Chapter 1

Accessing SAS Data without Using SAS Code

1.1 Abstract

Recent developments in SAS for Windows have provided users with routes to SAS data and applications without having to write SAS code using SAS. This chapter describes three examples of these interfaces: ODBC, DDE, and SAS Integration Technologies, which could place SAS at the center of any application development for the Windows platform.

1.2 Introduction

In the past, SAS has been used to read data from other Windows data sources, e.g., Microsoft Access tables using SAS/ACCESS for ODBC, and to control other external Windows applications using Dynamic Data Exchange (DDE). SAS is, of course, available as a Windows application itself and can now be used as an external application for those other Windows applications. This role reversal expands the range of uses for SAS in the Windows environment in areas where SAS has not been traditionally the first-choice application. The ability of SAS to read and maintain data from a wide range of sources can now be used throughout the Windows arena.

Further discussion on SAS Enterprise Guide, a thin-client application for the Windows client platform that uses SAS Integration Technologies to communicate with SAS installations on remote servers, can be found in Chapter 6 “Developing SAS Applications Using SAS Enterprise Guide.”

1.3 ODBC

The SAS ODBC driver has been supplied with Base SAS for Windows since SAS 6.10 to provide an interface to SAS data libraries for other Windows applications. Each application has its own particular uses and limitations for the ODBC interface. This section describes the practicalities of using the SAS ODBC driver 9.1 with Microsoft

Access 2000, Microsoft Excel 2000, Visual Basic 6.0, Lotus Approach Version 9, and OpenOffice.org 2.1. It should be noted here that StarOffice 8 is functionally equivalent to OpenOffice.org 2.1, and so all future references to OpenOffice.org 2.1 can be assumed to include StarOffice 8.

Single ODBC access to SAS data on the same machine that the user accesses uses the ODBCSERV procedure, which is supplied with Base SAS, running in a single SAS region. Multiple ODBC access to SAS data, or ODBC access to a remote machine, requires SAS/SHARE, and possibly SAS/SHARE*NET as well.

1.3.1 Setting Up a SAS Server for the SAS ODBC Driver

It is very important to plan, in advance, which SAS data libraries will be accessed via the SAS ODBC driver, as the LIBNAME statements must be defined using the ODBC Administrator application by selecting **Start ▶ Control Panel ▶ Administrative Tools ▶ Data Sources (ODBC)**. In particular, for any ODBC data source, there can be only one library reference that can be written to by an external application, i.e., USER, as Microsoft Access and similar applications can write to data sets with a single-level data set name only. This name, say XYZZY, would be assumed to be the data set WORK.XYZZY, except that the USER library name will override the normal default WORK library name, allowing permanent SAS data sets to be created whenever single-level names are used.

Other features of the ODBC data source definitions include the following:

- The SAS ODBC server must be added to the SERVICES file (found in C:\WINDOWS or C:\WINNT\SYSTEM32\DRIVERS\ETC, depending on the Windows platform used) prior to using the ODBC Administrator, as the SAS ODBC driver uses a TCP/IP connection to communicate with the SAS ODBC server. The additional lines should look like the following line, with a unique number greater than 1024 and the columns separated by tab characters:

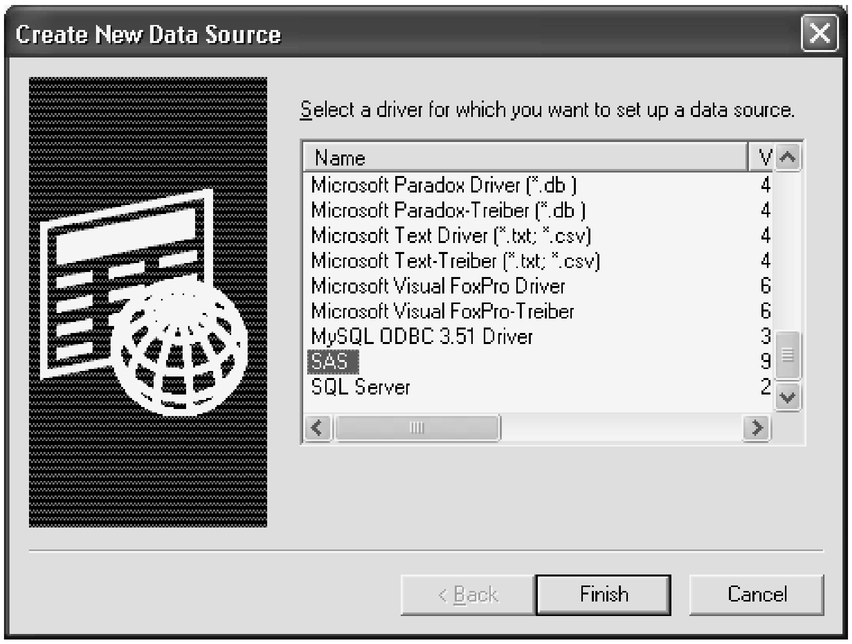
```
sasuser32     7001/tcp      #SAS OBDC Server
```

- Command line options when invoking SAS (e.g., -AUTOEXEC, -NOLOGO, etc.).
- SAS data library names used for importing into external Windows applications.
- Changes to the library references in a running SAS ODBC server can be made for subsequent ODBC connections.
- Library references within SAS 8 are limited to eight characters, which are not case sensitive. The names cannot include blanks or punctuation, must start with an alphabetic or underscore character, and the second and subsequent characters may be numeric characters.

- It should be noted that the SAS ODBC driver is unable to read the supplied SAS library references (i.e., MAPS, SASUSER, and SASHELP) using its own special library references. If these library references need to be read, they should be allocated to different library references in the ODBC Server Libraries panel, e.g., SMAPS, SUSER, and SHELP.
- Finally, but probably the most important, if you are trying to use the SAS ODBC driver on a Windows platform protected by a personal firewall, because the SAS ODBC server is accessed via a TCP/IP port, you will only be able to access the SAS ODBC server port if explicitly permitted by the personal firewall rules.

The setup procedure is as follows:

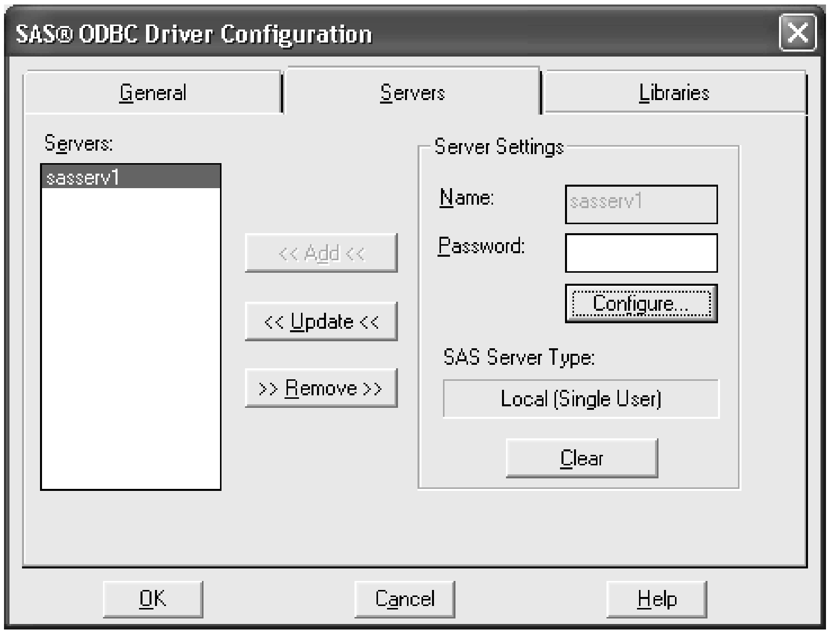
1. Select a user data sources (driver), e.g., **SAS**.
2. Click **Finish**.
3. Select a data source name, e.g., **SASUSER32**.


Create New Data Source

Select a driver for which you want to set up a data source.

Name	V
Microsoft Paradox Driver (*.db)	4
Microsoft Paradox-Treiber (*.db)	4
Microsoft Text Driver (*.txt; *.csv)	4
Microsoft Text-Treiber (*.txt; *.csv)	4
Microsoft Visual FoxPro Driver	6
Microsoft Visual FoxPro-Treiber	6
MySQL ODBC 3.51 Driver	3
SAS	9
SQL Server	2

4. Select a description, e.g., **SAS 9.1 ODBC Server**.
5. Select a server, e.g., **sasserv1**.
6. Click the **Servers** tab.
7. Select a server name, e.g., **sasserv1**.
8. Click **Configure**.

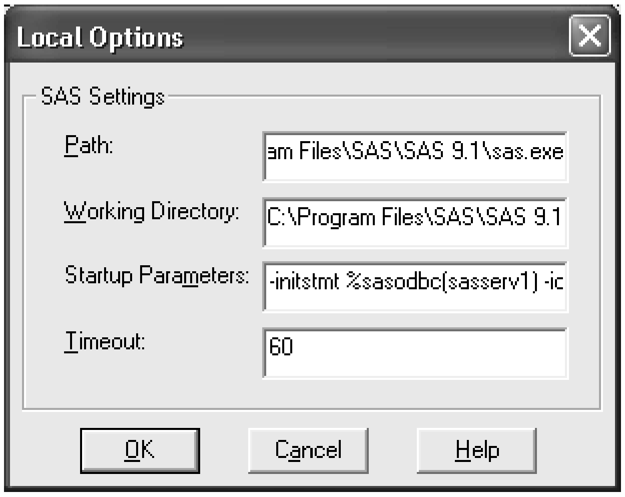

SAS® ODBC Driver Configuration

General | Servers | Libraries

Servers: sasserv1

Server Settings
Name: sasserv1
Password:
SAS Server Type: Local (Single User)

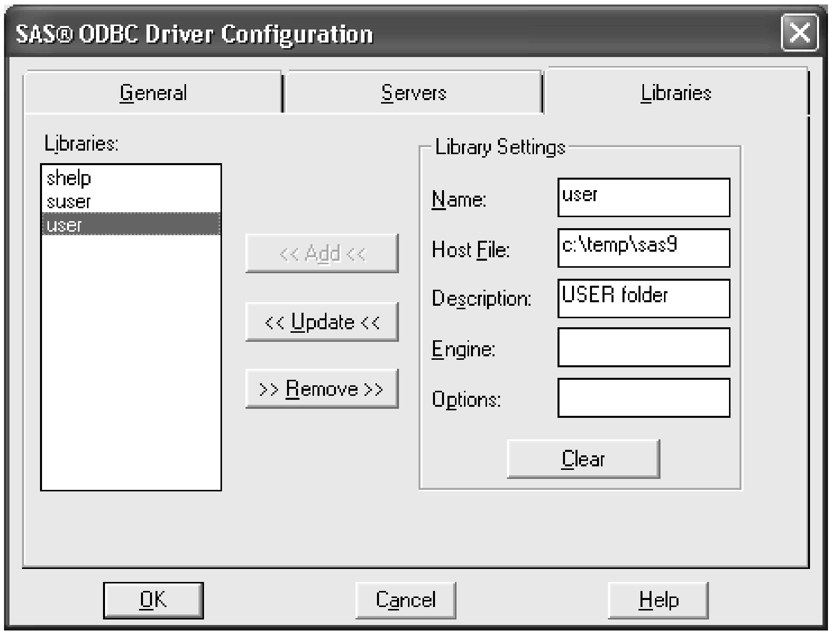
9. Select a SAS path, e.g., **C:\Program Files\SAS\SAS 9.1\sas.exe**. Note that this path can be used to determine which version of SAS is to be used. The SAS ODBC driver 8.2 can support SAS servers of SAS 6, 7, or 8, and the SAS ODBC driver 9.1 can support SAS servers of SAS 7, 8, or 9.
10. Select a SAS parameter, e.g., **-initstmt %sasodbc(sasserv1) -icon -nologo**.
11. Click **OK**.



Path: am Files\SAS\SAS 9.1\sas.exe
Working Directory: C:\Program Files\SAS\SAS 9.1
Startup Parameters: -initstmt %sasodbc(sasserv1) -ic
Timeout: 60

12. Click **Update** or **Add**.
13. Click the **Libraries** tab.

14. Select a library name, e.g., **user**.
15. Select a host file name, e.g., **c:\temp\sas9**.
16. Select a description, e.g., **USER folder**.
17. Click **Add** or **Update**.



Libraries:
shelp
suser
user

Library Settings
Name: user
Host File: c:\temp\sas9
Description: USER folder
Engine:
Options:

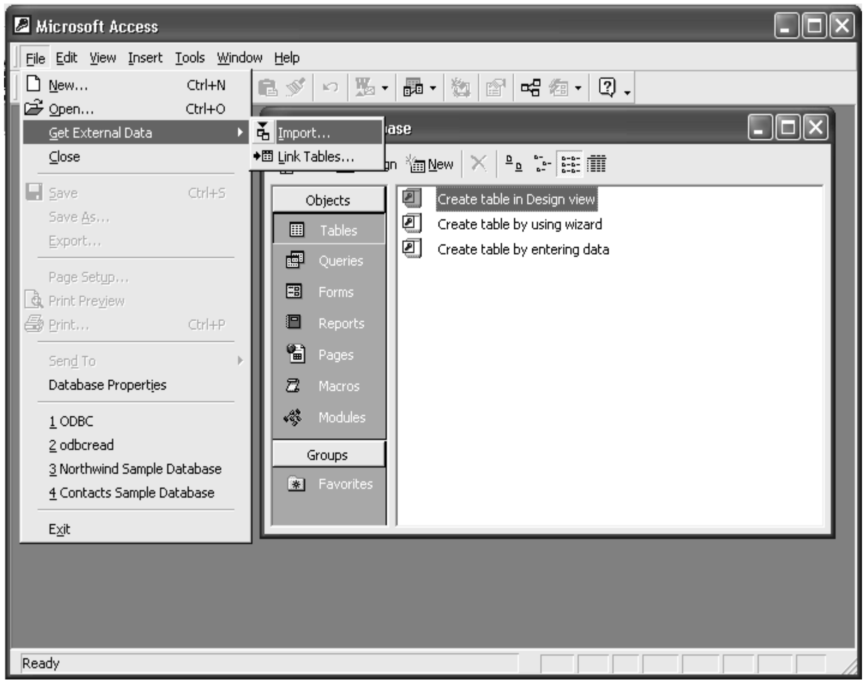
18. Repeat selections as required.
19. Click **OK**.
20. The SAS ODBC driver is now set up for use.

1.3.2 Microsoft Access 2000

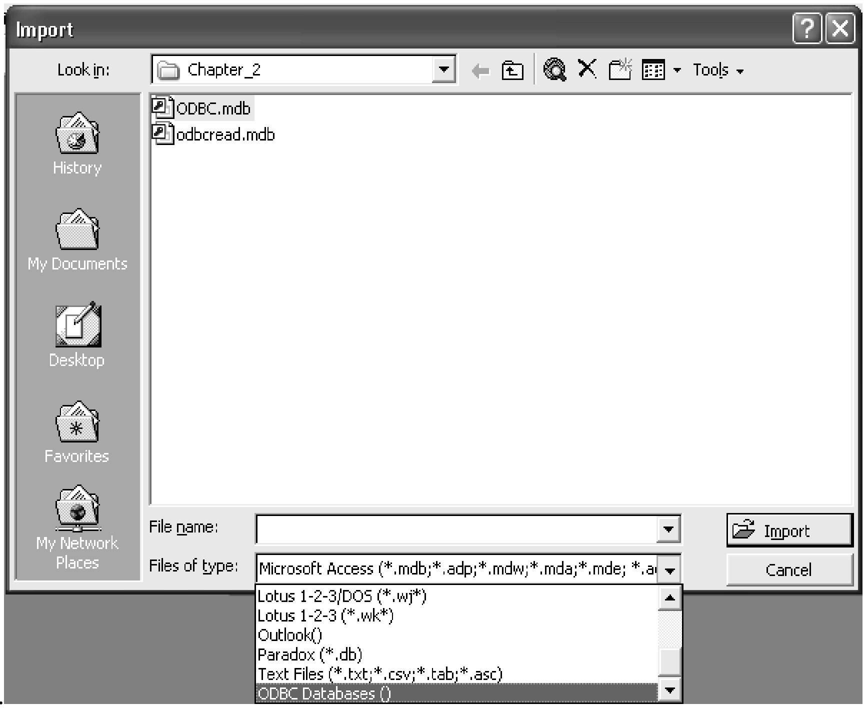
Microsoft Access can use the SAS ODBC driver to read from SAS data libraries with the Import menu option, or write to a specific SAS data library with the Export menu option.

The Import procedure is as follows:

1. Select **File ▶ Get External Data ▶ Import**.



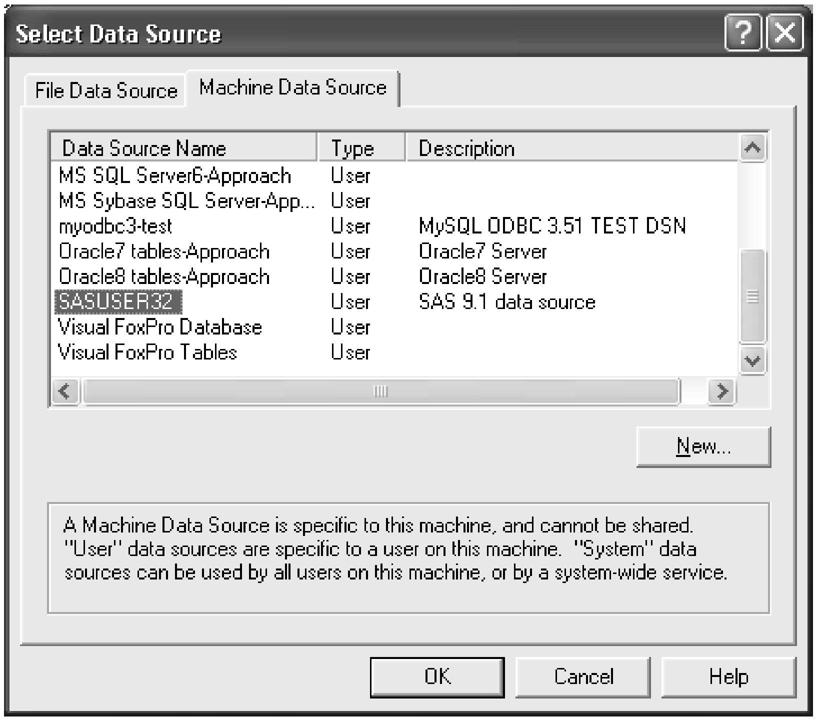
2. Select an import data source, e.g., **ODBC Databases**().



Files of type: Microsoft Access (*.mdb;*.adp;*.mdw;*.mda;*.mde; *.a

Lotus 1-2-3/DOS (*.wj*)
Lotus 1-2-3 (*.wk*)
Outlook()
Paradox (*.db)
Text Files (*.txt;*.csv;*.tab;*.asc)
ODBC Databases ()

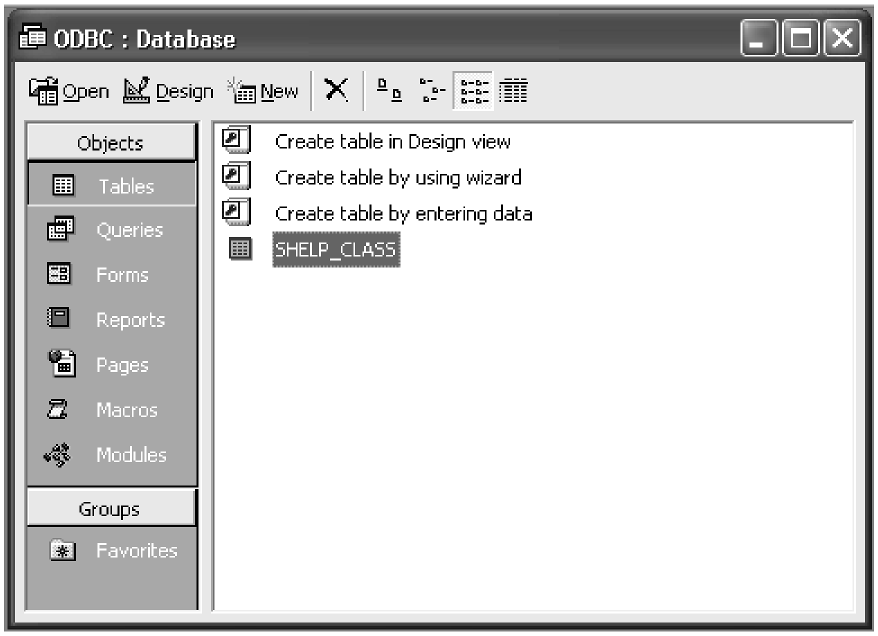
3. Click the **Machine Data Source** tab.
4. Select a data source, e.g., **SASUSER32**.
5. Click **OK**.



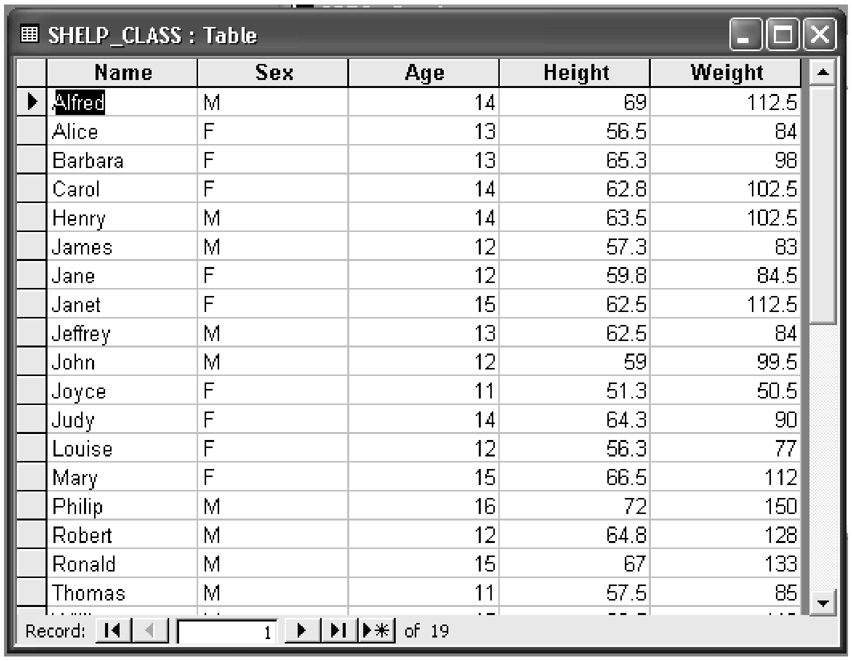
Data Source Name	Type	Description
MS SQL Server6-Approach	User	
MS Sybase SQL Server-App...	User	
myodbc3-test	User	MySQL ODBC 3.51 TEST DSN
Oracle7 tables-Approach	User	Oracle7 Server
Oracle8 tables-Approach	User	Oracle8 Server
SASUSER32	User	SAS 9.1 data source
Visual FoxPro Database	User	
Visual FoxPro Tables	User	

A Machine Data Source is specific to this machine, and cannot be shared. "User" data sources are specific to a user on this machine. "System" data sources can be used by all users on this machine, or by a system-wide service.

6. Select a data set, e.g., **SHELP_CLASS**.
7. Click **OK**.



8. Data is copied into a new Microsoft Access table, e.g., **SHELP_CLASS**.

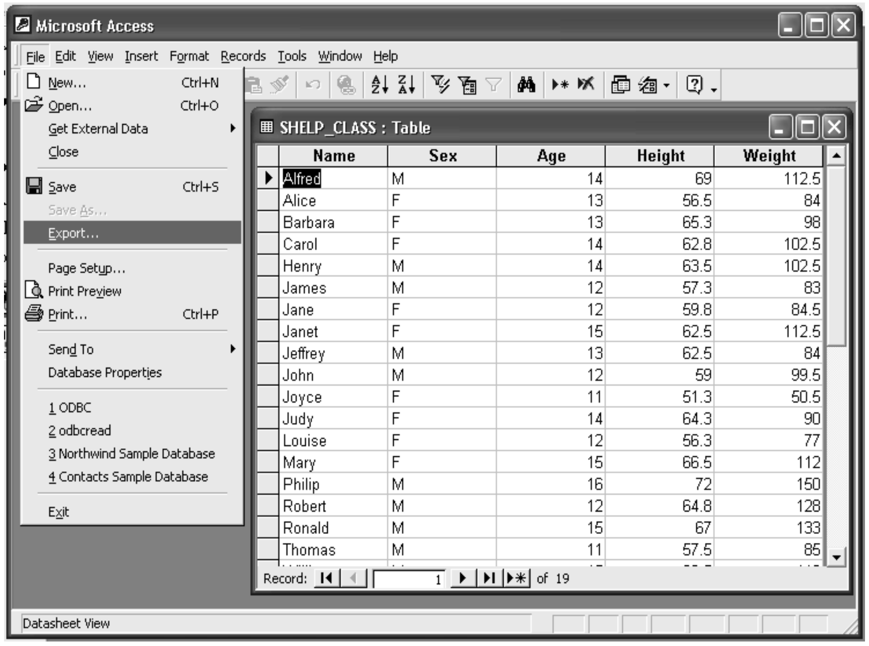


SHELP_CLASS : Table

Name	Sex	Age	Height	Weight
Alfred	M	14	69	112.5
Alice	F	13	56.5	84
Barbara	F	13	65.3	98
Carol	F	14	62.8	102.5
Henry	M	14	63.5	102.5
James	M	12	57.3	83
Jane	F	12	59.8	84.5
Janet	F	15	62.5	112.5
Jeffrey	M	13	62.5	84
John	M	12	59	99.5
Joyce	F	11	51.3	50.5
Judy	F	14	64.3	90
Louise	F	12	56.3	77
Mary	F	15	66.5	112
Philip	M	16	72	150
Robert	M	12	64.8	128
Ronald	M	15	67	133
Thomas	M	11	57.5	85

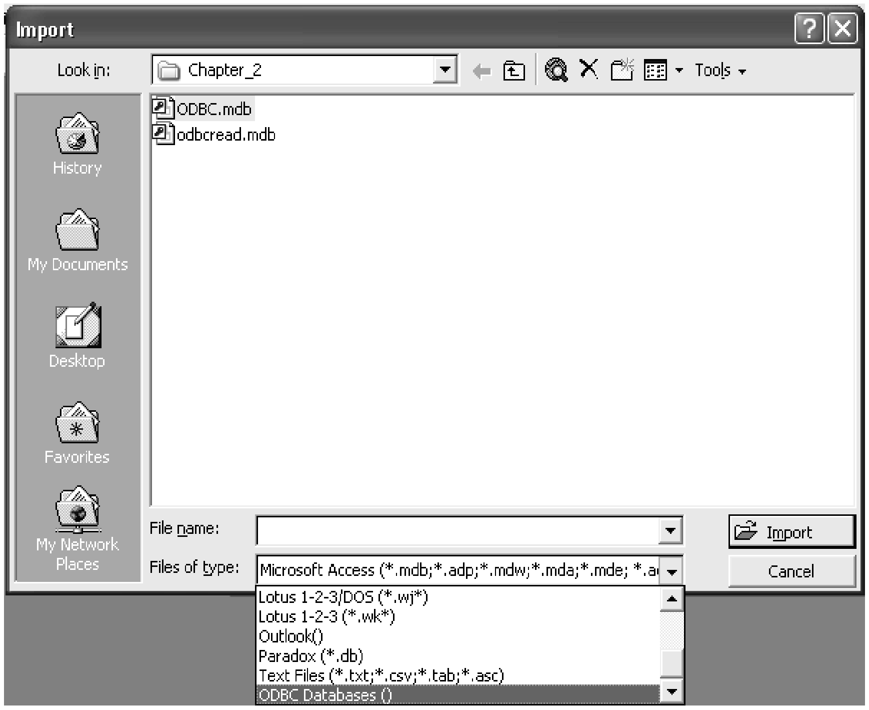
The Export procedure is as follows:

1. Select **File ▶ Export**.



Name	Sex	Age	Height	Weight
Alfred	M	14	69	112.5
Alice	F	13	56.5	84
Barbara	F	13	65.3	98
Carol	F	14	62.8	102.5
Henry	M	14	63.5	102.5
James	M	12	57.3	83
Jane	F	12	59.8	84.5
Janet	F	15	62.5	112.5
Jeffrey	M	13	62.5	84
John	M	12	59	99.5
Joyce	F	11	51.3	50.5
Judy	F	14	64.3	90
Louise	F	12	56.3	77
Mary	F	15	66.5	112
Philip	M	16	72	150
Robert	M	12	64.8	128
Ronald	M	15	67	133
Thomas	M	11	57.5	85

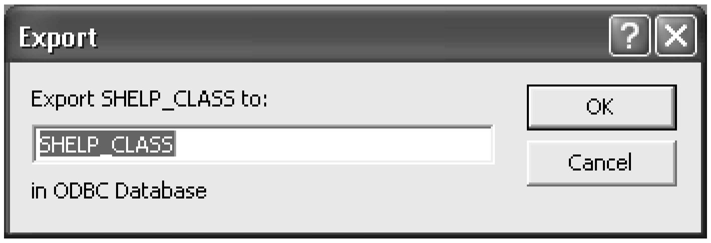
2. Select an export data source, e.g., **ODBC Databases()**.



Microsoft Access (*.mdb;*.adp;*.mdw;*.mda;*.mde; *.a
Lotus 1-2-3/DOS (*.wj*)
Lotus 1-2-3 (*.wk*)
Outlook()
Paradox (*.db)
Text Files (*.txt;*.csv;*.tab;*.asc)
ODBC Databases ()

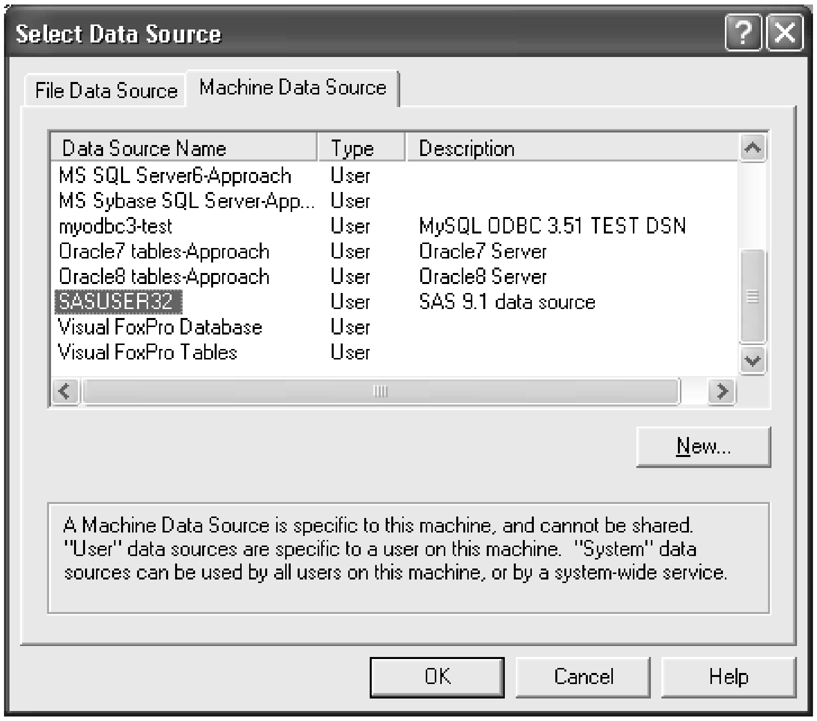
3. Select a Microsoft Access object, e.g., **SHELP_CLASS**.

4. Click **OK**.



Export SHELP_CLASS to:
SHELP_CLASS
in ODBC Database

5. Click the **Machine Data Source** tab.
6. Select a data source, e.g., **SASUSER32**.
7. Click **OK**.



Data Source Name	Type	Description
MS SQL Server6-Approach	User	
MS Sybase SQL Server-App...	User	
myodbc3-test	User	MySQL ODBC 3.51 TEST DSN
Oracle7 tables-Approach	User	Oracle7 Server
Oracle8 tables-Approach	User	Oracle8 Server
SASUSER32	User	SAS 9.1 data source
Visual FoxPro Database	User	
Visual FoxPro Tables	User	

A Machine Data Source is specific to this machine, and cannot be shared. "User" data sources are specific to a user on this machine. "System" data sources can be used by all users on this machine, or by a system-wide service.

8. Data is copied to a SAS data set, e.g., **USER.SHELP_CLASS**.

Problems

There are several problems when exporting Microsoft Access tables to SAS data libraries:

- Microsoft Access 2000 requires Jet 4.0 with Service Pack 6 applied to export data to a SAS data set. Without the service pack applied, the final step of the preceding export procedure gives the following error message:

  ODBC—call failed.
  [SAS][SAS ODBC Driver][SAS Serve (#-1) [SAS][SAS ODBC Driver][SAS Server]ERROR 76-3 (#-1)

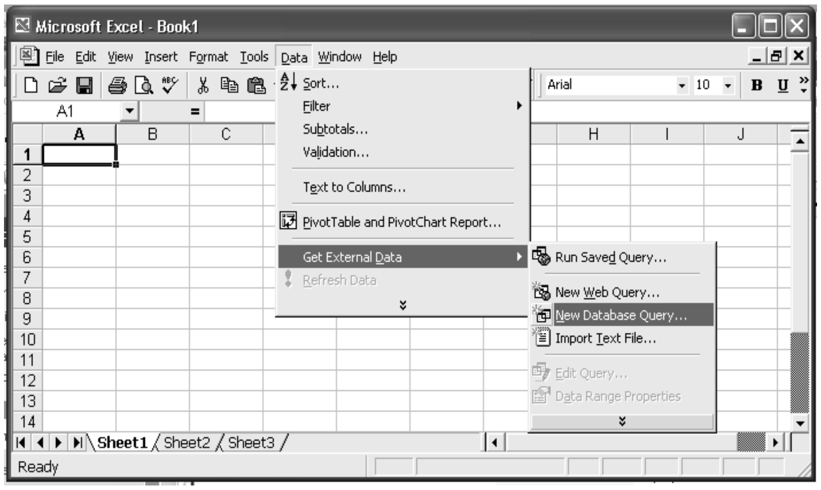
- SAS data sets can only be defined from Microsoft Access as single-level names, so a USER libname must be allocated in the ODBC setup to receive the exported data set.
- SAS data set names can be a maximum of only eight characters. The names cannot include punctuation or blanks. They can include only alphabetic characters (i.e., A–Z or a–z) or underscores as the first and subsequent characters. Numeric characters (i.e., 0–9) can only be used as the second and subsequent characters.
- SAS column names have the same rules as SAS data set names. Microsoft Access has a menu option (i.e., **File ▶ Imp/Exp Setup**) to allow changes to be made to the field information before exporting.

1.3.3 Microsoft Excel 2000

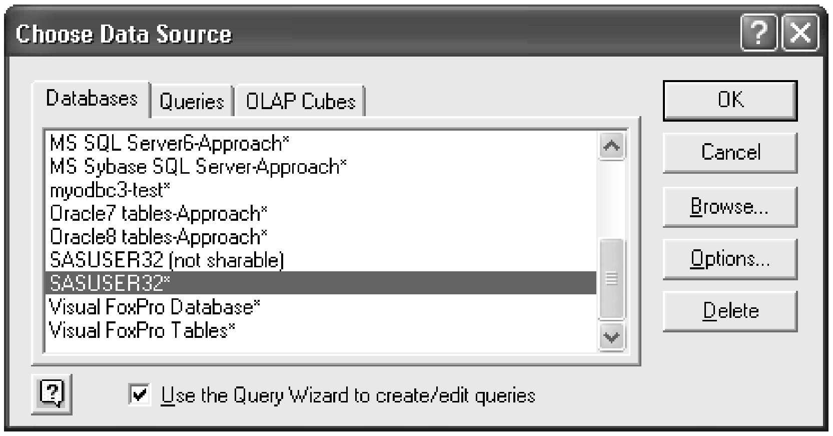
Microsoft Excel can only use the SAS ODBC driver to read from SAS data libraries with the **Get External Data** menu option.

The procedure is as follows:

1. Select **Data** ▶ **Get External Data** ▶ **New Database Query**.



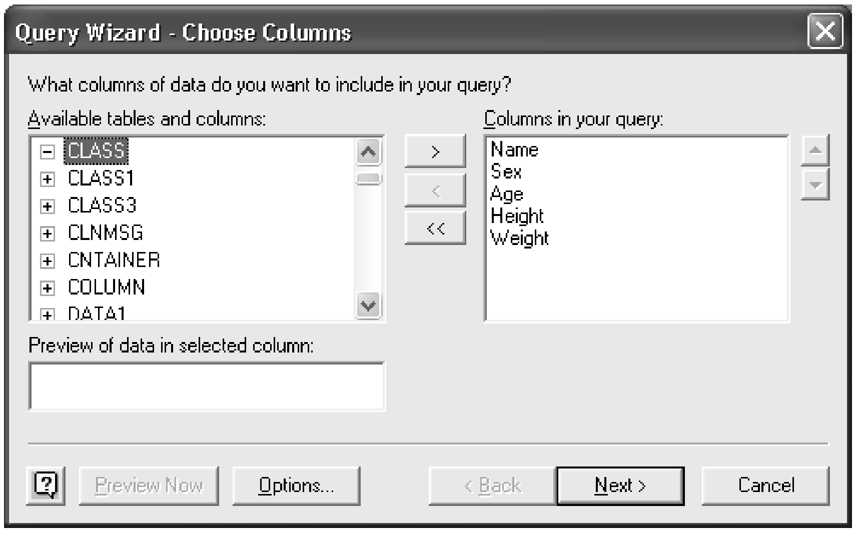
2. Select a data source, e.g., **SASUSER32***.
3. Select the **Use the Query Wizard to create/edit queries** check box.
4. Click **OK**.



MS SQL Server6-Approach*
MS Sybase SQL Server-Approach*
myodbc3-test*
Oracle7 tables-Approach*
Oracle8 tables-Approach*
SASUSER32 (not sharable)
SASUSER32*
Visual FoxPro Database*
Visual FoxPro Tables*

☑ Use the Query Wizard to create/edit queries

5. Select a data set, e.g., **CLASS**.
6. Click the right arrow (>) to select all the columns in your query. To select a subset of the columns, click the plus sign (+) next to the data set name first.
7. Click **Next**.



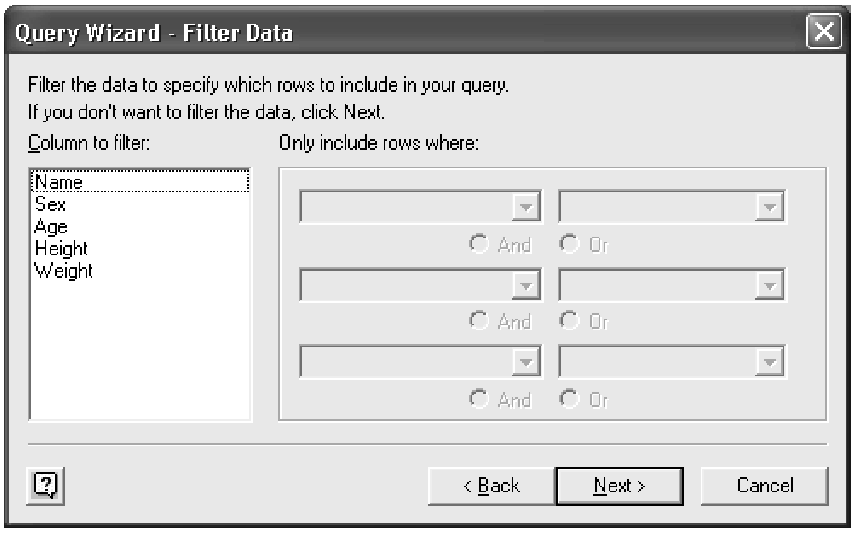
What columns of data do you want to include in your query?

Available tables and columns:
CLASS
CLASS1
CLASS3
CLNMSG
CNTAINER
COLUMN
DATA1

Columns in your query:
Name
Sex
Age
Height
Weight

Preview of data in selected column:

8. Select the columns to filter.
9. Click **Next**.

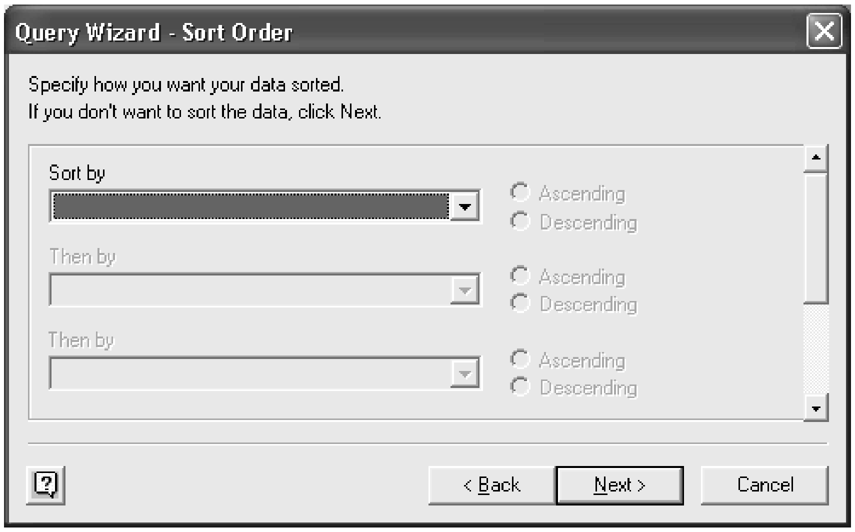

Query Wizard - Filter Data

Filter the data to specify which rows to include in your query.
If you don't want to filter the data, click Next.

Column to filter: Name, Sex, Age, Height, Weight

Only include rows where:

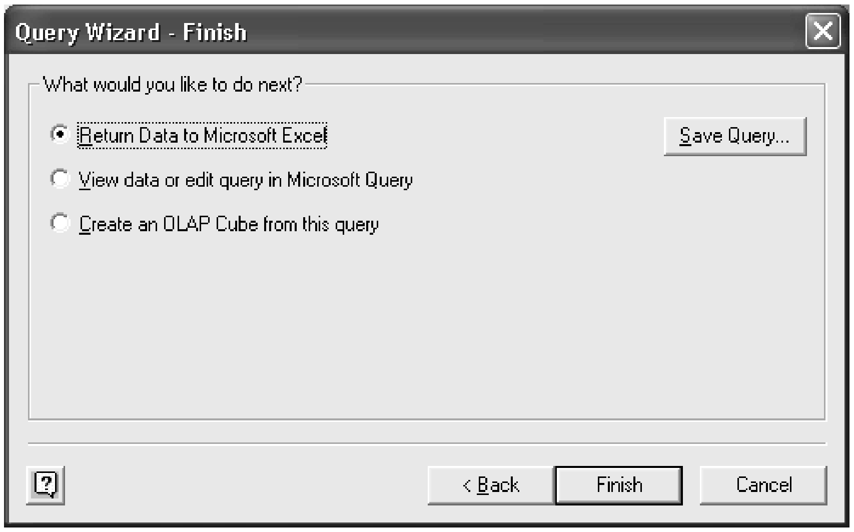
10. Select the sort order.
11. Click **Next**.


Query Wizard - Sort Order

Specify how you want your data sorted.
If you don't want to sort the data, click Next.

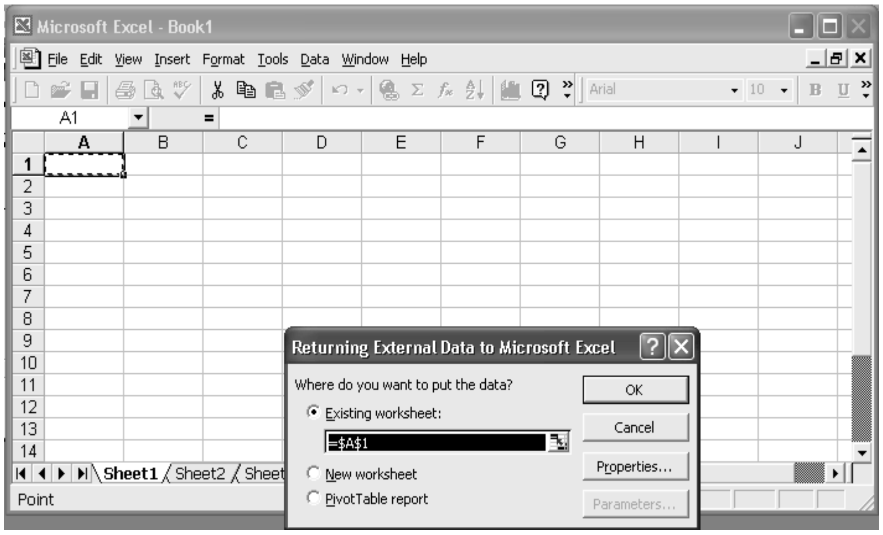
12. Select **Return Data to Microsoft Excel**.

13. Click **Finish**.

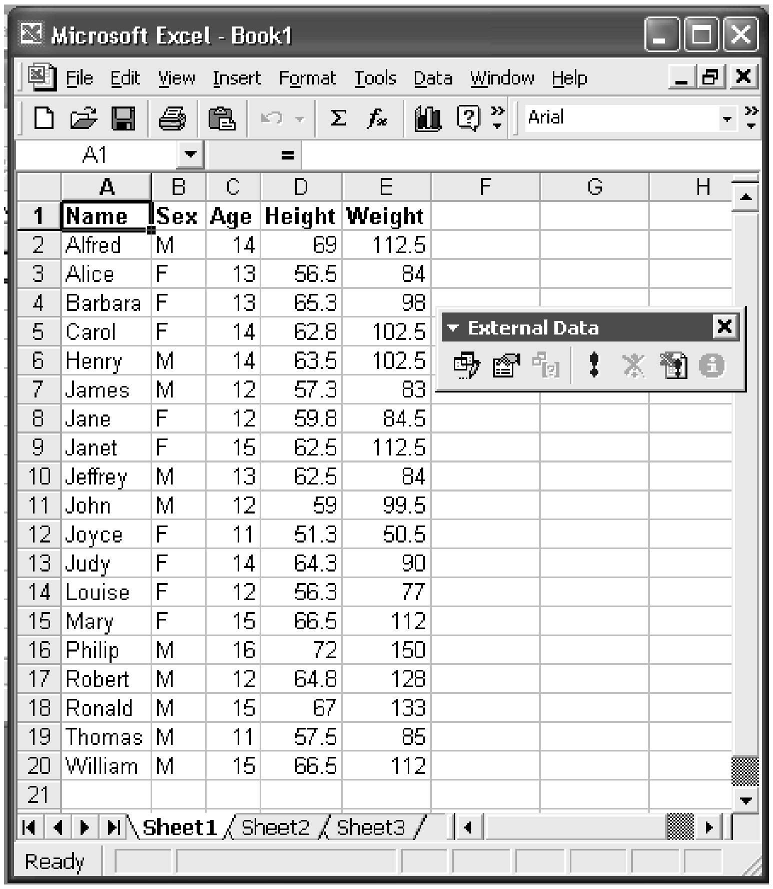

Query Wizard - Finish
What would you like to do next?
Return Data to Microsoft Excel
View data or edit query in Microsoft Query
Create an OLAP Cube from this query
Save Query...
< Back
Finish
Cancel

14. Select where you want to put the data.

15. Click **OK**.



16. Data is inserted at the destination location.



Name	Sex	Age	Height	Weight
Alfred	M	14	69	112.5
Alice	F	13	56.5	84
Barbara	F	13	65.3	98
Carol	F	14	62.8	102.5
Henry	M	14	63.5	102.5
James	M	12	57.3	83
Jane	F	12	59.8	84.5
Janet	F	15	62.5	112.5
Jeffrey	M	13	62.5	84
John	M	12	59	99.5
Joyce	F	11	51.3	50.5
Judy	F	14	64.3	90
Louise	F	12	56.3	77
Mary	F	15	66.5	112
Philip	M	16	72	150
Robert	M	12	64.8	128
Ronald	M	15	67	133
Thomas	M	11	57.5	85
William	M	15	66.5	112

1.3.4 Visual Basic 6.0

Visual Basic can use the SAS ODBC driver to read from SAS data libraries, with shared read-only access. The following example code will copy all of the data from a SAS data set into a dBase table, which can be read, or imported, into all the database and spreadsheet applications discussed in this chapter.

In order to be able to copy the SAS data from the ODBC connection to the dBase table, the target table has to be constructed to include exactly the same fields:

```
Public Function CopyStructDAO(daoRset1 As DAO.Recordset, _
                              daoDB2 As DAO.Database, _
                              DBTable As String) As Integer
  Dim i As Integer
  Dim daoTDef2 As DAO.TableDef
  Dim tmpTDef As DAO.TableDef
  Dim daoFld2 As DAO.Field
  Dim daoFld1 As DAO.Field
  Dim errorFlag As Boolean
  ' Search to see if table exists
  For i = 0 To daoDB2.TableDefs.Count - 1
    Set tmpTDef = daoDB2.TableDefs(i)
    If UCase(tmpTabDef.Name) = UCase(DBTable) Then
      daoDB2.TableDefs.Delete tmpTDef.Name
      Exit For
    End If
  Next
  Set daoTDef2 = daoDB2.CreateTableDef(DBTable)
  ' Strip off owner if present
  daoTDef2.Name = StripOwner(DBTable)
  ' Create fields
  For i = 0 To daoRset1.Fields.Count - 1
    Set daoFld1 = daoRset1.Fields(i)
    Set daoFld2 = Nothing
    errorFlag = True
    Select Case daoFld1.Type
      Case dbDouble
        Set daoFld2 = daoTDef2.CreateField(CVar(daoFld1.Name),_
                                  Cvar(daoFld1.Type))
        errorFlag = False
      Case dbText
        Set daoFld2 = daoTDef2.CreateField(CVar(daoFld1.Name),_
                                  Cvar(daoFld1.Type), _
                                  Cvar(daoFld1.FieldSize))
        errorFlag = False
    End Select
```

```
      If Not errorFlag Then
        daoTDef2.Fields.Append daoFld2
      End If
    Next
    ' Append new table
    daoDB2.TableDefs.Append daoTDef2
    CopyStructDAO = True
  End Function
```

Once the table structure has been copied and the two tables opened as recordsets, the data itself can then be copied record by record, 1,000 records at a time, using transactions:

```
  Public Function CopyDataDAO(defWSpace As Workspace, _
                              daoRset1 As DAO.Recordset, _
                              daoRset2 As DAO.Recordset) As Integer
    Dim i As Integer
    Dim j As Integer
    Dim numRecords As Integer
    Dim daoFld1 As DAO.Field
    Dim daoFld2 As DAO.Field
    ' Start workspace transactions
    defWSpace.BeginTrans
    While daoRset1.EOF = False
      daoRset2.AddNew
      ' Loop copies data from each field to new table
      For i = 0 To (daoRset1.Fields.Count - 1)
        Set daoFld1 = daoRset1.Fields(i)
        For j = 0 To (daoRset2.Fields.Count - 1)
          Set daoFld2 = daoRset2.Fields(j)
          If UCase(daoFld1.Name) = UCase(daoFld2.Name) Then
            daoRset2(daoFld2.Name).Value = daoFld1.Value
            Exit For
          End If
        Next
      Next
      daoRset2.Update
      daoRset1.MoveNext
      numRecords = numRecords + 1
      ' Commit transactions every 1000 records
      If numRecords = 1000 Then
        defWSpace.CommitTrans
        defWSpace.BeginTrans
        numRecords = 0
      End If
    Wend
    ' Commit changes now we have finished
    defWSpace.CommitTrans
    CopyDataDAO = True
  End Function
```

The following function is used to remove “owner” information from the ODBC table names, e.g., SHELP.CLASS becomes CLASS:

```
Public Function StripOwner(TableName As String) As String
  If InStr(TableName, ".") > 0 Then
    TableName = Mid(TableName, _
                    InStr(TableName, ".") + 1, _
                    Len(TableName))
  End If
  StripOwner = TableName
End Function
```

The functions already described can be brought together to copy the structure and data from the source table to the target table as follows:

```
Public Sub SaveDAOToDAO(defWSpace As Workspace, _
                        daoRset1 As DAO.Recordset, _
                        daoDB2 As DAO.Database, _
                        DBTable As String)
  Dim daoRset2 As DAO.Recordset
  Dim i As Integer
  If CopyStructDAO(daoRset1, daoDB2, DBTable) Then
    Set daoRset2 = daoDB2.OpenRecordset(DBTable)
    If CopyDataDAO(defWSpace, daoRset1, daoRset2) Then
      daoRset2.Close

      Debug.Print "CopyData to perm completed..."
    End If
  End If
End Sub
```

All that is left to do is to specify the ODBC server name and the source and target tables:

```
Public Sub ODBCread()
  ' Create DAO objects
  Dim odbcDB As DAO.Database
  Dim odbcRset As DAO.Recordset
  Dim defWSpace As Workspace
  Dim daoDB As DAO.Database
  Dim odbcServer As String
  Dim DataSet As String
  Dim DBFilePath As String
  Dim DBTable As String
  Dim defFileSystem As Object
  Dim DBFileObject As Object
  ' Set file and database folder values
  DataSet = "shelp.class"
  odbcServer = "sasuser32"
  DBTable = "class1"
  DBFilePath = "c:\temp"
  ' Verify path
  Set defFileSystem = CreateObject("Scripting.FileSystemObject")
  Set DBFileObject = defFileSystem.GetFolder(DBFilePath)
  ' Get default workspace.
  Set defWSpace = DBEngine.Workspaces(0)
  ' Make sure there isn't already a file with the same name
  ' in the folder.
  If Dir(Trim$(DBFileObject.Path) & _
         "\" & _
         Trim$(DBTable) & ".dbf") <> "" Then
   Kill Trim$(DBFileObject.Path) & "\" & Trim$(DBTable) & ".dbf"
  End If
  ' Open database
  Set daoDB = defWSpace.OpenDatabase(Trim$(DBFileObject.Path), _
                                     False, _
                                     False, _
                                     "dBase IV;")
  ' Set initialization properties
  Set odbcDB = defWSpace.OpenDatabase(odbcServer, _
                                      False, _
                                      True, _
                                      "ODBC;")
  ' Open recordset
  Set odbcRset = odbcDB.OpenRecordset(DataSet)
  Call SaveDAOToDAO(defWSpace, _
                    odbcRset, _
                    daoDB, _
                    DBTable)
  ' Close connection
  odbcRset.Close
  odbcDB.Close
  daoDB.Close
  Set defWSpace = Nothing
End Sub
```

Visual Basic can also use the SAS ODBC driver to write to SAS data libraries, with exclusive update access, using the following DatabaseName and Connect strings, e.g.:

```
Set DB = OpenDatabase("saswrite", True, False, "ODBC;")
```

The database, “DB”, can be manipulated in the same way as any Microsoft Access database using standard Visual Basic Data Access functions and methods. The SAS data sets within the ODBC data source can be accessed using the standard SAS 'libref.member' naming conventions, via Jet SQL supplied as part of Visual Basic and Microsoft Access.

1.3.5 Lotus Approach Version 9

LotusScript Version 9

In the same way that Visual Basic can use the SAS ODBC driver to read from SAS data libraries, LotusScript can be used to create a new Approach document to view a SAS data set:

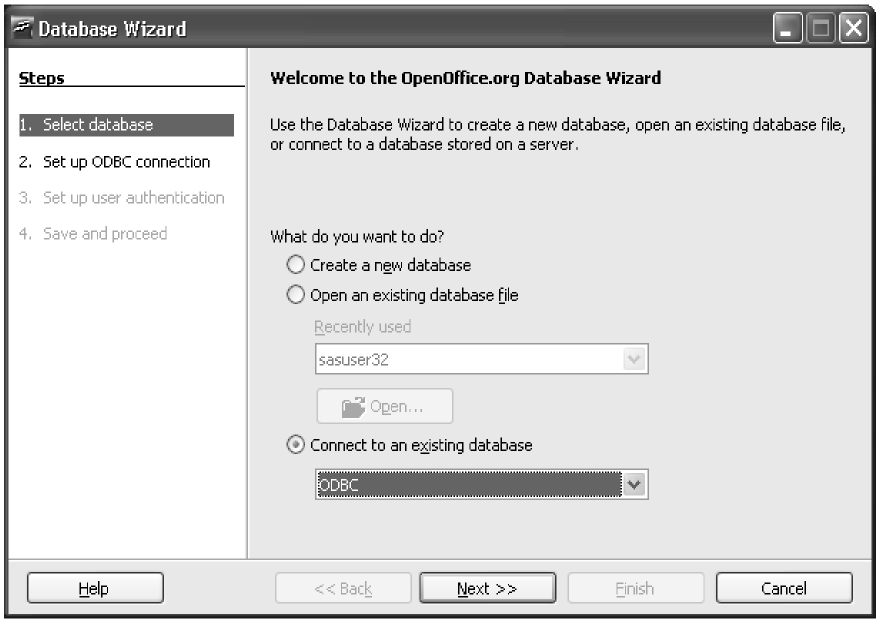
```
Sub ODBCread
  ' Create new connection
  Dim Con As New Connection()
  ' Create new query
  Dim Qry As New Query()
  ' Create new resultset
  Dim RS As New ResultSet()
  Dim MyDoc As Document
  Dim ServName As String
  Dim TName As String
  ' Specify ODBC server name
  ServName = "sasuser32"
  ' Specify source data set name
  TName = "shelp.class"
  ' Open ODBC connection to server
  ' (which must be prefixed with "!")
  If Con.ConnectTo("ODBC Data Sources", , , "!" & ServName) Then
    ' Associate query with connection
    Set Qry.Connection = Con
    ' Set table to open
    Qry.TableName = Tname
    ' Associate resultset with query
    Set RS.Query = Qry
    ' Populate resultset
    If RS.Execute Then
      ' Create Approach document for resultset
      Set MyDoc = New Document(RS)
    End If
    ' Close connection
    Con.Disconnect
  End If
End Sub
```

1.3.6 OpenOffice.org 2.1

While, in theory, OpenOffice.org Base can use ODBC drivers compatible with ODBC 3 to access database tables, the SAS ODBC driver was compatible only with ODBC 2 prior to SAS 9.1. Following the changes made to the SAS ODBC driver in SAS 9.1, it is now possible to import SAS data into OpenOffice.org spreadsheets without any errors.

Data from a SAS data set can now be imported into OpenOffice.org documents, but it must first be registered in an OpenOffice.org database as follows:

1. Open OpenOffice.org Base.
2. Select **Connect to an existing database**.
3. Select **ODBC** from the drop-down list.
4. Click **Next**.



Database Wizard

Steps

1. Select database
2. Set up ODBC connection
3. Set up user authentication
4. Save and proceed

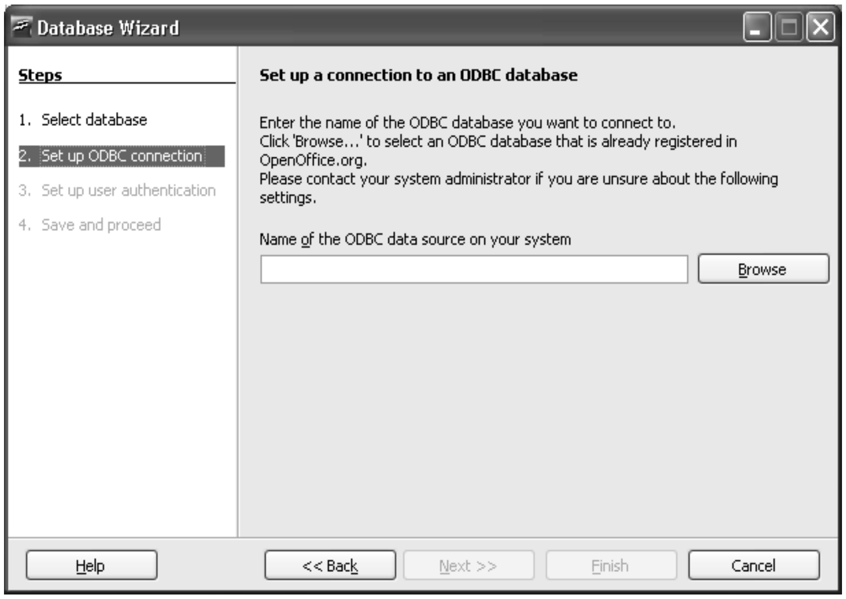
Welcome to the OpenOffice.org Database Wizard

Use the Database Wizard to create a new database, open an existing database file, or connect to a database stored on a server.

What do you want to do?

- Create a new database
- Open an existing database file
  - Recently used: sasuser32
  - Open...
- [✓ Connect to an existing database]
  - ODBC

5. Click **Browse**.



Database Wizard

Steps

1. Select database
2. Set up ODBC connection
3. Set up user authentication
4. Save and proceed

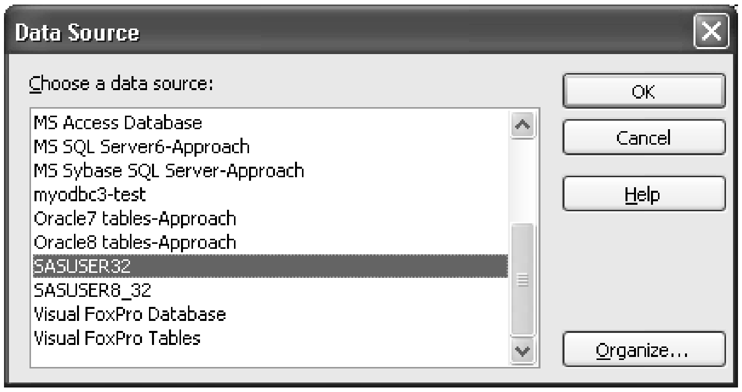
Set up a connection to an ODBC database

Enter the name of the ODBC database you want to connect to.
Click 'Browse…' to select an ODBC database that is already registered in OpenOffice.org.
Please contact your system administrator if you are unsure about the following settings.

Name of the ODBC data source on your system

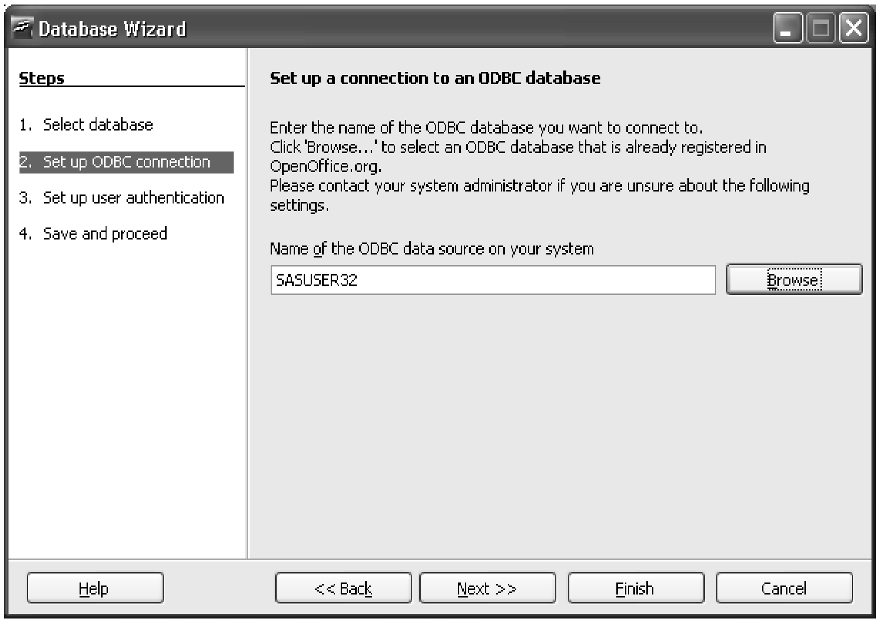
6. Select a data source, e.g., **SASUSER32**.

7. Click **OK**.



MS Access Database
MS SQL Server6-Approach
MS Sybase SQL Server-Approach
myodbc3-test
Oracle7 tables-Approach
Oracle8 tables-Approach
SASUSER32
SASUSER8_32
Visual FoxPro Database
Visual FoxPro Tables

8. Click **Next**.

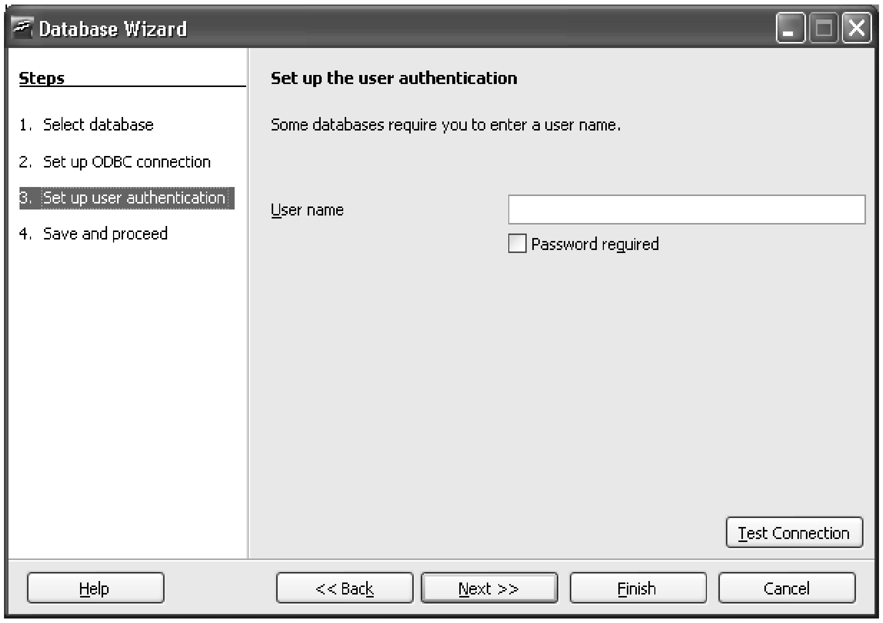


Set up a connection to an ODBC database

Enter the name of the ODBC database you want to connect to.
Click 'Browse...' to select an ODBC database that is already registered in OpenOffice.org.
Please contact your system administrator if you are unsure about the following settings.

Name of the ODBC data source on your system: SASUSER32

9. If a user name and password are required, fill in the details.
10. The connection to the SAS ODBC data source can be tested by clicking **Test Connection**. Otherwise, click **Next**.



Set up the user authentication

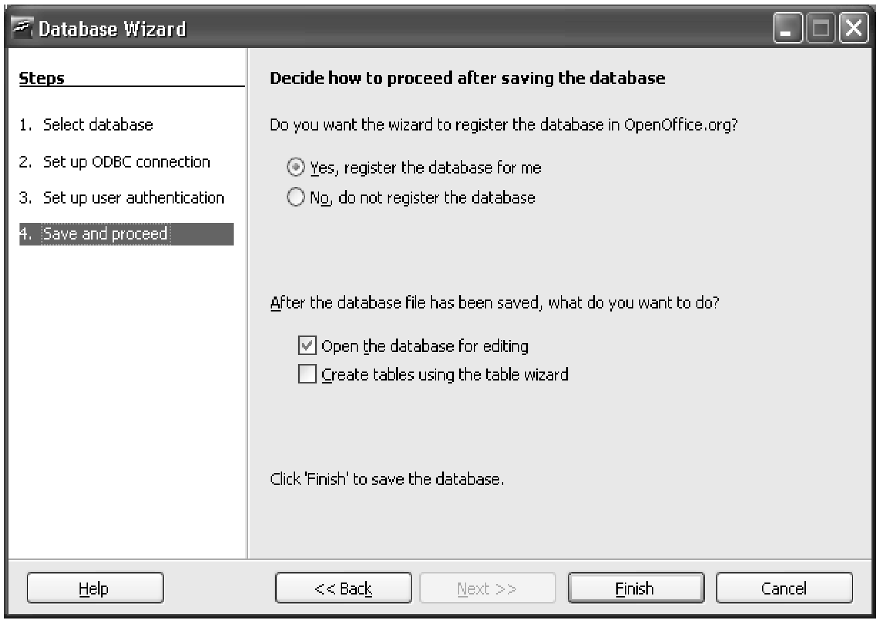
Some databases require you to enter a user name.

User name

☐ Password required

11. Select **Yes, register the database for me**.

12. Click **Finish**.



Decide how to proceed after saving the database

Do you want the wizard to register the database in OpenOffice.org?

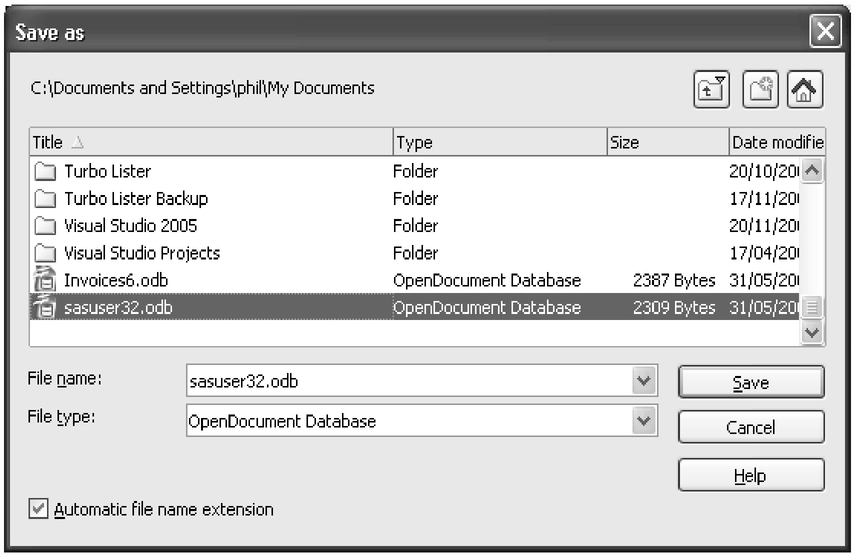
● Yes, register the database for me
○ No, do not register the database

After the database file has been saved, what do you want to do?

☑ Open the database for editing
☐ Create tables using the table wizard

Click 'Finish' to save the database.

13. Enter the location of the new OpenOffice.org database.
14. Click **Save**.



Save as

C:\Documents and Settings\phil\My Documents

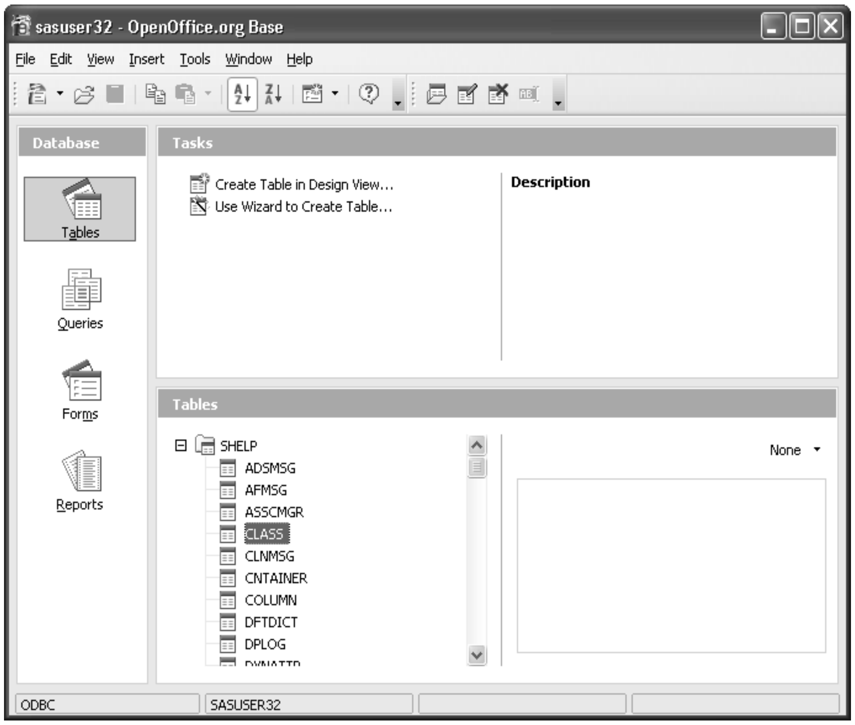
Title	Type	Size	Date modifie
Turbo Lister	Folder		20/10/20
Turbo Lister Backup	Folder		17/11/20
Visual Studio 2005	Folder		20/11/20
Visual Studio Projects	Folder		17/04/20
Invoices6.odb	OpenDocument Database	2387 Bytes	31/05/20
sasuser32.odb	OpenDocument Database	2309 Bytes	31/05/20

File name: sasuser32.odb

File type: OpenDocument Database

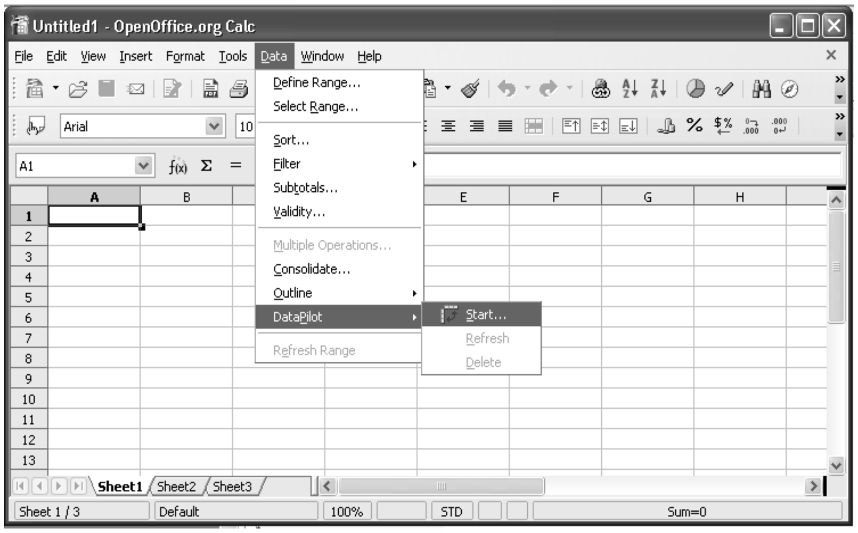
☑ Automatic file name extension

15. The database can now be accessed from an OpenOffice.org document, or reports can be created directly in OpenOffice.org Base.

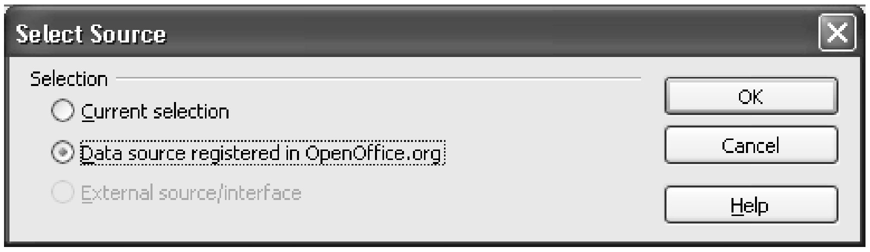


The data from the registered OpenOffice.org database created from the SAS ODBC data source can be added to an OpenOffice.org spreadsheet as follows:

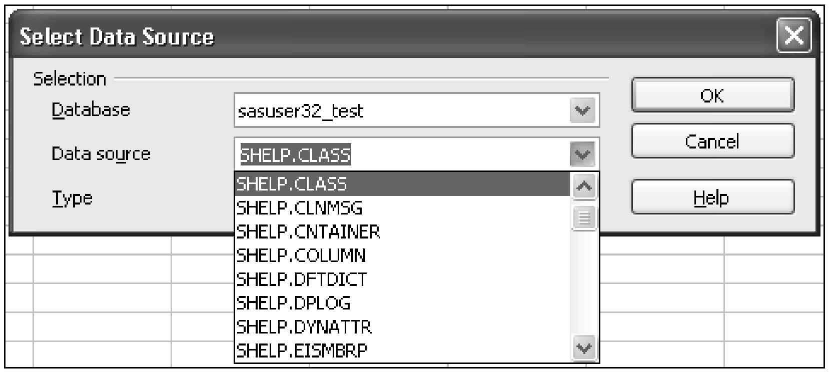
1. Select **Data ▶ DataPilot ▶ Start**.



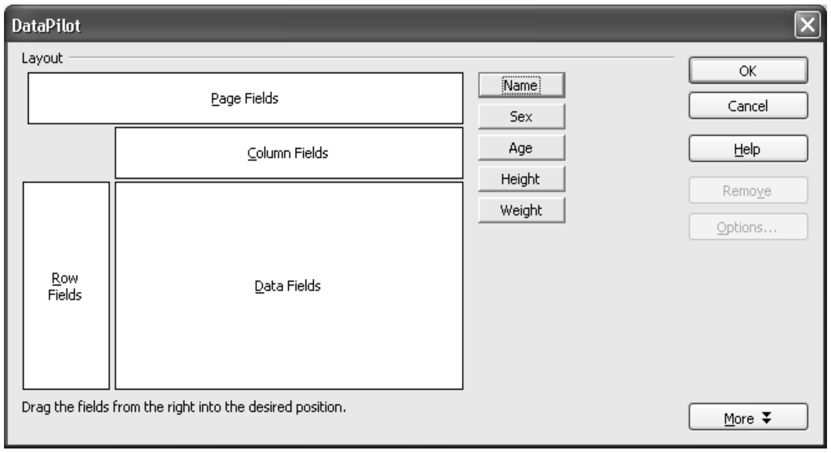
2. Select **Data source registered in OpenOffice.org**.
3. Click **OK**.


Select Source
Selection
Current selection
Data source registered in OpenOffice.org
External source/interface
OK
Cancel
Help

4. Select a database from the drop-down list.
5. Select a data source from the drop-down list.
6. Select a type, e.g., **Sheet**.
7. Click **OK**.

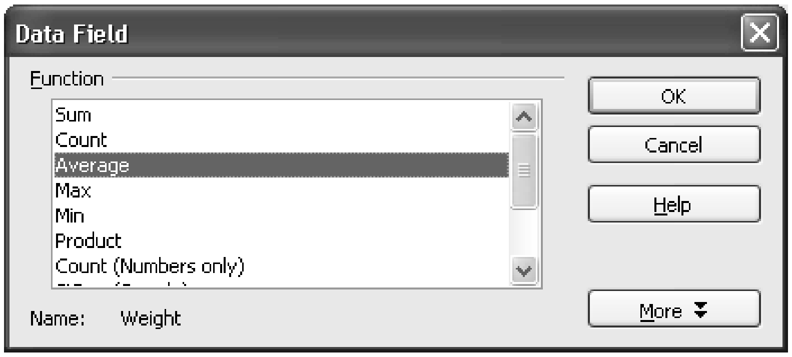


8. Drag fields into the report template.
9. Select a data field, and then click **Options** to change the summary statistic.



10. Select a statistic from the list for the data field.

11. Click **OK**.



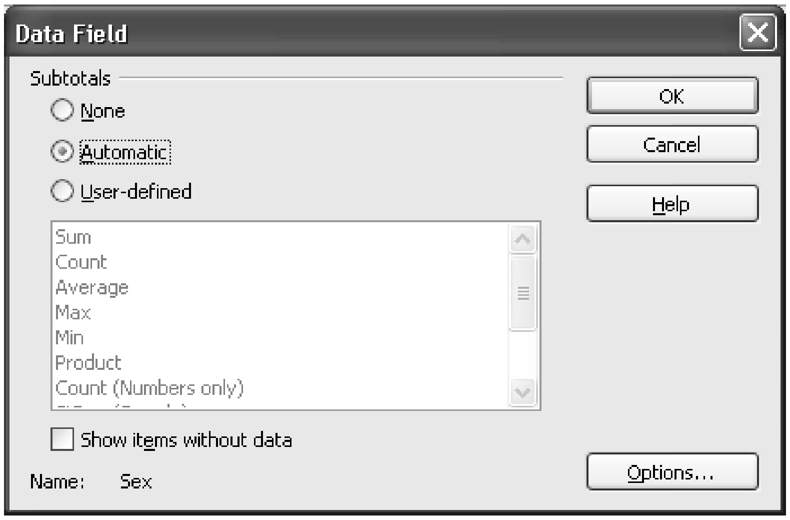
Sum
Count
Average
Max
Min
Product
Count (Numbers only)

Name: Weight

12. Select a column or row field, and then click **Options** to change the subtotals.

13. Select the **Subtotal** option.

14. Click **OK**.



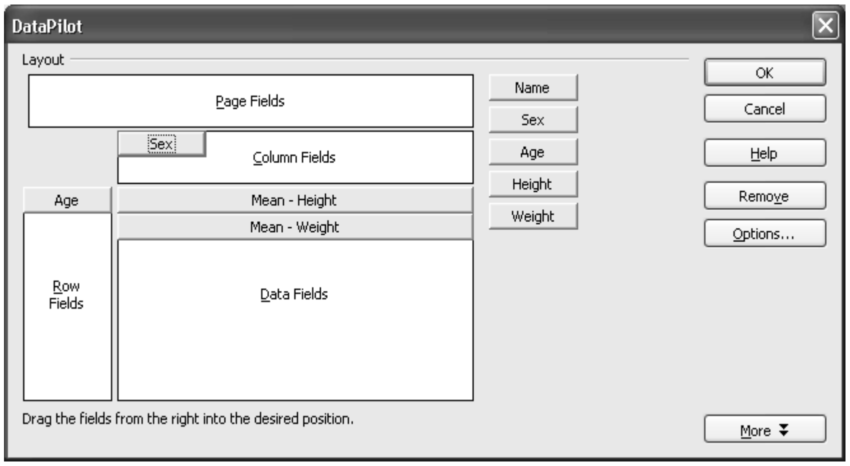
None [✓ Automatic] User-defined

Sum
Count
Average
Max
Min
Product
Count (Numbers only)

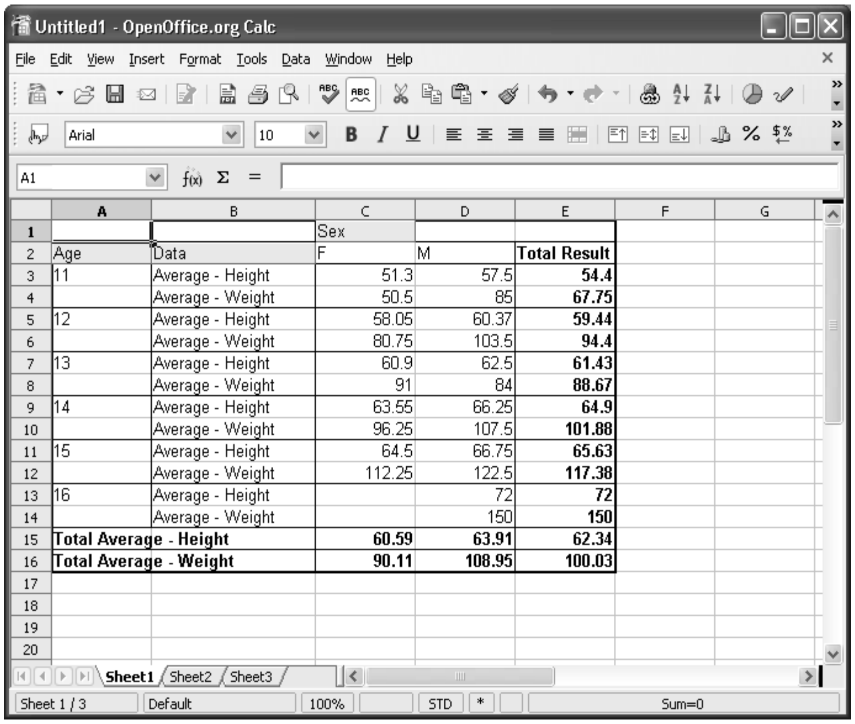
Show items without data

Name: Sex

15. Click **OK**.



16. The report is generated at the currently selected cell.


Untitled1 - OpenOffice.org Calc
File Edit View Insert Format Tools Data Window Help
Arial
10
A1
Sheet1 Sheet2 Sheet3
Sheet 1 / 3
Default
100%
STD
Sum=0

	A	B	C	D	E
1			Sex		
2	Age	Data	F	M	**Total Result**
3	11	Average - Height	51.3	57.5	**54.4**
4		Average - Weight	50.5	85	**67.75**
5	12	Average - Height	58.05	60.37	**59.44**
6		Average - Weight	80.75	103.5	**94.4**
7	13	Average - Height	60.9	62.5	**61.43**
8		Average - Weight	91	84	**88.67**
9	14	Average - Height	63.55	66.25	**64.9**
10		Average - Weight	96.25	107.5	**101.88**
11	15	Average - Height	64.5	66.75	**65.63**
12		Average - Weight	112.25	122.5	**117.38**
13	16	Average - Height		72	**72**
14		Average - Weight		150	**150**
15	**Total Average - Height**		**60.59**	**63.91**	**62.34**
16	**Total Average - Weight**		**90.11**	**108.95**	**100.03**

1.3.6.1 Visual Basic 6.0

It is also very easy to use Visual Basic to access the SAS ODBC driver, and then write directly into an OpenOffice.org Calc spreadsheet or into a table in an OpenOffice.org Writer document.

The following code is common to both processes, and should be included before the application-specific code:

```
Dim oServiceManager As Object
Dim oDesktop As Object
' Get the Service Manager object.
Set oServiceManager = _
    CreateObject("com.sun.star.ServiceManager")
' Get the Desktop object.
Set oDesktop = _
   oServiceManager.createInstance("com.sun.star.frame.Desktop")
' Create DAO objects
Dim odbcDB As DAO.Database
Dim odbcRecordset As DAO.Recordset
Dim defWSpace As Workspace
Dim odbcServer As String
Dim DataSet As String
Dim odbcField As Object
' Set file and database folder values
Dim row As Long
Dim column As Long
' Use this empty array when no arguments are needed.
Dim aNoArgs()
DataSet = "shelp.class"
odbcServer = "sasuser32"
' Get default workspace.
Set defWSpace = DBEngine.Workspaces(0)
' Set initialization properties
Set odbcDB = defWSpace.OpenDatabase(odbcServer, _
                                    False, _
                                    True, _
                                    "ODBC;")
' Open recordset
Set odbcRecordset = odbcDB.OpenRecordset(DataSet)
```

The following code is common to both processes, and should be included after the application-specific code:

```
' Close connection
odbcRecordset.Close
odbcDB.Close
Set defWSpace = Nothing
```

OpenOffice.org Calc

```
Dim oCalcDoc As Object
Dim oSheet As Object
' Create a new empty spreadsheet.
Set oCalcDoc = _
  oDesktop.loadComponentFromURL("private:factory/scalc", _
                              "_blank", 0, aNoArgs())
' Get the first spreadsheet from the sheets in the document.
Set oSheet = oCalcDoc.getSheets().getByIndex(0)
' Write table to Calc sheet
'this loop copies each field name into the 1st row of the sheet
row = 0
For column = 0 To (odbcRecordset.Fields.Count - 1)
  Set odbcField = odbcRecordset.Fields(column)
  oSheet.getCellByPosition(column, 0).setFormula _
        CStr(odbcField.Name)
Next
While odbcRecordset.EOF = False
  row = row + 1
  'this loop copies the data from each field to the new table
  For column = 0 To (odbcRecordset.Fields.Count - 1)
    Set odbcField = odbcRecordset.Fields(column)
    If odbcField.Type = 4 or odbcField.Type = 7 Then
      oSheet.getCellByPosition(column, row).setValue _
            odbcField.Value
    Else
      oSheet.getCellByPosition(column, row).setFormula _
            CStr(odbcField.Value)
    End If
  Next
  odbcRecordset.MoveNext
Wend
```

OpenOffice.org Writer

```
Dim oText As Object
Dim oText2 As Object
Dim oTable As Object
Dim oCursor As Object
' Create a new blank text document.
Set oText = _
    oDesktop.loadComponentFromURL("private:factory/swriter", _
                              "_blank", 0, aNoArgs())
' insert TextTable
Set oTable =
oText.createInstance("com.sun.star.text.TextTable")
' Create position cursor
Set oText2 = oText.GetText()
Set oCursor = oText2.createTextCursor()
' Write table to Text doc
' initialize the table with the correct number of columns + rows
oTable.Initialize 1, CLng(odbcRecordset.Fields.Count)
oTable.RepeatHeadline = True
' insert table now
oCursor.gotoStart False
oText2.insertTextContent oCursor, oTable, False
'this loop copies each field name into the 1st row of the table
row = 0
For column = 0 To (odbcRecordset.Fields.Count - 1)
  Set odbcField = odbcRecordset.Fields(column)
  oTable.getCellByPosition(column, 0).String = _
       CStr(odbcField.Name)
Next
odbcRecordset.MoveFirst
While odbcRecordset.EOF = False
  row = row + 1
  oTable.GetRows().insertByIndex row, 1
  'this loop copies the data from each field to the new table
  For column = 0 To (odbcRecordset.Fields.Count - 1)
    Set odbcField = odbcRecordset.Fields(column)
    oTable.getCellByPosition(column, row).String = _
         CStr(odbcField.Value)
  Next
  odbcRecordset.MoveNext
Wend
```

1.3.6.2 OpenOffice.org Base 2.1

The Basic programming language within the different components of OpenOffice.org is similar to Visual Basic for Applications. The following Base code, which should be run as a macro within the corresponding component, is functionally the same as the preceding code to access the SAS ODBC driver, and then to write directly into the OpenOffice.org Calc spreadsheet in a sheet called “Class report,” or into a new table at the end of the OpenOffice.org Writer document.

The following code is common to both processes, and should be included before the application-specific code:

```
Dim DatabaseContext As Object
Dim DataSource As Object
Dim Connection As Object
Dim InteractionHandler as Object
Dim Statement As Object
Dim ResultSet As Object
Dim row As Long
Dim column As Long
DatabaseContext =
     createUnoService("com.sun.star.sdb.DatabaseContext")
DataSource = DatabaseContext.getByName("sasuser32")
If Not DataSource.IsPasswordRequired Then
  Connection = DataSource.GetConnection("","")
Else
  InteractionHandler =
       createUnoService("com.sun.star.sdb.InteractionHandler")
  Connection =
       DataSource.ConnectWithCompletion(InteractionHandler)
End If
Statement = Connection.createStatement()
ResultSet = Statement.executeQuery("SELECT * FROM shelp.class")
```

The following code is common to both processes, and should be included after the application-specific code:

```
' Close connection
ResultSet.Close
Connection.Close
```

OpenOffice.org Calc

```
Dim Doc As Object
Dim Sheet As Object
Dim Cell As Object
' Create a new empty spreadsheet.
Doc = ThisComponent 'StarDesktop.CurrentComponent
Sheet = Doc.Sheets(0)
If Doc.Sheets.hasByName("Class Report") Then
  Sheet = Doc.Sheets.getByName("Class Report")
Else
  Sheet = Doc.createInstance("com.sun.star.sheet.Spreadsheet")
  Doc.Sheets.insertByName("Class Report", Sheet)
End If
' Write table to Calc sheet
'this loop copies each field name into the 1st row of the sheet
row = 0
For column = 0 To (ResultSet.Columns.Count - 1)
  Cell = Sheet.getCellByPosition(column, 0)
  Cell.Formula = ResultSet.Columns(column).Name
Next
While ResultSet.Next()
  row = row + 1
  'this loop copies the data from each field to the new table
  For column = 0 To (ResultSet.Columns.Count - 1)
    Cell = Sheet.getCellByPosition(column, row)
    If ResultSet.Columns(column).Type = 4 or
       ResultSet.Columns(column).Type = 7 Then
      Cell.Value = ResultSet.Columns(column).getValue()
    Else
      Cell.Formula = ResultSet.Columns(column).getString()
    End If
  Next
Wend
```

OpenOffice.org Writer

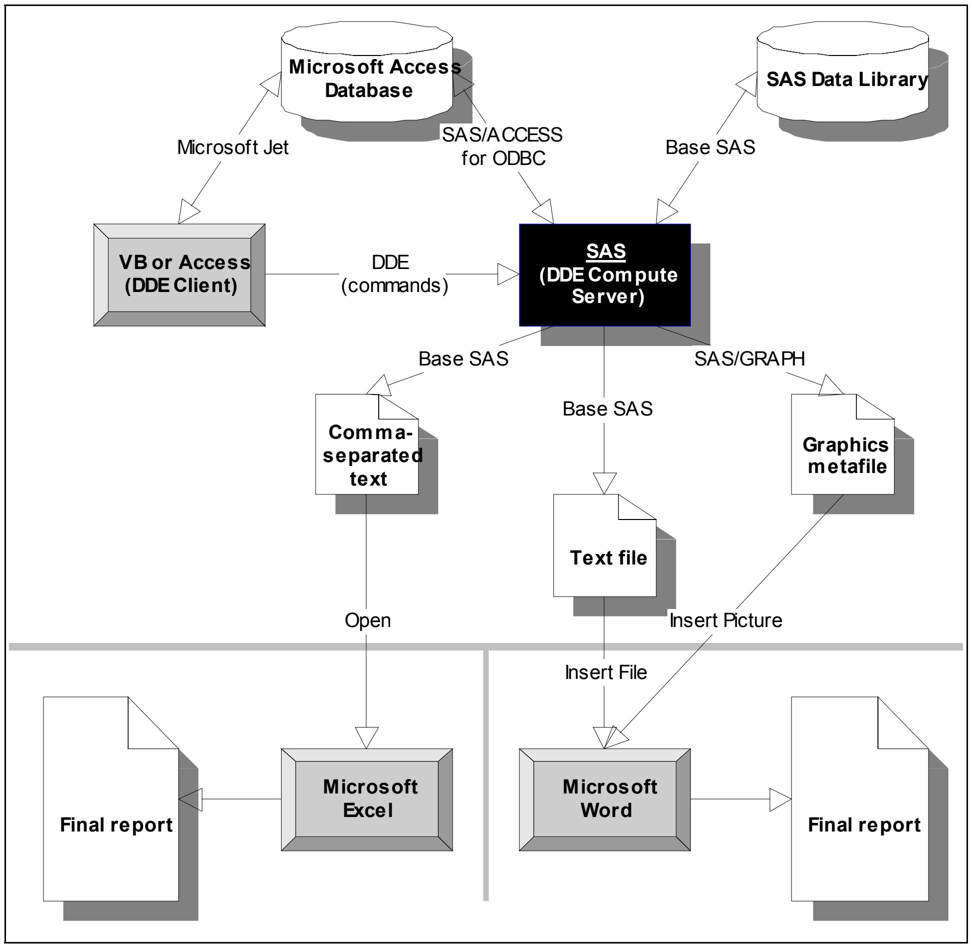
```
Dim Doc As Object
Dim Table As Object
Dim Cursor As Object
' Allocate this text document.
Doc = ThisComponent
' Create position cursor
Cursor = Doc.Text.createTextCursor()
' insert table at the end of the document
Cursor.gotoEnd(false)
' insert TextTable
Table = Doc.createInstance("com.sun.star.text.TextTable")
' initialize the table with the correct number of columns + rows
Table.initialize(1, ResultSet.Columns.Count)
' Write table to Text doc
Table.RepeatHeadline = True
Doc.Text.insertTextContent(Cursor, Table, False)
'this loop copies each field name into the 1st row of the table
row = 0
For column = 0 To (ResultSet.Columns.Count - 1)
  Table.getCellByPosition(column, 0)
       .setString(ResultSet.Columns(column).Name)
Next
Table.getRows(0)
While ResultSet.Next()
  row = row + 1
  Table.getRows().insertByIndex(row, 1)
  'this loop copies the data from each field to the new table
  For column = 0 To (ResultSet.Columns.Count - 1)
    Table.getCellByPosition(column, row)
         .setString(ResultSet.Columns(column).getString())
  Next
Wend
```

1.4 Dynamic Data Exchange

In the same way that SAS can be used to start and access Windows applications, SAS can be accessed by other Windows applications using Dynamic Data Exchange (DDE) operations. The DDE interface allows data to be sent to SAS via code (e.g., %LET statements, DATA step code with instream data, etc.). It is possible for SAS Display Manager commands to be executed, menu items to be clicked, SAS code to be written into the Program Editor—in fact any SAS operations that can be performed using the keyboard—using SendKeys instructions that include special string values for special keys, e.g., ALT, CTRL, ESC, ENTER, F1, etc.

Data cannot easily be passed back to the calling program via DDE, but data can be written to a file that can be read by the calling application using SAS code, e.g., as a Microsoft Access table using SAS/ACCESS for ODBC, as comma-separated text to a text file, or as a computer graphics metafile (e.g., *.CGM, *.WMF) using SAS/GRAPH, etc.

Figure 1.1 An Example of a SAS DDE Server at the Center of a Reporting System



1.4.1 Visual Basic 6.0

Visual Basic 6.0 allows any Visual Basic application to start SAS using a Windows Shell function. When the SAS session has been initialized, commands can be sent to SAS using the SendKeys function, e.g.:

```
'start SAS in maximized window
rc = Shell("d:\sas.exe",vbMaximizedFocus)
'wait for SAS to initialize
For i = 1 To 1000
  DoEvents()
Next
'Globals, Program Editor
SendKeys "%GP",True
'set SAS macro variable
SendKeys "{%}let macrovar = 1234;~",True
'run AF application
SendKeys "dm 'af c=lib.cat.prog.frame' af;~",True
'Locals, Submit
SendKeys "%LS"
'Globals, Program Editor
SendKeys "%GP",True
'Exit SAS
SendKeys "endsas;~",True
'Locals, Submit
SendKeys "%LS"
```

Additional processing by the Visual Basic application would be required if the SAS part of the application allows user input. The Visual Basic application must be hidden from view while the SAS part is executing. One way of performing this is to have a regularly executed function that initially creates a temporary "lock" file, and then hides the Visual Basic window while this file exists. The SAS part can then be seen and run as required. When the SAS processing has been completed, SAS can delete the lock file to tell Visual Basic to restore the visibility of the calling window.

1.5 SAS Integration Technologies

SAS Integration Technologies provides a number of useful external interfaces to SAS data, including an OLE DB driver, which can be accessed using ActiveX Data Objects (ADO), and a SAS Workspace Manager, which provides facilities to execute SAS code. It should be noted that, while access to SAS data via a remote SAS system requires that the SAS Integration Technologies component be licensed on that remote system, access to a local SAS system only requires the installation and licensing of Base SAS.

1.5.1 Visual Basic 6.0

Although the two types of libraries required to access the facilities within SAS Integration Technologies, SAS Integrated Object Model (IOM) and SAS Workspace Manager, are installed at the same time as the client software, they must still be added to the references within your Visual Basic project using the References menu item.

Access to SAS data sets via the OLE DB driver using ADO is very similar to accessing them via the SAS ODBC driver using Data Access Objects (DAO). However, ADO and DAO are not completely interchangeable, e.g., the field attributes are not the same for corresponding data types, which means that the code below has a number of significant differences from that used to access the SAS data sets via ODBC and DAO.

OLE DB and ADO

Allowing for the fact that the data type codes for ADO and DAO are different, and the field attributes also have different names, in order to be able to copy the SAS data from the ADO connection to the dBase table, the target table has to be constructed to include the same fields:

```
Public Function CopyStructADO(adoRset As ADODB.Recordset, _
                              daoDB As DAO.Database, _
                              DBTable As String) As Integer
  Dim i As Integer
  Dim daoTDef As DAO.TableDef
  Dim tmpTDef As DAO.TableDef
  Dim daoFld As DAO.Field
  Dim adoFld As ADODB.Field
  Dim errorFlag As Boolean
  ' Search to see if table exists
  For i = 0 To daoDB.TableDefs.Count - 1
    Set tmpTDef = daoDB.TableDefs(i)
    If UCase(tmpTDef.Name) = UCase(DBTable) Then
      daoDB.TableDefs.Delete tmpTDef.Name
      Exit For
    End If
  Next
  Set daoTDef = daoDB.CreateTableDef(DBTable)
  ' Create fields
  For i = 0 To adoRset.Fields.Count - 1
    Set adoFld = adoRset.Fields(i)
    Set daoFld = Nothing
    errorFlag = True
    ' Convert ADO field types to DAO equivalents
```

```
      Select Case adoFld.Type
        Case adDouble
          Set daoFld = daoTDef.CreateField(CVar(adoFld.Name), _
                                        Cvar(dbDouble))
          errorFlag = False
        Case adChar
          Set daoFld = daoTDef.CreateField(CVar(adoFld.Name), _
                                        Cvar(dbText), _
                                        Cvar(adoFld.DefinedSize))
          errorFlag = False
      End Select
      If Not errorFlag Then
        daoTDef.Fields.Append daoFld
      End If
    Next
    ' Append new table
    daoDB.TableDefs.Append daoTDef
    CopyStructADO = True
  End Function
```

Once the table structure has been copied and the two tables opened as recordsets, the data itself can then be copied record by record, 1,000 records at a time, using transactions:

```
  Public Function CopyDataADO(defWSpace As Workspace, _
                              adoRset As ADODB.Recordset, _
                              daoRset As DAO.Recordset) As Integer
    Dim i As Integer
    Dim j As Integer
    Dim numRecords As Integer
    Dim adoFld As ADODB.Field
    Dim daoFld As DAO.Field
    ' Start workspace transactions
    defWSpace.BeginTrans
    While adoRset.EOF = False
      daoRset.AddNew
      ' Loop copies data from each field to new table
      For i = 0 To (adoRset.Fields.Count - 1)
        Set adoFld = adoRset.Fields(i)
        For j = 0 To (daoRset.Fields.Count - 1)
          Set daoFld = daoRset.Fields(j)
          If UCase(adoFld.Name) = UCase(daoFld.Name) Then
            daoRset(daoFld.Name).Value = adoFld.Value
            Exit For
          End If
        Next
      Next
      daoRset.Update
      adoRset.MoveNext
      numRecords = numRecords + 1
      ' Commit transactions every 1000 records
      If numRecords = 1000 Then
        defWSpace.CommitTrans
        defWSpace.BeginTrans
        numRecords = 0
      End If
    Wend
    ' Commit changes now we have finished
    defWSpace.CommitTrans
    CopyDataADO = True
  End Function
```

The functions already described can be brought together to copy the structure and data from the source table to the target table as follows:

```
Public Sub SaveADOToDAO(defWSpace As Workspace, _
                        adoRset As ADODB.Recordset, _
                        daoDB As DAO.Database, _
                        DBTable As String)
  Dim daoRset As DAO.Recordset
  Dim i As Integer
  If CopyStructADO(adoRset, daoDB, DBTable) Then
    Set daoRset = daoDB.OpenRecordset(DBTable)
    If CopyDataADO(defWSpace, adoRset, daoRset) Then
      daoRset.Close
    End If
  End If
End Sub
```

All that is left to do is to specify the location of the SAS data library and the source and target tables:

```
Public Sub ADOread()
  ' Create ADO and DAO objects
  Dim adoConnection As New ADODB.Connection
  Dim adoRset As New ADODB.Recordset
  Dim defWSpace As Workspace
  Dim daoDB As DAO.Database
  Dim FilePath As String
  Dim DataSet As String
  Dim DBFilePath As String
  Dim DBTable As String
  Dim defFileSystem As Object
  Dim adoFileObject As Object
  Dim DBFileObject As Object
  ' Set file and database folder values
  DataSet = "hello"
  FilePath = "c:\temp"
  DBTable = "hello1"
  DBFilePath = "c:\temp"
  ' Verify path
  Set defFileSystem = CreateObject("Scripting.FileSystemObject")
  Set adoFileObject = defFileSystem.GetFolder(FilePath)
  Set DBFileObject = defFileSystem.GetFolder(DBFilePath)
  ' Get default workspace.
  Set defWSpace = DBEngine.Workspaces(0)
  ' Make sure there isn't already a file with the same name
  ' in the folder.
  If Dir(Trim$(DBFileObject.Path) & _
         "\" & _
         Trim$(DBTable) & ".dbf") <> "" Then
    Kill Trim$(DBFileObject.Path) & "\" & Trim$(DBTable) & ".dbf"
  End If
  ' Open database
  Set daoDB = defWSpace.OpenDatabase(Trim$(DBFileObject.Path), _
                                     False, _
                                     False, _
                                     "dBase IV;")
  ' Set initialization properties
  adoConnection.Provider = "SAS.LocalProvider.1"
  adoConnection.Properties("Data Source") = adoFileObject.Path
  ' Open connection
  adoConnection.Open
  ' Open recordset
  adoRset.Open DataSet, _
               adoConnection, _
               adOpenForwardOnly, _
               adLockReadOnly, _
               adCmdTableDirect
  Call SaveADOToDAO(defWSpace, _
                    adoRset, _
                    daoDB, _
                    DBTable)
  ' Close connection
  adoConnection.Close
  Set adoConnection = Nothing
  daoDB.Close
  Set defWSpace = Nothing
End Sub
```

SAS Workspace Manager

The SAS Workspace Manager provides a number of objects that can be used to access the processing power of SAS. In the following examples, SAS will be installed on the same system as the Visual Basic application, thereby not requiring any additional SAS component licensing.

The first example demonstrates how to run a simple SAS program in a local SAS session by submitting code stored in a string array. The SAS.LanguageService object is used to submit the code. The SAS output from the execution of these code lines is written to a text file located in the same folder as the Visual Basic application:

```
Public Sub Wsrun()
  ' Create workspace on local machine using Workspace Manager
  Dim sasWSMgr As New SASWorkspaceManager.WorkspaceManager
  Dim sasWSpace As SAS.Workspace
  Dim errorString As String
  Set sasWSpace = sasWSMgr.Workspaces.CreateWorkspaceByServer _
                      ("My workspace", _
                       VisibilityNone, _
                       Nothing, _
                       "", _
                       "", _
                       errorString)
  Dim sasLangService As SAS.LanguageService
  ' Declare fixed size array of strings to hold input statements
  Dim arraySource(2) As String
  ' Declare dynamic array of strings to hold list output
  Dim arrayList() As String
  ' These arrays will return line types and carriage control
  Dim arrayCC() As LanguageServiceCarriageControl
  Dim arrayLT() As LanguageServiceLineType
  Dim vOutLine As Variant
  arraySource(0) = _
            "data loop; do x=1 to 10; y=2-x; output; end; run;"
  arraySource(1) = "proc print; title 'Loop code';"
  arraySource(2) = "run;"
  Set sasLangService = sasWSpace.LanguageService
  sasLangService.SubmitLines arraySource
  ' Get up to 1000 lines of output
  sasLangService.FlushListLines 1000, arrayCC, arrayLT, arrayList
  ' Print each name in returned array
  Open ".\loop.txt" For Output As #1
  For Each vOutLine In arrayList
    Print #1, vOutLine
  Next
  Close #1
  sasWSpace.Close
End Sub
```

The second example demonstrates how to use the SAS.DataService object to query the environment of the SAS session to find information about which SAS library references exist. The SAS output from the investigation is written to a text file located in the same folder as the Visual Basic application:

```
  Public Sub WSlibname()
    ' Create workspace on local machine using Workspace Manager
    Dim sasWSMgr As New SASWorkspaceManager.WorkspaceManager
    Dim sasWSpace As SAS.Workspace
    Dim errorString As String
    Dim vName As Variant
    ' Declare dynamic array of strings to hold libnames
    Dim arrayLibnames() As String
    Set sasWSpace = sasWSMgr.Workspaces.CreateWorkspaceByServer _
                        ("My workspace", _
                         VisibilityNone, _
                         Nothing, _
                         "", _
                         "", _
                         errorString)
    ' Get reference to workspace's DataService.
    Dim sasDService As SAS.DataService
    Set sasDService = sasWSpace.DataService
    ' Assign libref named "saslib2" within new workspace
    Dim sasLibref As SAS.Libref
    Set sasLibref = sasDService.AssignLibref _
                        ("saslib2", _
                         "", _
                         ".\", _
                         "")
    ' Should print "saslib2"
    Open ".\libref.log" For Output As #2
    Print #2, "Newest libname = " & sasLibref.Name
    Print #2, " "
    ' Pass dynamic array variable to "ListLibrefs".
    ' Upon return, array variable will be filled in with array
    ' of strings one element for each libref in workspace
    sasDService.ListLibrefs arrayLibnames
    ' Print each name in returned array
    For Each vName In arrayLibnames
      Print #2, vName
    Next
    ' Print size of array
    Print #2, " "
    Print #2, "Number of librefs was: " & UBound(arrayLibnames) + 1
    Close #2
    ' Deassign libref.
    sasDService.DeassignLibref sasLibref.Name
    ' Close workspace.
    sasWSpace.Close
  End Sub
```

SAS Workspace Manager, IOM, and ADO

The SAS Workspace Manager also provides an interface suitable for accessing SAS data sets via ADO. As was seen in the preceding example for the SAS Workspace Manager, the SAS.LanguageService object is used to submit some SAS code to create a temporary SAS data set. The ADO interface then uses the local SAS IOM data provider, SAS.IOMProvider.1, to access the temporary SAS data set. The code then calls the SaveADOToDAO subroutine, described earlier in the section about OLE DB and ADO, to copy its structure and contents to a dBase file:

```
  Public Sub WSADOread()
    ' Create workspace on local machine using Workspace Manager
    Dim sasWSpace As SAS.Workspace
    Dim sasWSMgr As New SASWorkspaceManager.WorkspaceManager
    Dim errorString As String
    ' Create ADO and DAO objects
    Dim adoConnection As New ADODB.Connection
    Dim adoRset As New ADODB.Recordset
    Dim defWSpace As Workspace
    Dim daoDB As DAO.Database
    Dim daoRset As DAO.Recordset
    Dim DBFilePath As String
    Dim DBTable As String
    Dim defFileSystem As Object
    Dim DBFileObject As Object
    ' Set file and database folder values
    DBTable = "looping"
    DBFilePath = "c:\temp"
    ' Verify path
    Set defFileSystem = CreateObject("Scripting.FileSystemObject")
    Set DBFileObject = defFileSystem.GetFolder(DBFilePath)
    ' Get default workspace.
    Set defWSpace = DBEngine.Workspaces(0)
    ' Make sure there isn't already a file with the same name
    ' in the folder.
    If Dir(Trim$(DBFileObject.Path) & _
           "\" & _
           Trim$(DBTable) & ".dbf") <> "" Then
       Kill Trim$(DBFileObject.Path) & "\" & Trim$(DBTable) & ".dbf"
    End If
    ' Open database
    Set daoDB = defWSpace.OpenDatabase(Trim$(DBFileObject.Path), _
                                       True, _
                                       False, _
                                       "dBase IV;")
    Set sasWSpace = sasWSMgr.Workspaces.CreateWorkspaceByServer _
                        ("MyWorkspaceName", _
                         VisibilityProcess, _
                         Nothing, _
                         "", _
                         "", _
                         errorString)
    ' Submit SAS code
    sasWSpace.LanguageService.Submit _
        "data looping; do x=1 to 50; y=100; z=x*x; output; run;"
    ' Connect to local SAS IOM data provider
    adoConnection.Open _
        "Provider=SAS.IOMProvider.1; SAS Workspace ID=" + _
        sasWSpace.UniqueIdentifier
       ' Read temporary SAS data set
    adoRset.Open "work.looping", _
                 adoConnection, _
                 adOpenStatic, _
                 adLockReadOnly, _
                 adCmdTableDirect
    ' Copy the data to dBase file
    Call SaveADOToDAO(defWSpace, _
                      adoRset, _
                      daoDB, _
                      DBTable)
    ' Close connections
    adoConnection.Close
    Set adoConnection = Nothing
    daoDB.Close
    Set defWSpace = Nothing
    ' If we don't close SAS, the SAS process may stay around
    ' forever
    If Not (sasWSpace Is Nothing) Then
      sasWSMgr.Workspaces.RemoveWorkspace sasWSpace
      sasWSpace.Close
    End If
  End Sub
```

1.5.2 LotusScript Version 9

Lotus Word Pro 9 with SAS Workspace Manager, IOM, and ADO

The SAS Workspace Manager also provides an interface suitable for accessing SAS data sets via ADO, which can also be used within the applications that form Lotus SmartSuite. The following code extracts the data from a SAS data set called CLASS, which is located in the folder **'c:\temp\sas'** (note that the file separators need to be changed in the code to UNIX-style separators!), and then writes it record by record to the end of a Lotus Word Pro document.

```
Sub ADORead_WP()
  Dim vConn As Variant
  ' create connect and recordset options
  Set vConn = CreateObject("ADODB.Connection")
  Set rs = CreateObject("ADODB.recordset")
  ' open the connection to the mdb
  vConn.Provider = "SAS.LocalProvider.1"
  vConn.Open "c:/temp/sas"
  rs.Open "class", vConn, 0, 1, 512
  ' Write the data to the end of the current document.
  .Type "[ctrlEnd]"
  iNumField = rs.Fields.Count
  .Type "[ENTER]Num Fields is " & Str(iNumField) & "[ENTER]"
  rs.MoveFirst
  Do While Not rs.EOF
    .Type "[ENTER]" & rs.Fields("Name").Value
    .Type "[TAB]" & rs.Fields("Age").Value
    .Type "[TAB]" & rs.Fields("Sex").value
    .Type "[TAB]" & rs.Fields("Height").Value
    .Type "[TAB]" & rs.Fields("Weight").Value
    rs.MoveNext
  Loop
  rs.close
  vConn.Close
  Set rs = Nothing
  Set vConn = Nothing
End Sub
```

Lotus 1-2-3 9 with SAS Workspace Manager, IOM, and ADO

The following code performs the same actions as above, but writes the data to cells starting at the top left cell of the first sheet in a Lotus 1-2-3 spreadsheet file. Again, the location of the input SAS data set must be specified in the code with file separators changed to be UNIX-style.

```
Sub ADORead_123()
  Dim vConn As Variant
  Dim row As Long
  Dim report As Range
  Set report = [A:A3..A:E65535]
  ' create connect and recordset options
  Set vConn = CreateObject("ADODB.Connection")
  Set rs = CreateObject("ADODB.recordset")
  ' open the connection to the mdb
  vConn.Provider = "SAS.LocalProvider.1"
  vConn.Open "c:/temp/sas"
  rs.Open "class", vConn, 0, 1, 512
  iNumField = rs.Fields.Count
  [A:A1].Contents = "Num Fields is " & Str(iNumField)
  [A:A2].Contents = "Name"
  [A:B2].Contents = "Age"
  [A:C2].Contents = "Sex"
  [A:D2].Contents = "Height"
  [A:E2].Contents = "Weight"
  rs.MoveFirst
  row = 0
  Do While Not rs.EOF
    report.Cell(row,0).Contents = rs.Fields("Name").Value
    report.Cell(row,1).Contents = rs.Fields("Age").Value
    report.Cell(row,2).Contents = rs.Fields("Sex").Value
    report.Cell(row,3).Contents = rs.Fields("Height").Value
    report.Cell(row,4).Contents = rs.Fields("Weight").Value
    rs.MoveNext
    row = row + 1
  Loop
  rs.close
  vConn.Close
  Set rs = Nothing
  Set vConn = Nothing
End Sub
```

1.5.3 OpenOffice.org 2.1

It is easier to use Visual Basic to access the SAS Integration Technologies services, and then write directly into an OpenOffice.org Calc spreadsheet or into a table in an OpenOffice.org Writer document.

The following code is common to both processes, and should be included before the application-specific code:

```
Dim oServiceManager As Object
Dim oDesktop As Object
' Get the Service Manager object.
Set oServiceManager = _
    CreateObject("com.sun.star.ServiceManager")
' Get the Desktop object.
Set oDesktop = _
    oServiceManager.createInstance("com.sun.star.frame.Desktop")
Dim adoConnection As New ADODB.Connection
Dim adoRecordset As New ADODB.Recordset
Dim FilePath As String
Dim DataSet As String
Dim defFileSystem As Object
Dim adoFileObject As Object
Dim adoField As Object
Dim row As Long
Dim column As Long
' Use this empty array when no arguments are needed.
Dim aNoArgs()
' Set file values
FilePath = "c:\temp\sas"
DataSet = "class"
' Verify path
Set defFileSystem = CreateObject("Scripting.FileSystemObject")
Set adoFileObject = defFileSystem.GetFolder(FilePath)
' Set initialization properties
adoConnection.Provider = "SAS.LocalProvider.1"
adoConnection.Properties("Data Source") = adoFileObject.Path
' Open the Connection and display its properties
adoConnection.Open
' Open the Recordset
adoRecordset.Open DataSet, _
                  adoConnection, _
                  adOpenForwardOnly, _
                  adLockReadOnly, _
                  ADODB.adCmdTableDirect
```

The following code is common to both processes, and should be included after the application-specific code:

```
' Close the Connection
adoConnection.Close
Set adoConnection = Nothing
```

OpenOffice.org Calc

```
Dim oCalcDoc As Object
Dim oSheet As Object
' Create a new empty spreadsheet.
Set oCalcDoc = _
    oDesktop.loadComponentFromURL("private:factory/scalc", _
                              "_blank", 0, aNoArgs())
' Get the first spreadsheet from the sheets in the document.
Set oSheet = oCalcDoc.getSheets().getByIndex(0)
' Write table to Calc sheet
'this loop copies each field name into the 1st row of the sheet
row = 0
For column = 0 To (adoRecordset.Fields.Count - 1)
    Set adoField = adoRecordset.Fields(column)
    oSheet.getCellByPosition(column, 0).setFormula _
          CStr(adoField.Name)
Next
While adoRecordset.EOF = False
    row = row + 1
    'this loop copies the data from each field to the new table
    For column = 0 To (adoRecordset.Fields.Count - 1)
        Set adoField = adoRecordset.Fields(column)
        If adoField.Type = 200 Then
            oSheet.getCellByPosition(column, row).setFormula _
                  CStr(adoField.Value)
        Else
            oSheet.getCellByPosition(column, row).setValue _
                  adoField.Value
        End If
    Next
    adoRecordset.MoveNext
Wend
```

OpenOffice.org Writer

```
Dim oText As Object
Dim oText2 As Object
Dim oTable As Object
Dim oCursor As Object
' Create a new blank text document.
Set oText = _
      oDesktop.loadComponentFromURL("private:factory/swriter", _
                                    "_blank", 0, aNoArgs())
' insert TextTable
Set oTable = oText.createInstance("com.sun.star.text.TextTable")
' Create position cursor
Set oText2 = oText.GetText()
Set oCursor = oText2.createTextCursor()
' Write table to Text doc
' initialize the table with the correct number of columns + rows
oTable.Initialize 1, CLng(adoRecordset.Fields.Count)
oTable.RepeatHeadline = True
' insert table now
oCursor.gotoStart False
oText2.insertTextContent oCursor, oTable, False
'this loop copies each field name into the 1st row of the table
row = 0
For column = 0 To (adoRecordset.Fields.Count - 1)
    Set adoField = adoRecordset.Fields(column)
    oTable.getCellByPosition(column, 0).String = _
          CStr(adoField.Name)
Next
adoRecordset.MoveFirst
While adoRecordset.EOF = False
    row = row + 1
    oTable.GetRows().insertByIndex row, 1
    'this loop copies the data from each field to the new table
    For column = 0 To (adoRecordset.Fields.Count - 1)
        Set adoField = adoRecordset.Fields(column)
        oTable.getCellByPosition(column, row).String = _
              CStr(adoField.Value)
    Next
    adoRecordset.MoveNext
Wend
```

1.6 Conclusions

This chapter has only scraped the surface of what is possible using SAS as a file server or compute server for other Windows-based applications. It should now be clear that a large number of different PC users could benefit from using SAS effectively as a “black box” processor with their own applications, reducing the need to fully train them in SAS coding techniques. The SAS data libraries and SAS application development can be done for them by SAS specialists, providing the users with a well-documented and stable interface that they can use without any requirement for prior knowledge of SAS.

1.7 Recommended Reading

For more information, go to www.hollandnumerics.com/books/Saving_Time_and_Money_using_SAS.htm. This page includes a chapter-by-chapter list of recommended reading.