on OLE and COM technologies.

[7] For more information on SAP Desktop Office Integration, see Rainer Ehre's article, "SAP Desktop Office Integration (SAP DOI) — An Easier Way for ABAP Programmers to Integrate Desktop Applications with R/3," in the March/April 2000 issue of the *SAP Professional Journal*.

Figure 6 SAP DOI Automation Model — Table Collection



starter classes, `C_OI_CONTROL_CREATOR` or `C_OI_FACTORY_CREATOR`. For in-place Excel activation, you would use the first class — `C_OI_CONTROL_CREATOR`. Our focus is on *external* Excel activation, so we use the globally available static call `C_OI_FACTORY_CREATOR=>GET_CONTAINER_CONTROL` to return a reference into the `I_OI_FACTORY_DOCUMENT` reference variable `h_factory`,[8] as shown in **Listing 5**.[9]

Through the `h_factory` reference, we can access other DOI interfaces as needed. In SAP Release 4.6C, the two starter classes and interfaces are combined.[10]

A call to `start_factory`, the factory's interface method, creates (instantiates) the DOI container object (see **Listing 6**). The table collection object and its associated interface, `i_oi_table_collection`, is used to transfer data between SAP and the client PC in the form of an internal table.

At the client PC, the SAP Table Factory OCX provides access to the data through the DOI automation model data component. Parts of the model that are relevant to this example are listed in **Figure 6**. The code at the bottom of the figure details the SAP DOI Automation hand-off to Excel via VBA.

A call to the `get_table_collection` method of the factory interface, as shown in **Listing 7**, returns a reference to the interface `i_oi_table_collection`. The internal table is transported to the client PC through the table collection interface using the `add_table` method, as shown in **Listing 8**.

[8] In-place activation means hosting the server application, Excel, within the SAP GUI framework. With external activation, Excel will be launched in its own window.

[9] The code in this listing, and the following listings in this section (except Listing 9), is excerpted from the `ZIOIEXCEL` include file available at **www.SAPpro.com**.

[10] In SAP Release 4.6A, starter classes `C_OI_CONTROL_CREATOR` and `C_OI_FACTORY_CREATOR` are combined into the single interface `i_oi_container_control`.

Listing 5: Starting Point — Retrieving a Control Framework Handle

```
40     H_FACTORY TYPE REF TO I_OI_DOCUMENT_FACTORY,

63     call method c_oi_factory_creator=>get_document_factory
64          exporting factory_type = 'OLE'
65          IMPORTING FACTORY = H_FACTORY
66                    RETCODE = S_RETCODE.
```

Listing 6: DOI Container Object Creation

```
69     CALL METHOD H_FACTORY->START_FACTORY
70          EXPORTING R3_APPLICATION_NAME = 'SAP-Excel DOI'
71                    REGISTER_ON_CLOSE_EVENT = 'X'
72          IMPORTING RETCODE = S_RETCODE.
```

Listing 7: Navigating to the Table Collection Interface

```
 41   DATA: H_TABLES TYPE REF TO I_OI_TABLE_COLLECTION,

156       CALL METHOD H_FACTORY->GET_TABLE_COLLECTION
157                     IMPORTING TABLE_COLLECTION = H_TABLES
158                               RETCODE = S_RETCODE.
```

Listing 8: Transferring Data from the Server to the Client

```
162 *transfer data to presentation server
163     CALL METHOD H_TABLES->ADD_TABLE
164          EXPORTING TABLE_NAME   = 'ITAB'
165                    TABLE_TYPE   = H_TABLES->TABLE_TYPE_OUTPUT
166                    DDIC_NAME    = S_TABLENAME
167                    DESCRIPTION  = 'Block Data'
168          IMPORTING
169                    RETCODE      = S_RETCODE
170          CHANGING  DATA_TABLE   = I_TAB.
```

The table collection class uses the SAP Control Framework and specifically the SAP Data Provider OCX to move the data from the server to the client PC. Parameter `table_name` is assigned `ITAB` and we use the name when loading the table object inside the Excel document. Transferring data from SAP controls, on the client PC, to Excel is done through VBA coding.

Parameter `ddic_name` contains the passed table Data Dictionary name. The `add_table` method

loads the table object's associated recordset "fields" collection with table column specifications such as field name, size, precision, type, and numeric scale using the `ddic_name` specification. Inside Excel, we query this recordset and configure the spreadsheet columns accordingly. Parameter `data_table` passes the table query data.

Another and more explicit way to transfer internal tables from the SAP application server to a client PC is to call SAP's Data Provider function `DP_CREATE_URL_FROM_TABLE`. The internal table is transported to the client PC. Access to the transferred data is provided in the returned URL variable. The Data Provider object controls data conversion and maps the data to standard window data types. A unique URL name that takes the form of "SAPR3://<GUID>" is returned. The data is added to the table collection with a call to the table collection method `add_table_by_url` (see **Listing 9**).

To create an instance of Excel, we need a reference to the DOI OLE2 automation server object. The reference is returned by querying our container interface `h_factory` (**Listing 10**). The `DOCUMENT_TYPE` parameter is a Windows registry entry stored under the `HKEY_CLASSES_ROOT` directory. It represents the program ID of a valid ActiveX document server application.

To keep things simple, we are going to open an existing Excel document that already has the required Visual Basic code attached in the Excel startup macro `Module1.LoadR3Data` (**Listing 11**).[11]

The document proxy interface `open_document` method accepts a `DOCUMENT_URL` parameter that represents the address of the document to open — in this case, "FILE://C:\Temp\TestRs.xls". The document URL can point to an HTTP server, an FTP server, a UNIX directory, or a local PC. Parameter `STARTUP_MACRO` specifies a macro to be run when the Excel document is opened. The R/3-to-Excel data transfer is controlled by the startup macro.

[11] A sample Excel document, TestRs.xls, is available for download at **www.SAPpro.com**.

Class `COIEXCEL`, defined in include file `ZIOIEXCEL`, encapsulates our required DOI functionality to provide tighter code modularization and reuse. The class is designed to simplify and extend our use of DOI. Our overall plan is to let the constructor/destroy methods of the class take care of Control Framework support. Simplified methods wrap the DOI table creation, loading, and transfer support. Methods for adding a table to the table collection, transferring a table to the client PC, and opening an Excel document are defined. SAP DOI-associated includes and required function pool definitions are defined in the class. Note that in SAP Release 4.0 through 4.5, the DOI interface and class definitions are defined in the `OFFICEINTEGRATIONINCLUDE` file. In Release 4.6, the definitions are defined globally, so the `OFFICEINTEGRATIONINCLUDE` file is not required.

To review, user-selected data was exported to memory through `ZSE16GETDATA`, our custom function. We placed the class object instantiation of `COIEXCEL` in the dynamic program implementation section (Listing 3), thus creating the DOI control object. Class `COIEXCEL` wraps the DOI required calls for ease of use and reuse. The control framework used by DOI is created in class `COIEXCEL`'s constructor. A call to class `COIEXCEL` method `LAUNCHSE16` kicks off the rest of the DOI process by implementing DOI-required calls. A table object named `ITAB` loaded with the user-selected data is transferred to the client PC. The SAP side of this process ends by opening a specified Excel document and calling an Excel startup macro, `Module1.LoadR3Data` (Listing 11).

✓ *Tip*

Class COIEXCEL's constructor creates the Control Framework and sets the reference to the document proxy.

Memory cleanup is performed in class COIEXCEL's destroy method.

Listing 9: Transferring Data Through a URL Specification

```
call function 'DP_CREATE_URL_FROM_TABLE'
         exporting tabname  = ''
         tables data = i_tab
   changing url      = s_docurl

call method h_tables->add_table_by_url
         exporting url     = s_url
                   i_tab      = i_tab
         importing retcode = s_retcode
```

Listing 10: Creating and Retrieving a Reference to a Document Proxy

```
42     DATA: H_DOCUMENT TYPE REF TO I_OI_DOCUMENT_PROXY,

74     CALL METHOD H_FACTORY->GET_DOCUMENT_PROXY
75       EXPORTING DOCUMENT_TYPE = 'EXCEL.SHEET.8'
76       IMPORTING DOCUMENT_PROXY = H_DOCUMENT
77             RETCODE = S_RETCODE.
```

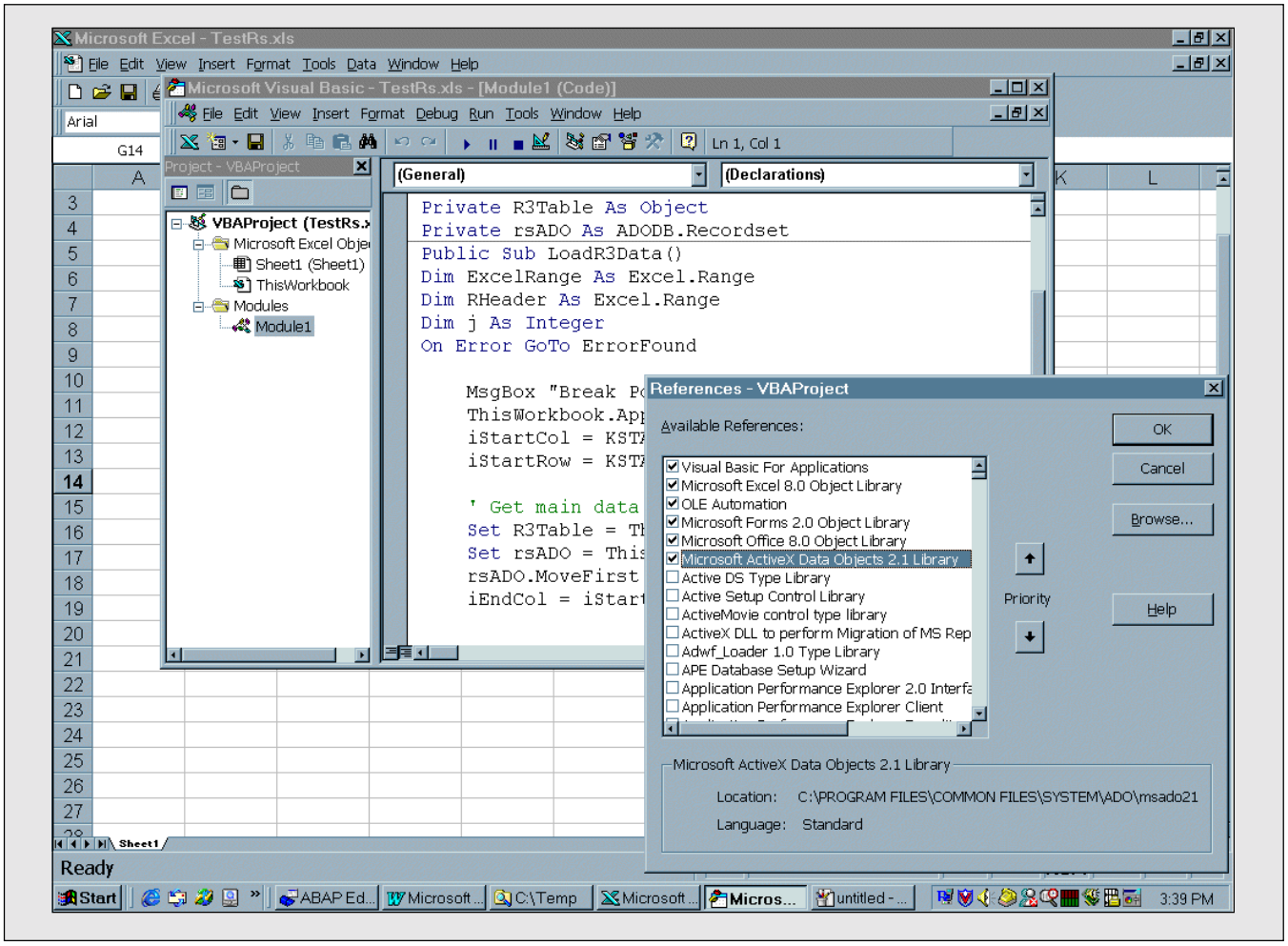
Listing 11: Opening an Existing Document at a Specified URL

```
139     CALL METHOD H_DOCUMENT->OPEN_DOCUMENT
140            EXPORTING
141                     DOCUMENT_URL = S_FILEURL
142                     OPEN_INPLACE = ' '
143                     STARTUP_MACRO    = 'Module1.LoadR3Data'
144          IMPORTING RETCODE = S_RETCODE.
```

Step 4: Create the Visual Basic Code to Support Data Downloads

Now that the data is transported to the Excel instance, we next need to create the VB code to support the data downloads. Visual Basic Desktop Office Integration support is defined in the `LoadR3Data` subroutine. `LoadR3Data` formats the Excel spreadsheet and does a block transfer of data from the DOI table data object into a defined Excel range object. Most of the code listed in this section is taken from the `LoadR3Data` subroutine, which is available as part of the sample Excel file TestRs.xls.

Figure 7 Excel VBA — Setting a Reference to the ActiveX Data Objects Library



The DOI recordset object is used to define the column names. We can get a reference to the recordset object by assigning it to a generic VBA object, as follows:

```
Dim RS As Object
Set RS = ThisWorkbook.Container.
  Tables("ITAB").Recordset
```

Alternatively, we can assign the reference to a typed Microsoft ADODB[12] recordset object. By assigning the reference to a typed object, we get improved performance, type checking, and Microsoft's automatic object methods and parameter help associated with the object. Before we can declare an ADODB typed object, we need to add a reference to the ADODB library in the VBA project. In the Visual Basic editor, with our project open, we select **Tools** → **References**, and then select "Microsoft ActiveX Data Objects 2.x Library," as shown in **Figure 7**.

Then we declare the object and assign the reference:

```
Private rsADO As ADODB.Recordset
Set rsADO = ThisWorkbook.Container.
  Tables("ITAB").Recordset
```

[12] Microsoft's ActiveX Data Objects for accessing OLE DB data sources.

The recordset object contains all the DOI table object information, but in a model that is Microsoft-compliant. In the `LoadR3Data` subroutine, we simply extract the field names and width to use in formatting the spreadsheet columns. Data type conversions and formatting are handled automatically for us by the SAP Data Provider OCX and by Microsoft Excel. The recordset object does provide us with all the information necessary to explicitly handle grid column typing ourselves.

To load Excel with the internal table data, we first declare an `ExcelRange` object and assign the proper Excel sheet cells defining the range of cells that make up our block. Then we perform a block data transfer:

```
Dim ExcelRange As Excel.Range
Set ExcelRange = Sheet1.
  Range(Sheet1.Cells(iStartRow,…
Set ExcelRange.Value = ThisWorkbook.
  Container.Tables("ITAB").Data
```

✓ Tip

If you write some VBA code, there will probably come a time when you will want to turn on the VBA debugger and step into the code. The VBA debugger can be enabled by adding a MsgBox call at the line you want to start debugging, and entering <CTRL-BREAK> when the message prompt appears. Another procedure is to rerun the startup macro. From Excel, go to ***Tools → Macro → Macros****. Select the subroutine and the "Step Into" option.*

✓ Tip

The ADODB recordset object contains a collection of field objects and is an iterator over the downloaded data. It supports extensive functionality and is a central component of Microsoft's programming architecture.

Conclusion

SAP Desktop Office Integration offers an extensible option for integrating SAP R/3 with various office applications. The assortment of target applications includes all OLE2-compliant servers such as Microsoft Word 97, Microsoft Excel 97, and others. The document proxy interface, `i_oi_document_proxy`, only works with ActiveX document servers, limiting extensibility to ActiveX document server applications. Of course, this does include custom-developed Visual C++ and Visual Basic ActiveX document server applications that can wrap most anything.

The versatility of the Desktop Office Integration interfaces is demonstrated by the ease with which the example discussed here can be ported to another ActiveX document server. Simply changing the `h_factory->get_document_proxy DOCUMENT_TYPE` parameter from `'EXCEL.SHEET.8'` to `'WORD.DOCUMENT.8'` switches the target application from Excel to Word. We then need to point the `h_document-> open_document DOCUMENT_URL` to the document we want to open. Of course, the document's associated VB code would need to be modified.

Combining SAP Desktop Office Integration with popular office application functionality like Microsoft Word and Microsoft Excel is a common user request. DOI launches the application locally on the user's PC, giving the user all the office application functionality he or she is accustomed to. This includes printing on Windows system printers, saving files locally or to mapped drives, and sending documents through Microsoft Exchange. Users are able to step into a different environment with a document that was formatted for them.

SAP Release 4.6 introduced the SAP List Viewer (ALV) Grid Control, which offers an alternative method for transferring data from the SAP application server to an Excel instance on the client PC. The ALV Grid Control, and the DOI interfaces, are built around SAP's Control Framework technology. DOI

For More Information…

The premier articles on SAP Desktop Office Integration are those written by Rainer Ehre, the development manager of ABAP component integration at SAP. "SAP Desktop Office Integration Using ABAP Objects" in the *SAP Technical Journal* gives a high-level view of the Desktop Office Integration classes and focuses on a sample program provided by SAP. This journal has been discontinued, but the article is archived on the *Intelligent ERP* Web site at **www.intelligenterp.com/feature/archive/ehre.shtml**. A second article, "SAP Desktop Office Integration (SAP DOI) — An Easier Way for ABAP Programmers to Integrate Desktop Applications with R/3," appears in the March/April 2000 issue of the *SAP Professional Journal*. This article provides an excellent explanation of DOI architecture and how to use it in an SAP 4.x environment.

Desktop Office Integration source code and a number of sample programs are available for viewing in the repository browser by displaying development class `SOFFICEINTEGRATION`. DOI R/3 4.x online help is located at **Help** → **SAP Library** → **Basis Components** → **Component Integration** → **Desktop Office Integration**.

SAP Web-based help can be found at **http://help.sap.com**. From this Web site, drill down to the DOI online help by following the same path as above. To go directly to the DOI Web help pages, use the following URL:

http://help.sap.com/saphelp_45b/helpdata/en/e9/0bee9f408e11d1893b0000e8323c4f/frameset.htm

is targeted toward integrating office applications with SAP, while ALV tries to extend the SAP GUI by providing a richer set of desktop controls. ALV offers ABAP programming extensions through the global classes `g_container` and `g_grid`, along with a number of control containers such as `CL_GUI_CUSTOM_CONTAINER` and `CL_GUI_DOCKING_CONTAINER`. The ALV Grid Control is not offered in the SAP Release 4.6C Data Browser, but through VBA code we could emulate the ALV functionality in our external Excel instance.

Desktop Office Integration offers much more power than what was demonstrated in the example presented in this article. The office application can be launched inside the SAP presentation server frame — SAP terminology calls this *in-place activation*, while Microsoft calls it *in-situ activation*. Through in-place activation, the office user interface becomes part of the SAP GUI. DOI provides support for events and bidirectional data transfers in the table collection interface. There are a number of DOI interfaces available that have not been mentioned in this article (see the resources mentioned in the sidebar above for further information).

Philip Bremner holds a B.S. in Industrial Engineering from the State University of New York at Buffalo, and a B.S. in Computer Science from California State University, Bakersfield. He is a Microsoft Certified Professional holding nine Microsoft certifications, including two Visual Basic certifications. Currently he is a senior programming analyst with Aera Energy LLC in Bakersfield, California, where he is acting as an information technology lead in the company's Enterprise Architecture Implementation project. Phil can be reached at pbremner@aeraenergy.com.

The views expressed in this article are those of the author and not of Aera Energy LLC.

