

OpenOffice.org Base Macro Programming By Andrew Pitonyak

Last Modified

Tuesday, July 28, 2009 at 09:54:04 PM

Document Revision: 43

Information Page

Copyright

This document is Copyright © 2005-2009 by its contributors as listed in the section titled **Authors**. You can distribute it and/or modify it under the terms of the Creative Commons Attribution License, version 2.0 or later (<http://creativecommons.org/licenses/by/2.0/>).

All trademarks within this guide belong to their legitimate owners.

Authors

Andrew Pitonyak

Feedback

Maintainer: Andrew Pitonyak [andrew@pitonyak.org]

Please direct any comments or suggestions about this document to:
authors@user-faq.openoffice.org

Acknowledgments

I find it difficult to properly credit all of my sources, because so many people are helpful in an endeavour of this size. There are, however, a few people who do indeed stand out in my mind as having provided significant encouragement.

I have no explanation as to precisely why my wife **Michelle** allows me to spend so much time working with OpenOffice.org. Perhaps she is really the person that you should thank for my productivity. I Love you Michelle, you complete me.

All of the people with whom I have interacted at Sun Microsystems have been very tolerant and patient with my endless questions. In the creation of this document, **Frank Schönheit**, however stands out in this regard. Mr. Schönheit spends a lot of time helping people with problems and questions, and most notable for me, he answers my questions. Thank you Frank!

There is a large community volunteering their time with OpenOffice.org. **Drew Jensen** has stood out in my mind as an incredibly prolific and knowledgeable individual. Drew clearly has vast experience using database products, and he has brought this vast experience to the OOo community. He has created numerous excellent examples on the OOo Forums and mailing lists. Thank you Drew.

There is a large community of helpers, who are simply too numerous to mention. I owe you all a thank you for your help and encouragement. In the general community, however, I will single out **G. Roderick Singleton**, who helps numerous people every day on the mailing lists. He also very proactive in keeping the documentation up-to-date. Mr. Singleton, I thank you for all of your help as well.

This document is updated a lot, so it hardly makes sense to track changes at this time. Perhaps when I come up with at least a version 1.0.

Table 1. Modification History

Date	Comment
9/23/06	Integrated changes from Jo < ml@winfix.it >
3/13/07	Moved document to a new format.
4/4/07	Discovered that I did NOT transfer the macros.
1/30/08	Comments related to fields in forms.
2/18/08	New changes coming for OOo 3.0; watch out!

I encourage people to turn on change tracking in OOo (**Edit > Changes > Record**) and make corrections, enhancements, and/or updates to this document. When you are finished, please send the document to me for integration into the final document. Please make note of the “Last Modified” date (Tuesday, July 28, 2009 at 09:54:04 PM) and the revision number (43) so that I will know if you have the latest document version, which makes my life much easier.

Table of Contents

Information Page	2
Copyright	2
Authors	2
Feedback	2
Acknowledgments	2
Table of Contents	v
1. Introduction	1
1.1. Introductory comments	1
1.2. Document organization and introduction	1
1.3. Prepare for big changes in OOo 3.0	2
2. Storing images (binary data) in Base	4
2.1. Create the initial Base document	4
2.1.1. Using the GUI	4
2.1.2. Using a macro	4
2.1.3. Using a macro to open the wizard	5
2.2. Create the table	5
2.2.1. Using the GUI	5
2.2.2. Using a macro	6
2.2.3. Using SQL statements to modify tables	8
2.2.4. Refresh the tables	8
2.2.5. Creating and deleting tables using SQL	9
2.2.6. Increase a field's length	11
2.3. Create a form	11
2.3.1. Using the GUI	11
2.3.2. Using a macro	13
2.4. Open a form using a macro	17
2.5. Accessing the binary data	20
2.5.1. Adding binary data	20
2.5.2. Extracting binary data	22
3. One-To-Many relationships	25
3.1. Create the tables	25
3.1.1. Create the DEALER table	25
3.1.2. Create the ITEM table	26
3.2. Define the data relationships	28
3.3. Add data to the DEALER and ITEM tables	29
4. Forms	32
4.1. The internal object model	32
4.1.1. A control's shape is in the draw page	32
4.1.2. A draw page contains forms	33
4.1.3. A control's data model is in a form	34
4.1.4. A control's view model is in the controller	35

4.1.5. Enabling and setting controls visible – an example	36
4.1.6. Finding a control from an event – an example	36
4.1.7. Control connected to a database	37
4.1.8. Control model summary	38
4.2. Database Forms act like a result set	38
4.2.1. Duplicate record macro	39
4.3. Show one item and the corresponding dealer	42
4.4. Use a combo box with the dealer id	44
4.5. Use a list box with the dealer name	45
4.6. Relations in a single table	47
4.6.1. Solution	47
4.6.2. Solution characteristics	48
4.7. Use a “help and fill” button	49
5. Many-to-many relationships	50
6. Database fields	51
6.1. Storing numbers	54
6.1.1. Integer numbers	54
6.1.2. Floating point numbers	55
6.1.3. NUMERIC and DECIMAL types	56
6.2. Bit and Boolean Types	56
6.3. Date and time	57
6.4. Text data	57
6.5. Binary data	58
6.6. Other data type	58
6.7. Database sequences and auto-value fields	58
7. A few easy database definitions	60
7.1. Schema	61
8. Database connections	62
8.1. Obtain a database context	62
8.1.1. Registered data sources	63
8.1.2. Unregistering a data source	63
8.1.3. Registering a data source	64
8.2. Connect to a database	64
8.3. Connect using an interaction handler	65
8.4. Connections	65
8.4.1. Extended SDB connections	67
8.4.2. Meta-data	67
8.4.3. Inspecting the meta-data	74
8.4.4. GetBestRowIdentifier	81
8.4.5. GetColumnPrivileges	82
8.4.6. GetColumns	83

8.4.7. GetExportedKeys	84
8.4.8. GetIndexInfo	85
8.4.9. GetPrimaryKeys	87
8.4.10. GetTablePrivileges	87
8.4.11. GetTables	88
8.4.12. GetTypeInfo()	88
8.4.13. GetUDTS	89
8.4.14. GetVersionColumns	90
8.5. Connections	91
8.6. Connections without a data source	91
8.6.1. Delimited text files	96
8.6.2. Fixed width text files	98
8.6.3. Help, I still can not import my CSV file	103
8.6.4. Address books	105
8.6.5. MySQL using JDBC	105
8.6.6. Paradox using ODBC	106
8.6.7. Conclusion	108
9. Connecting to MySQL using JDBC	109
10. Mailmerge	111
11. Copying an entire database	112
12. General utility macros	113
12.1. Choose a directory	114
12.2. Get a document's directory	115
12.3. Choose a file	115
12.4. Finding a (loaded) OOo document	117
12.5. Append to an array	119
12.6. Compare data in an array	119
12.7. Create a property	120
12.8. Create a Point and a Size	120
12.9. Append a data array to a Calc document	121
12.10. Dynamically call object methods	122
12.11. Display numeric constants as meaningful text	126
12.12. Select from a list in a list box	127
13. Database utility macros	130
13.1. Quoting table and field names	130
13.2. Convert between an UNO Date and a Basic Date	130
13.3. Convert a result set to an array of data	132
13.4. Create and populate a dialog from a result set	135
14. Tips and tricks	137
14.1. Limit the number of returned records	137
15. Connect to a Base document using JDBC	140

Appendix A. Stuff I Own	143
A.1. Tables	143
A.1.1. Category	143
A.1.2. Dealer	143
A.1.3. Images	144
A.1.4. Item	144
A.2. Forms	144
A.2.1. Item Two Tables	144
A.2.2. Item One Table	146
A.2.3. Item Fields	147
A.3. Add an image macro	148
A.4. Delete an image macro	149
A.5. Replace an image macro	150
A.6. Extract an image macro	152
A.7. Clean the database	153
A.8. Things to do	153

1. Introduction

1.1. *Introductory comments*

Although I was going to write a book on this subject, I have been discouraged from completing this project. I opted, instead, to create this somewhat fragmented, less time intensive document. Hopefully you will find it useful.

This document is not even remotely finished. If you find errors, or have some favorite additions, then please do the following:

- 1) Download the latest version of the document.
- 2) Make note of the “Last Modified” date (Tuesday, July 28, 2009 at 09:54:04 PM) and the revision number (43) so that I will know if you have the latest document version, which makes my life much easier.
- 3) Time permitting, warn me ahead of time so that I can send you the latest version if I have not posted it.
- 4) Use **Edit > Changes > Record** to turn on edit tracking.
- 5) I attempted to use the the styles and formatting recommended at the OOo Authors web site (see <http://www.oooauthors.org/>). The primary difference is that I do not embed place graphics in a frame with the caption. I prefer them to be in their own paragraph and not in a frame. I experienced bugs related to using frames for this, which caused me to lose information from this document; the information is still missing today.
- 6) Send the modified document to me.

I will incorporate the changes into the latest document and reformat the document to be compliant with the OOo Authors web site criteria if required. I really do appreciate bug reports, and if you desire to add sections or material, I am open to that as well. Thanks to Szymon Nikliborc, who provided the first bug report.

1.2. *Document organization and introduction*

The database component in OpenOffice.org (OOo) contains numerous complexities. In some ways the vast capabilities are mature, and in others they are not. I have solved many problems using Base and I add them to this document as I solve them and as I have time to add them. The advantage is that all of the problems end up in a single document. The disadvantage, however, is that coverage is disjointed and not consistent. Sometimes I assume that you know nothing about Base, and at other times I might assume that you are an expert user.

I, Andrew Pitonyak, was unable to find significant documentation dealing specifically with binary data stored in a Base document, so I decided to figure out how it works. This document starts by demonstrating how to use binary fields, with an emphasis on using macros to manipulate the data.

TIP The OOo Write version of this document contains the macros described in this document. The document also contains buttons that call the macros contained in this document. For obvious reasons, if this document is converted to a different format, such as a PDF or DOC, the macros will be lost and the buttons will not call the macros. In other words, if you are reading a PDF version of the document, the buttons don't work.

The initial section dealing with binary data provides easy steps for creating your first database. The binary section also demonstrates many useful methods such as creating and opening forms using macros.

This document contains a library named AndrewBase, which contains the main macros shown in this document. Buttons are inserted throughout the document to call the macros shown in the text. When OOo loads a document, only the Standard library is loaded, which means that the macros stored in the AndrewBase library are not available to be called from a button. The standard library contains helper macros, that wrap calls to the macros of interest. A typical helper macro, CallCreateBinaryDB, is shown in *Listing 1*. All helper macros start by calling LoadDBLibs, which loads the library containing the worker macros.

Listing 1: Macro used to create the empty Base document.

```
Const sDBBaseName$ = "BaseFieldDB.odt"

Sub LoadDBLibs()
    If NOT BasicLibraries.IsLibraryLoaded("AndrewBase") Then
        BasicLibraries.LoadLibrary("AndrewBase")
    End If
End Sub

Sub CallCreateBinaryDB()
    LoadDBLibs()
    CreateBinaryDB(GetSourceCodeDir() & sDBBaseName, True)
End Sub
```

Notice that the computer code uses syntax highlighting as is done by the Basic IDE. I feel that this enhances the readability of the code, so I wrote a macro that will search the entire document for computer code, and then create syntax highlighting.

1.3. Prepare for big changes in OOo 3.0

In OOo 2.x, a Base document can not contain macros, but the contained reports and forms can. In OOo 3.x, this is to be reversed; a Base document can contain macros and contained reports and forms can not.

http://wiki.services.openoffice.org/wiki/Macros_in_Database_Documents

With version 3.x, ThisComponent will always be the component which was active when the macro was invoked. This holds no matter whether the macro is located in the database document's or in the application's Basic library. Also, it holds no matter whether the active component is a database document or any of its sub components. In particular, the various designers are also available as ThisComponent. The trick is that ThisComponent may end up pointing to a database document, especially when running a macro from an IDE.

The variable ThisDatabaseDocument will be introduced for basic macros embedded in a Base document, and always refer to the Base document.

This may be confusing if a macro is invoked from a report, because forms and report definitions are documents.

ThisComponent always served two purposes: For a Basic macro embedded in a document, it refers to the containing document. For a Basic macro located elsewhere, it refers to the currently active document. Unfortunately, the term currently active is not well-defined across different platforms and window managers.

Now, macros can exist in database documents and are allowed to run macros in the sub components of the Base document. Unfortunately, this provides a contradiction, so a choice had to be made on which of the two meanings for ThisComponent to preserve.

The final decision was to keep the meaning for ThisComponent referring to the document that was invoked (so if a menu or button on a menu was used to call a macro) as opposed to the document containing the macro.

This does not really change much, except for cases when ThisComponent would have been undefined because the database documents, and its sub components (forms, reports, queries, tables, relation designer) didn't participate in the ThisComponent game (since the implementation of this global property was purely SFX based). So, if somebody wrote a (global) macro and triggered this from within one of the DB components/documents, ThisComponent was effectively undefined.

This scenario - executing such macros from with DB components - is the only one I know where 3.1 differs from 3.0, so there should be no issues.

2. Storing images (binary data) in Base

In this section, we will create a database that contains a field of type Image. An Image field is really a “long variable binary” field, which means that it can contain any type of binary data, not just images. If I choose to store images in my binary field, then I can use an Image viewing control in a form to see the images—and seeing the pretty picture allows for immediate feedback that things are working.

2.1. Create the initial Base document

You need a base document that will contain the image data.

2.1.1. Using the GUI

Use the following step by step instructions to create a sample database for use.

- 1) Use **File > New Database** to open the new database wizard.
- 2) Select the *Create a new database* radio button and click **Next**.
- 3) Select the *No, do not register the database* radio button, the *Open the database for editing* checkbox, and click **Finish**.
- 4) Name the database ImageDB and click **Save**.

2.1.2. Using a macro

Creating a Base document using a macro is easy, but it is easy to make a mistake in the details. There are a few key items to create a Base document.

- 1) Use the DatabaseContext to create an empty data source.
- 2) Set the data source URL to `sdbc:embedded:hsqldb` for an internal HSQL database.
- 3) Obtain the database document from the data source and save it. You can not add tables to a Base document until after it has been saved.

The macro in *Listing 2* demonstrates how to create a Base document. If the database URL is not specified, then a dialog asks for a file name. The filter list, which contains the Base file extensions is obtained from *Listing 59*, and then the macro in *Listing 58* displays the dialog asking for the new database name. Use the **Create Database** button `Create Database` to run the wrapper method which calls `CreateBinaryDB` (see *Listing 1* and *Listing 2*); this will create the `BaseFieldDB.odt` file in the same directory used by this document (see *Listing 57*).

Listing 2: Create an empty Base document.

```
REM Use "Option Compatible", or you can not use a default argument.
Sub CreateBinaryDB(Optional dbURL$ = "", Optional bVerbose = False)
    Dim oDBContext    'DatabaseContext service.
    Dim oDB            'Database data source.

    REM No URL Specified, get one.
    If dbURL = "" Then dbURL = ChooseAFile(OOoBaseFilters(), False)
```

```

REM Still No URL Specified, exit.
If dbURL = "" Then Exit Sub

If FileExists(dbURL) Then
  If bVerbose Then Print "The file already exists."
Else
  If bVerbose Then Print "Creating " & dbURL
  oDBContext = createUnoService( "com.sun.star.sdb.DatabaseContext" )
  oDB = oDBContext.CreateInstance()
  oDB.URL = "sdbc:embedded:hsqldb"
  oDB.DatabaseDocument.storeAsURL(dbURL, Array())
End If
End Sub

```

2.1.3. Using a macro to open the wizard

This is a little snippet that I have not tested, but according to Sevastian Foglia (sevastian.foglia@yacme.com), the following should work:

Listing 3: Start the Base document wizard using a macro.

```

sURL = "private:factory/sdatabase?Interactive"
doc = StarDesktop.loadComponentFromURL(sURL, "_blank", 0, args)

```

2.2. Create the table

The image database is intentionally very simple (see *Table 2*). The field names use uppercase characters and contain no spaces because it simplifies the SQL—you do not have to quote the field names in SQL statements. The annoying thing about quoting, is that the same quote character is not always used. The query builder might require that non-uppercase characters be quoted using a double quote character ("), but an SQL statement in a macro might require a back-tic (`). A macro that deals with this problem is demonstrated in *Listing 77*.

Table 2. Fields in the binary table.

Field	Field Type	Comment
ID	Integer [INTEGER]	Table's primary key
NAME	Text [VARCHAR]	Name for the data, most likely, a file name.
DATA	Image [LONGVARBINARY]	The binary data.

2.2.1. Using the GUI

Now, create the table to hold the image. Select **Tables** from the left hand side and then choose the **Create Table in Design View** task. Enter the fields in the table design window.

- 1) Create the primary key.
 - 1) Set the *Field Name* to ID.
 - 2) Set the *Field Type* to Integer.
 - 3) Set *Auto Value* to Yes.
 - 4) Right click to the left of the field name and choose **Primary Key**.

- 2) Create the name field.
 - 1) Set the *Field Name* to NAME.
 - 2) Set the *Field Type* to Text [VARCHAR].
 - 3) Set *Entry required* to Yes.
 - 4) Set *Length* to 255.
- 3) Create the image field.
 - 1) Set the *Field Name* to DATA.
 - 2) Set the *Field Type* to Image.
 - 3) Set *Entry required* to No.
 - 4) Leave the *Length* at the default value of 2147483647.

Use **File > Save** to save the table. Name the table BINDATA, and then use **File > Close** to close the table design window.

Caution


Saving the table saves the table definition into the Base document, but the document itself has not been saved. You must also save the Base document.

2.2.2. Using a macro

Use the **Create Binary Tables** button Create Binary Tables to run the macro in *Listing 4*. If the table exists, it will be deleted and recreated. Also, if the document does not yet exist, it will be created. The macro uses the standard OOo API.

Listing 4: Create a table in a Base document using the API.

```
REM Create the database specified by dbURL. If it
REM   does not exist, then it is created.
REM If bForceNew is True, then an existing table is deleted first.
REM If bVerbose is True, progress messages are printed.
Sub CreateBinaryTables(dbURL As String, _
    Optional bForceNew = False, _
    Optional bVerbose = False)
    Dim sTableName$      'The name of the table to creat.
    Dim oTable           'A table in the database.
    Dim oTables          'Tables in the document
    Dim oTableDescriptor 'Defines a table and how it looks.
    Dim oCols            'The columns for a table.
    Dim oCol             'A single column descriptor.
    Dim oCon             'Database connection.
    Dim oBaseContext    'Database context service.
    Dim oDB              'Database data source.

    REM If the database does not exist, then create it.
    If NOT FileExists(dbURL) Then
```

```

    CreateBinaryDB(dbURL, bVerbose)
End If

REM Use the DatabaseContext to get a reference to the database.
oBaseContext = CreateUnoService("com.sun.star.sdb.DatabaseContext")
oDB = oBaseContext.getByName(dbURL)
oCon = oDB.getConnection("", "")

oTables = oCon.getTables()

sTableName$ = "BINDATA"
If oTables.hasByName(sTableName$) Then
    If bForceNew Then
        If bVerbose Then Print "Deleting table " & sTableName
        oTables.dropByName(sTableName)
        oDB.DatabaseDocument.store()
        'oCon.close()
        'Exit Sub
    Else
        If bVerbose Then Print "Table " & sTableName & " already exists!"
        oCon.close()
        Exit Sub
    End If
End If

REM For now, this should always be True
If NOT oTables.hasByName(sTableName$) Then
    oTableDescriptor = oTables.createDataDescriptor()
    oTableDescriptor.Name = sTableName$

    oCols = oTableDescriptor.getColumns()
    oCol = oCols.createDataDescriptor()
    oCol.Name = "ID"
    oCol.Type = com.sun.star.sdbc.DataType.INTEGER
    oCol.IsNullable = com.sun.star.sdbc.ColumnValue.NO_NULLS
    oCol.IsAutoIncrement = True
    oCol.Precision = 10
    oCol.Description = "Primary Key"
    oCols.appendByDescriptor(oCol)

    oCol.Name = "NAME"
    oCol.Type = com.sun.star.sdbc.DataType.VARCHAR
    oCol.Description = "Filename"
    oCol.Precision = 255
    oCol.IsAutoIncrement = False
    oCols.appendByDescriptor(oCol)

    oCol.Name = "DATA"
    oCol.Type = com.sun.star.sdbc.DataType.LONGVARBINARY
    oCol.Precision = 2147483647
    oCol.IsNullable = com.sun.star.sdbc.ColumnValue.NULLABLE
    oCol.Description = "Binary Data"

```

```

oCols.AppendByDescriptor(oCol)

oTables.AppendByDescriptor(oTableDescriptor)
End If

REM Do not dispose the database context or you will NOT be able to
REM get it back without restarting OpenOffice.org.
REM Store the associated document to persist the changes to disk.
oDB.DatabaseDocument.store()
oCon.close()
If bVerbose Then Print "Table " & sTableName & " created!"
End Sub

```

TIP In my testing, I wanted to completely delete an OOo Base document and start over. Unfortunately, after using a Base document, OOo holds the file open, so OOo must be shutdown and restarted before the document can be deleted. This is a known bug in OOo version 2.0 and should be fixed in OOo 2.01.

2.2.3. Using SQL statements to modify tables

You can create and manage tables using SQL statements rather than the OOo API. The SQL statements differ depending on the back-end database system, so the SQL is database dependent—this is never a good thing. The OOo API does a good job of insulating you from the system specific details. Unfortunately, some things just can not be done using the API. For example, as of OOo version 2.0, you can only set default values to a constant value. It is not possible to default a time or date field to the current date or time—this is trivial using SQL.

2.2.4. Refresh the tables

Care must be taken when using SQL to modify a table, because the OOo GUI will not automatically notice that changes have been made. You must refresh the internal OOo structures. You can use **View > Refresh Tables** from the OOo Base GUI. A kind macro programmer will perform this task in the macro that modifies the database structure. Refreshing the table view should be simple.

Listing 5: Refresh the tables in an OOo Base document should be simple.

```
oCon.getTables().refresh()
```

Unfortunately, calling refresh (see *Listing 5*) does not always work; in my limited testing, it did not properly update when deleting tables, and it sometimes worked when adding a table. Using a dispatch (see *Listing 6*), worked for my few test cases. Unfortunately, the database must be loaded in the GUI to use a dispatch; hopefully this will be fixed in version 2.01. ??

Listing 6: Refresh the tables in an OOo Base document using a dispatch.

```

REM Using SQL DDL commands to modify the table structure bypasses
REM the normal OOo API, which does not give OOo an opportunity to
REM notice that the table structure has changed. Tell OOo to
REM refresh the table view.
Sub RefreshTables(sURL$, oCon)
    Dim oDoc    'Document to refresh.
    Dim oDisp   'Dispatch helper.
    Dim oFrame  'Current frame.

```

```

REM This should be the same as
REM   oCon.getTables().refresh()
REM but it is not...
oDoc = FindComponentWithURL(sURL, False)
If NOT IsNULL(oDOC) AND NOT IsEmpty(oDoc) Then
    oDisp = createUnoService("com.sun.star.frame.DispatchHelper")
    oFrame = oDoc.getCurrentController().getFrame()
    oDisp.executeDispatch(oFrame, ".uno:DBRefreshTables", "", 0, Array())
End If
End Sub

```

2.2.5. Creating and deleting tables using SQL

The macro in *Listing 7* performs the following tasks, which means that it demonstrates how to do them using SQL:

- 1) Determine if a table exists. To determine how to do this, I examined the meta data from the connection (see *Listing 35* and following).
- 2) Delete a table.
- 3) Create a new table.

Create Table Using SQL If the table exists, it is deleted and then the macro returns. If the table does not exist, then it is created. If any forms or other items rely on this table, then they will also be deleted; you have been warned.

Listing 7: Create a table in a Base document using the API.

```

REM Create the database specified by dbURL. If it
REM   does not exist, then it is created.
REM If bForceNew is True, then an existing table is deleted first.
REM If bVerbose is True, progress messages are printed.
Sub CreateBinaryTablesUseSQL(dbURL As String, _
    Optional bForceNew = False, _
    Optional bVerbose = False)
    Dim sTableName$      'The name of the table to creat.
    Dim oTable           'A table in the database.
    Dim oTables           'Tables in the document
    Dim oTableDescriptor 'Defines a table and how it looks.
    Dim oCols            'The columns for a table.
    Dim oCol             'A single column descriptor.
    Dim oCon             'Database connection.
    Dim oBaseContext     'Database context service.
    Dim oDB              'Database data source.
    Dim oResult          'Restul of executing an SQL statement.
    Dim nCount As Long   'Counting variable.

    Dim oStmt
    Dim sSql$

    REM If the database does not exist, then create it.
    If NOT FileExists(dbURL) Then

```



```

    CreateBinaryDB(dbURL, bVerbose)
End If

REM Use the DatabaseContext to get a reference to the database.
oBaseContext = CreateUnoService("com.sun.star.sdb.DatabaseContext")
oDB = oBaseContext.getByName(dbURL)
oCon = oDB.getConnection("", "")

oStmt = oCon.createStatement()
sTableName$ = "BINDATA"

REM First, check to see if the table exists!
sSql = "select count(*) from INFORMATION_SCHEMA.SYSTEM_TABLES " & _
      "where TABLE_NAME='" & sTableName & "' " & _
      "AND TABLE_SCHEM='PUBLIC'"

nCount = 0
oResult = oStmt.executeQuery(sSql)
If NOT IsNull(oResult) AND NOT IsEmpty(oResult) Then
    oResult.Next()
    nCount = oResult.getLong(1)
End If

If nCount <> 0 Then
    If bForceNew Then
        If bVerbose Then Print "Deleting table " & sTableName
        REM The default behavior is to use RESTRICT rather than CASCADE.
        REM RESTRICT prevents the deletion if other things depend on
        REM this table.
        sSql = "DROP TABLE " & _
              DBQuoteName(sTablename, oCon) & _
              "IF EXISTS CASCADE"
        oStmt.executeQuery(sSql)
        RefreshTables(dbURL$, oCon)
        oCon.close()
        Exit Sub
    Else
        If bVerbose Then Print "Table " & sTableName & " already exists!"
        oCon.close()
        Exit Sub
    End If
End If

REM I did not quote the field names because I know that
REM they are all uppercase with nothing special about them.
sSql = "CREATE TABLE " & _
      DBQuoteName(sTableName, oCon) & _
      "(ID INTEGER NOT NULL IDENTITY PRIMARY KEY, " & _
      " NAME VARCHAR(255) NULL, " & _
      " DATA LONGVARBINARY NULL)"

oStmt.executeQuery(sSql)

```

```

If bVerbose Then Print "Created table in " & dbURL
RefreshTables(dbURL$, oCon)

REM Do not dispose the database context or you will NOT be able to
REM get it back without restarting OpenOffice.org.
REM Store the associated document to persist the changes to disk.
oDB.DatabaseDocument.store()
oCon.close()
If bVerbose Then Print "Table " & sTableName & " created!"
End Sub

```

2.2.6. Increase a field's length

I needed to increase the length of a field, but was unable to save my change. OOo offers to delete the field and insert a new field. This removes all data associated with that field. I used the following method to increase the length of the text field. In the following example, I modify the field named “COMMENT” in the ITEM table.

- 1) From the Base document, right click on the table and choose edit.
- 2) Rename the COMMENT field to something else such as COMMENT1 and save the change.
- 3) Add a new field named COMMENT with the desired length or properties and save the change.
- 4) From the Base document, use **Tools > SQL** to open the Execute SQL Statement dialog.
- 5) Enter the desired SQL statement to copy the content from field COMMENT1 to COMMENT; I used “UPDATE ITEM SET COMMENT=COMMENT1”. Remember that ITEM is the table name.
- 6) Open the ITEM table and verify that the data has been copied.
- 7) From the table design window (remember right click on the table and choose edit), delete the old COMMENT1 field leaving only the new COMMENT field.

These steps may require a few changes if the field is used someplace else (for example, in a relation set using **Tools > Relationships**).

2.3. Create a form

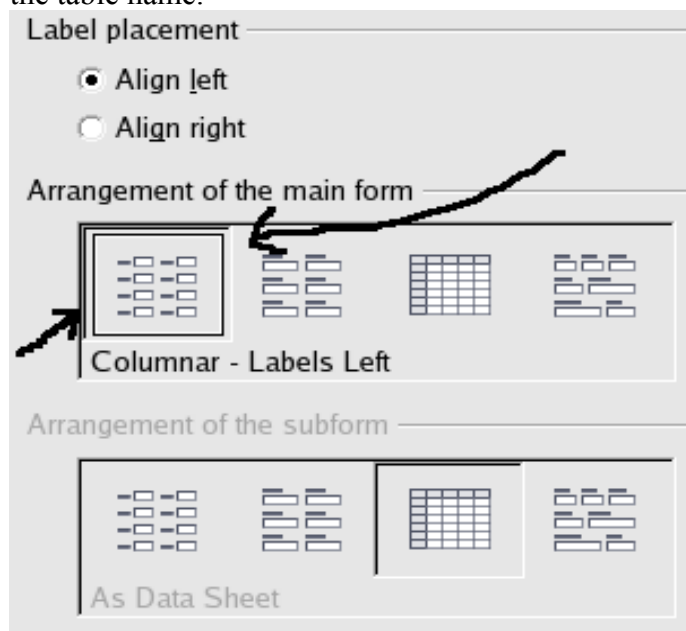
2.3.1. Using the GUI


I want a simple form so that I can insert images. Although OOo calls a LONGVARBINARY field an Image field, you can store any type of binary data in an Image field. Although there is an Image field that is able to display images, an automated binary field viewer does not exist. OOo does not assume that all binary fields contain image data, which is why binary fields are not added to a form using the form wizard.


TIP To add an Image control to view an Image field, you must manually edit the form after using the form wizard.

1) Create the initial form using the Wizard

- 1) Select **Forms** on the left hand side and then choose the **Use Wizard to Create Form** task.
- 2) Set the *tables or queries* field Table:BINDATA.
- 3) Click on the >> button to add all available fields to the form. This will only include the ID and NAME field. Image fields are not included in the wizard in OOo version 2.0.
- 4) Click **Next** to continue to the sub forms page.
- 5) Click **Next** to continue without creating a sub form.
- 6) Select the Columnar for a single record per form and click **Next**. If you click **Finish**, the form will automatically be saved using the name BINDATA, which corresponds to the table name.



- 7) Ignore the data entry mode and click **Next**.
- 8) Ignore the styles and click **Next**.
- 9) Set the name to "BINDATAImage" and click **Finish**.
- 2) The form is automatically opened for editing. Close the form using **File > Close**.
- 3) Now, open the form in edit mode and add an Image control.
 - 1) Select **Forms** on the left hand side.
 - 2) Right click on the BINDATAImage form created using the wizard and choose **Edit**.
 - 3) The Form Controls toolbar should already be visible. You can check this using **View > Toolbars**; there should be a check mark next to *Form Controls*.
 - 4) Click on the more controls icon () to open the more controls toolbar.

- 5) Click on the Image control icon () and then draw out the control on the form. Be warned that the Image control icon is very similar to the Image Button icon.
 - 6) Right click on the newly inserted control and choose **Control** to open the control properties dialog.
 - 7) On the Data tab, set the *Data field* to DATA, which contains the binary data. If we happen to store non-image data, this is likely to be a problem for the image control, which now expects this data to be an image. Do not use this form if you store non-image data in the DATA field.
 - 8) Use **File > Save** to save the form into the Base document.
 - 9) Use **File > Close** to close the form.
- 4) The form has only been saved into the Base document, but now you must save the Base document. Use **File > Save** to save the Base document.

2.3.2. Using a macro

Stated simply, a form is a document that contains controls. In this case, the controls are connected to a database. The document's draw page contains shapes and forms. Each control is associated with a shape, the shape dictates where the shape is displayed. The macro in *Listing 8* creates a simple form, which is very similar to the form created using the form wizard; there are a few notable differences, however.

The macro in *Listing 8* creates one shape for each control. There is a separate shape for each data control and each label. The form wizard creates an additional GroupShape for each control/label pair. The group shape is used to keep a data control with its label. When the form is in design mode and you select a control, you are selecting the shape. If a control is grouped with its label, you select the two controls together because you are selecting the group shape rather than the individual controls. The disadvantage is that you can not easily select a specific control or its label to edit the individual properties—use the form navigator to select each individual component, even when they are grouped with others.

The form wizard in OOo version 2.0 creates forms using the .sxw file extension used in OOo version 1.x. The macro in *Listing 8* creates a Writer document using the newer .odt file extension.

The macro in *Listing 8* creates a Writer document as a form and stores it in the same directory as the Base document. After the form is created, it is added into the Base document. Although I have not tested this, there is no particular reason that an existing document can not be added into a Base document using the code shown in this macro. If you choose to do this, please report your results. ??

Another thing to consider is that a form does not really imply that a Writer document is used. My guess is that you should be able to create and store a Calc document into a Base document. If you try this, let me know how it works. ??

Create A Form

Listing 8: Create and add a form to a Base document.

```

Sub AddBinForm(sDBURL$, sTableName$)
    Dim oDoc          'Newly created Form document
    Dim oDrawPage     'Draw page for the form document.
    Dim s$            'Generic temporary string variable.
    Dim oDBDoc        'The Base database document.
    Dim sDBName$      'Name portion from sDBURL.
    Dim sFormURL$     'URL where the temporary form is stored.
    Dim oFormDocs     'Form documents in the Base document.
    Dim sFormName$    'Form name as stored in the Baes form documents.
    Dim oDocDef       'Document defition of the form stored in Base.
    Dim oDBForm

    Dim NoArgs() As new com.sun.star.beans.PropertyValue
    Dim oProps(2) as new com.sun.star.beans.PropertyValue

    REM Create a new document for the form.
    s$ = "private:factory/swriter"
    oDoc = StarDesktop.LoadComponentFromURL(s$, "_default", 0, NoArgs())

    REM The form will in edit mode, rather than design mode, by default.
    oDoc.ApplyFormDesignMode = False

    Dim oViewSettings
    oViewSettings = oDoc.CurrentController.ViewSettings
    oViewSettings.ShowTableBoundaries = False
    oViewSettings.ShowOnlineLayout = True

    REM Get the document's draw page and force the top level form to
    REM exist and be named "Standard".
    oDrawPage = oDoc.DrawPage
    If oDrawPage.Forms.Count = 0 Then
        s$ = "com.sun.star.form.component.Form"
        oDBForm = oDoc.CreateInstance(s$)
        oDrawPage.Forms.InsertByIndex (0, oDBForm)
    Else
        oDBForm = oDrawPage.Forms.GetByIndex(0)
    End If
    oDBForm.Name = "Standard"

    REM Cause the form to use the table as a datasource.
    oDBForm.DataSourceName = sDBURL
    oDBForm.Command = sTableName
    oDBForm.CommandType = com.sun.star.sdb.CommandType.TABLE

    REM Service names for controls.
    Dim sLabel$ : sLabel = "com.sun.star.form.component.FixedText"
    Dim oControl 'A control to insert into the form.
    Dim oShape   'Control's shape in the draw page.
    Dim oLControl 'Label control.
    Dim oLShape   'Label control's shape in the draw page.

```

```

REM Anchor the controls to paragraphs.
Dim lAnchor As Long
lAnchor = com.sun.star.text.TextContentAnchorType.AT_PARAGRAPH

REM Insert the ID label
oLControl = oDoc.CreateInstance(sLabel$)
oLControl.Label = "ID"
oLControl.Name = "lblID"

oLShape = oDoc.CreateInstance("com.sun.star.drawing.ControlShape")
oLShape.Size = createSize(1222, 443)
oLShape.Position = createPoint(1000, 1104)
oLShape.AnchorType = lAnchor
oLShape.control = oLControl
REM Do not add the label control yet!

REM Insert the ID formatted text field
s$ = "com.sun.star.form.component.FormattedField"
oControl = oDoc.CreateInstance(s$)
oControl.LabelControl = oLControl
oControl.BackgroundColor = 14540253
oControl.Border = 1
oControl.DataField = "ID"
oControl.EffectiveMax = 2147483647
oControl.EffectiveMin = -2147483648
oControl.EnforceFormat = True
oControl.HideInactiveSelection = True
oControl.Name = "fmtID"
oControl.TreatAsNumber = True

oShape = oDoc.CreateInstance("com.sun.star.drawing.ControlShape")
oShape.Size = createSize(2150, 651)
oShape.Position = createPoint(2522, 1000)
oShape.AnchorType = lAnchor
oShape.control = oControl
oDrawpage.Add(oLShape)
oDrawpage.Add(oShape)

REM Insert the Name label
oLControl = oDoc.CreateInstance(sLabel$)
oLControl.Label = "NAME"
oLControl.Name = "lblName"

oLShape = oDoc.CreateInstance("com.sun.star.drawing.ControlShape")
oLShape.Size = createSize(1222, 443)
oLShape.Position = createPoint(1000, 1954)
oLShape.AnchorType = lAnchor
oLShape.control = oLControl

REM Insert the Name text field
s$ = "com.sun.star.form.component.TextField"

```

```

oControl = oDoc.CreateInstance(s$)
oControl.BackgroundColor = 14540253
oControl.Border = 1
oControl.DataField = "NAME"
oControl.LabelControl = oLControl
oControl.Name = "txtNAME"

oShape = oDoc.CreateInstance("com.sun.star.drawing.ControlShape")
oShape.Size = createSize(8026, 651)
oShape.Position = createPoint(2522, 1850)
oShape.AnchorType = lAnchor
oShape.control = oControl
oDrawpage.Add(oLShape)
oDrawpage.Add(oShape)

REM Add the Image control
s$ = "com.sun.star.form.component.DatabaseImageControl"
oControl = oDoc.CreateInstance(s$)
oControl.BackgroundColor = 14540253
oControl.Border = 1
oControl.DataField = "DATA"
oControl.Name = "imgDATA"

oShape = oDoc.CreateInstance("com.sun.star.drawing.ControlShape")
oShape.Size = createSize(10504, 7835)
oShape.Position = createPoint(2522, 3332)
oShape.AnchorType = lAnchor
oShape.control = oControl
oDrawpage.Add(oShape)

REM At this point, we have a Form, which is a Writer document.
REM Store the stand alone form to disk. This form is usable as is.

REM Use some methods from the Tools library.
If NOT GlobalScope.BasicLibraries.isLibraryLoaded("Tools") Then
    GlobalScope.BasicLibraries.LoadLibrary("Tools")
End If

sDBName = GetFileNameWithoutExtension(sDBURL, "/")
sFormName = "Form_" & sTableName
s$ = DirectoryNameoutofPath(sDBURL, "/") & "/"
sFormURL = s$ & "Form_" & sDBName & "_" & sTableName & ".odt"

REM Store the form to disk and then close the document.
oDoc.StoreAsUrl(sFormURL, NoArgs())
oDoc.close(True)

REM Now, convert the form on disk to a document definition and
REM store it in a Base document.
oDBDoc = FindComponentWithURL(sDBURL$, True)
oFormDocs = oDBDoc.getFormDocuments()
If oFormDocs.hasByName(sFormName) Then

```

```

Print "Removing " & sFormName & " from the database"
oFormDocs.removeByName(sFormName)
End If

oProps(0).Name = "Name"
oProps(0).Value = sFormName
oProps(1).Name = "Parent"
oProps(1).Value = oFormDocs()
oProps(2).Name = "URL"
oProps(2).Value = sFormUrl
s$ = "com.sun.star.sdb.DocumentDefinition"
oDocDef = oFormDocs.createInstanceWithArguments(s$, oProps())
oFormDocs.insertByName(sFormName, oDocDef)
Print "Added " & sFormName & " to the database"
End Sub

```

2.4. Open a form using a macro

The macro in *Listing 9* performs the following operations:

- 1) Open a database document.
- 2) Allow the user to select a form.
- 3) Open the form using the macro `OpenFormInDB1()`.

Listing 9: Choose and open a form from a database.

```

Sub ChooseAndOpenFormInDB(sDBURL$)
    Dim oDoc
    Dim oForms
    Dim sFormName$
    Dim s$

    REM Find the database document and open it if required.
    oDoc = FindComponentWithURL(sDBURL$, True)
    If IsNULL(oDoc) OR IsEmpty(oDoc) Then
        Print "The document was not found"
        Exit Sub
    End If

    REM Choose a form to open!
    oForms = oDoc.getFormDocuments()
    If oForms.getCount() < 1 Then
        Print "The database contains no forms"
    ElseIf oForms.getCount() = 1 Then
        REM If there is ONLY one form, then open the one form!
        Dim x()
        x() = oForms.getElementNames()
        sFormName = x(0)
    Else
        s$ = "Choose A Form To Open"
    End If
End Sub

```



```

        sFormName = DialogSelectItem(oForms.getElementNames(), s$)
    End If

    If sFormName = "" Then Exit Sub
    OpenFormInDB1(sDBURL$, sFormName$)
End Sub

```

The form can now be loaded as shown in *Listing 10*. The form can be loaded in design mode without an active connection, but it is required for all other modes.

Choose And Open Form You will not be asked to choose a form if the database only contains one form.

Listing 10: Load a form from a database using LoadComponentFromURL.

```

Function OpenFormInDB1(sDBURL$, sFormName$)
    Dim oDBDoc          'The database document that contains the form.
    Dim oFormDef         'com.sun.star.sdb.DocumentDefinition of the form.
    Dim oFormDocs        'The form documents container.
    Dim oFormDoc         'The actual form document.
    Dim oCon             'Database connection.
    Dim oParms() As New com.sun.star.beans.PropertyValue
    Dim oBaseContext     'Global database context service.
    Dim oDataBase        'Database obtained from the database context.

    REM Find the database document and open it if required.
    oDBDoc = FindComponentWithURL(sDBURL$, True)
    If IsNULL(oDBDoc) OR IsEmpty(oDBDoc) Then
        Print "The document was not found"
        Exit Function
    End If

    oFormDocs = oDBDoc.getFormDocuments()
    If NOT oFormDocs.hasByName(sFormName) Then
        Print "The database does not have a form named " & sFormName
        Exit Function
    End If

    oFormDef = oDBDoc.getFormDocuments().getByName(sFormName)

    REM Without this, the form opens and then disappears!
    REM This is a bug that will hopefully be fixed in OOo version 2.0.1.
    REM oDummyFormDef is defined in the main module.
    oDummyFormDef = oFormDef

    oBaseContext = CreateUnoService("com.sun.star.sdb.DatabaseContext")
    oDataBase = oBaseContext.getByName(sDBURL)
    oCon = oDataBase.getConnection("", "")
    REM OpenMode is rumored to support "open", "openDesign",
    REM and "openForMail"
    AppendProperty(oParms(), "OpenMode", "open")
    AppendProperty(oParms(), "ActiveConnection", oCon)
    oFormDoc = oFormDocs.loadComponentFromURL(sFormName, "", 0, oParms())
    OpenFormInDB1() = oFormDoc

```

```

REM If you close the connection, then the form loses its connection.
REM The requirement of an Active connection should be removed,
REM hopefully in version 2.0.1.
REM This really looks like a resource leak, but I have not checked.
REM oCon.close()
End Function

```

TIP The macro in *Listing 10* obtains a reference to the database document using the method `FindComponentWithURL()`. The database document is available from the database context using the `DatabaseDocument` property.

```

oDataBase = oBaseContext.getByName(sDBURL)
oFormDoc = oDataBase.DatabaseDocument

```

In OOo version 2.0.1, you should be able to load a form without loading the document. In OOo version 2.0, this causes a crash.

Internally, `loadComponentFromURL()` performs an `execute` on the form definition object. The macro in *Listing 11*, demonstrates how to use the `execute` method. A connection is not required to open the form in design mode, but it is for all other modes (see *Listing 12*).

Listing 11: Load a form by executing the form definition.

```

Function OpenFormInDB2(sDBURL$, sFormName$)
    Dim oDBDoc          'The database document that contains the form.
    Dim oFormDef         'com.sun.star.sdb.DocumentDefinition of the form.
    Dim oFormDocs        'The form documents container.
    Dim oFormDoc         'The actual form document.
    Dim oBaseContext     'Global database context service.
    Dim oDataBase        'Database obtained from the database context.
    Dim oCon             'Database connection.
    Dim oParms() As New com.sun.star.beans.PropertyValue

    REM Find the database document and open it if required.
    oDBDoc = FindComponentWithURL(sDBURL$, True)
    If IsNULL(oDBDoc) OR IsEmpty(oDBDoc) Then
        Print "The document was not found"
        Exit Function
    End If

    oFormDocs = oDBDoc.getFormDocuments()
    If NOT oFormDocs.hasByName(sFormName) Then
        Print "The database does not have a form named " & sFormName
        Exit Function
    End If

    oFormDef = oDBDoc.getFormDocuments().getByName(sFormName)
    oBaseContext = CreateUnoService("com.sun.star.sdb.DatabaseContext")
    oDataBase = oBaseContext.getByName(sDBURL)
    'oCon = oDataBase.getConnection("", "")
    AppendProperty(oParms(), "ActiveConnection", oCon)

    Dim identifier as Long
    identifier = oFormDef.createCommandIdentifier()

```

```

Dim UcbCommand as new com.sun.star.ucb.Command
UcbCommand.Name = "openDesign" 'Or "open" or "openForMail"
Dim Arguments as new com.sun.star.ucb.OpenCommandArgument2
Arguments.Mode = com.sun.star.ucb.OpenMode.DOCUMENT
UcbCommand.Argument = Arguments

Dim environment as Object
oFormDoc = oFormDef.execute( UcbCommand, identifier, environment )
OpenFormInDB2() = oFormDoc
End Function

```

The macro in *Listing 12* demonstrates how to open a form with the current connection. The `onClickOpenForm` method would be called from a form in the same Base document.

Listing 12: Load a form by executing the form definition.

```

Sub onClickOpenForm ( oEvent as variant )
    OpenForm(oEvent, "Form Name")
End sub

Sub OpenForm( oEvent as variant, aFormName as string) as variant
    Dim args(1) As New com.sun.star.beans.PropertyValue
    Dim container as variant
    Dim oCon

    oCon = oEvent.Source.Model.Parent.ActiveConnection
    container = oCon.Parent.DatabaseDocument.FormDocuments

    args(0).Name = "ActiveConnection"
    args(0).Value = oCon
    args(1).Name = "OpenMode"
    args(1).Value = "open"
    container.loadComponentFromURL(aFormName,"_blank",0,args())
End Sub

```

According to FS, If you want to open a form (in the sense of: display UI) when the document loads, you should use methods from the controller, not from the document. So, something like:

```

oController = ThisDatabaseDocument.CurrentController
If ( Not oController.isConnected() ) Then
    oController.connect()
End If
oController.loadComponent(com.sun.star.sdb.application.DatabaseObject.FORM, "my
form",FALSE )

```

2.5. Accessing the binary data

2.5.1. Adding binary data

Adding binary data using a macro is pretty easy. Unfortunately, it is not possible with Ooo version 2.0 to use an output stream directly, you must first read the data into an array of bytes. ?? check this in version 2.04.

Listing 13: Add binary data to a table.

```

Sub InsertImage(sDBURL$, sFileURL$)
    Dim sFileName$      'File to save in the database.
    Dim oData()         'Array of bytes.
    Dim lLen As Long    'Number of bytes in the file.
    Dim oDB              'Database object.
    Dim oStream
    Dim oSimpleFileAccess
    Dim oBaseContext
    Dim oStatement
    Dim sSQL$
    Dim oCon
    Dim s$

    If NOT FileExists(sFileURL) Then
        Print "Sorry, " & sFileURL & " does not exist"
        Exit Sub
    End If

    If NOT FileExists(sDBURL) Then
        CreateBinaryDB(sDBURL, True)
    End If
    CreateBinaryTables(sDBURL, False, False)

    REM Load the Tools library
    If NOT GlobalScope.BasicLibraries.isLibraryLoaded("Tools") Then
        GlobalScope.BasicLibraries.LoadLibrary("Tools")
    End If

    REM Call methods in the Tools library to parse the path.
    sFileName = FileNameOutOfPath(sFileURL, "/")

    oBaseContext = CreateUnoService("com.sun.star.sdb.DatabaseContext")
    oDB = oBaseContext.getByName(sDBURL)
    oCon = oDB.getConnection("", "")

    s$ = "com.sun.star.ucb.SimpleFileAccess"
    oSimpleFileAccess = createUnoService(s$)
    oStream = oSimpleFileAccess.openFileRead(sFileURL)

    REM Get the total length and then dimension the array.
    lLen = oStream.getLength()
    ReDim oData(0 To lLen-1)
    oStream.readBytes(oData(), lLen)

    REM Use a prepared statement to insert the data.
    REM Notice that I do not set the ID because it is
    REM an auto-value field.
    sSQL = "insert into BINDATA (NAME, DATA) values (?, ?)"
    oStatement = oCon.PrepareStatement(sSQL)
    oStatement.SetString( 1, sFileName)

```

```

REM I should be able to simply use the stream
REM but there is a bug that prevents this. Too bad!
'oStatement.setBinaryStream(2, oStream, oStream.getLength())
oStatement.setBytes(2, oData(), lLen)
oStatement.ExecuteUpdate()
oStream.closeInput()
Print "Inserted " & sFileName
oCon.close()
Exit Sub
End Sub

```

Add Binary File To DB The following example, selects a specific file and inserts the file into a database. Although the macro name implies that an image is inserted, any file can be inserted. You can add the same file many times, the macro neither knows, nor cares. It is likely to be a problem when you try to extract data, however.

Listing 14: Select a file and add it to the database.

```

Sub CallInsertImage()
    Dim sFileURL$
    Dim sDBURL$

    LoadDBLibs()
    sFileURL = ChooseAFile(GraphicFilters(), True)
    If sFileURL = "" Then
        Exit Sub
    End If
    sDBURL = GetSourceCodeDir() & sDBBaseName
    InsertImage(sDBURL$, sFileURL$)
End Sub

```

2.5.2. Extracting binary data

Extracting a binary file is easier than adding a binary file. The difficult part is determining which file to extract. The following steps are performed in this example:

- 1) Connect to the database.
- 2) The database has a field/column, which contains a list of file names. When a graphic is added to the database, the name field is set to contain the file name of the graphic. A result set is generated, which contains a sorted list of graphic file names from the database.
- 3) Use the macro in *Listing 83* to select a file name from the result set.
- 4) If the file exists, the `chooseAFile()` macro from *Listing 58* allows the user to select a different file.
- 5) If the file does not exist, then it is written into the same directory containing this document.

Extract A File For obvious reasons, you should add data to the database before you try to extract it.

Listing 15: Extract a binary file from the database.

```

Sub ExtractBinaryFile(sPathURL$, sDBURL$)
    Dim sFileName$      'File to save from the database.
    Dim oDB              'Database object.
    Dim oStream
    Dim oSimpleFileAccess
    Dim oBaseContext
    Dim oStatement
    Dim oResult
    Dim sSQL$
    Dim oCon
    Dim sURL$
    Dim s$

    If NOT FileExists(sDBURL) Then
        Print "The DB does not exist, sorry"
        Exit Sub
    End If

    oBaseContext = CreateUnoService("com.sun.star.sdb.DatabaseContext")
    oDB = oBaseContext.getByName(sDBURL)
    oCon = oDB.getConnection("", "")
    oStatement = oCon.createStatement()
    sSQL = "SELECT NAME FROM BINDATA ORDER BY NAME"
    oResult = oStatement.executeQuery(sSQL)
    sFileName = SelItemFromResult(oResult, 1, 100)
    If sFileName = "" Then
        oCon.close()
        Exit Sub
    End If

    sSQL = "SELECT DATA FROM BINDATA WHERE NAME='" & sFileName & "'"
    oResult = oStatement.executeQuery(sSQL)
    If Not IsNull(oResult) Then
        oResult.next()
        REM I could get a byte array, but this is easier.
        oStream = oResult.getBinaryStream(1)
        If oResult.isNull() Then
            Print "The image was NULL"
        Else
            s = "com.sun.star.ucb.SimpleFileAccess"
            oSimpleFileAccess = createUnoService(s)
            sURL = sPathURL & sFileName
            If FileExists(sURL) Then
                sURL = ChooseAFile$(GraphicFilters(), False, sURL)
            End If
            If sURL <> "" Then
                oSimpleFileAccess.writeFile(sURL, oStream)
                Print "Wrote " & sURL
            End If
        End If
    End If
End Sub

```

```
End If  
oCon.close()  
End Sub
```

?? Open a document directly into OOo!

To open a document directly in OOo, I must obtain an appropriate file stream. Unfortunately, the file stream returned from my query is not sufficient – it does not support all of the required interfaces. I can accomplish this in Java by creating my own class that supports all of the requisite interfaces, but I am using Basic in this document.

3. One-To-Many relationships

I decided to write this chapter because I have difficulties figuring out the best way to represent associated data in forms. By associated data, I mean a one to many, or many to many relationship. Every attempt is made to spell things out in detail, so that even a first timer can at least follow along. Things may become a bit more difficult when macros are introduced.

For the one-to-many relationship, consider a simplified inventory that associates items to a dealer. The assumption is that each item comes from one, and only one dealer.

You need a base document that will contain the data. Use the following step by step instructions to create a sample database for use.

- 1) Use **File > New Database** to open the new database wizard.
- 2) Select the *Create a new database* radio button and click **Next**.
- 3) Select the *No, do not register the database* radio button, the *Open the database for editing* checkbox, and click **Finish**.
- 4) Name the database OooBaseAssociateData.odt and click **Save**.

3.1. Create the tables

The first table contains the dealer information (see *Table 3*). The database is intentionally very simple. A “useful” table is likely to contain more information. The field names use uppercase characters and contain no spaces.

Table 3. Fields in the DEALER table.

Field	Field Type	Comment
ID	Integer [INTEGER]	Table's primary key
NAME	Text [VARCHAR]	Dealer name.

Maintaining “simple” theme, the item table is also very simple (see *Table 4*). The DEALER field links back to the ID field in the DEALER table.

Table 4. Fields in the ITEM table.

Field	Field Type	Comment
ID	Integer [INTEGER]	Table's primary key
ITEM	Text [VARCHAR]	Item name.
DEALER	Integer [INTEGER]	Dealer ID.

3.1.1. Create the DEALER table

Select **Tables** from the left hand side and then choose the **Create Table in Design View** task. Enter the fields in the table design window.

- 1) Create the primary key.
 - 1) Set the *Field Name* to ID.
 - 2) Set the *Field Type* to Integer.
 - 3) Set *Auto Value* to Yes.
 - 4) Right click to the left of the field name and choose **Primary Key**.
- 2) Create the name field.
 - 1) Set the *Field Name* to NAME.
 - 2) Set the *Field Type* to Text [VARCHAR].
 - 3) Set *Entry required* to Yes.
 - 4) Keep the default length of 50.

Use **File > Save** to save the table. Name the table DEALER, and then use **File > Close** to close the table design window.

3.1.2. Create the ITEM table

Although the ITEM table is easily created using the same method used to create the DEALER table, it is always instructive to learn a new method; creating the ITEM table from the DEALER table. Select **Tables** from the left hand side to view the existing tables.

- 1) Right click on the DEALER table and choose copy.
- 2) Right click near the DEALER table, but not on the table and choose paste.
 - 1) Change the *Table name* from DEALER to ITEM
 - 2) There is no data in the DEALER table yet, but we only need to copy the Definition.
 - 3) Do not check *Create primary key*, this will create a new primary key that we do not want.
 - 4) Click **Next** to open the Apply columns dialog.
- 3) Use the Apply columns dialog to copy all of the fields from the left hand side to the right hand side. This causes all of the fields to be added to the ITEM table. Click **Next** to open the Type formatting dialog.

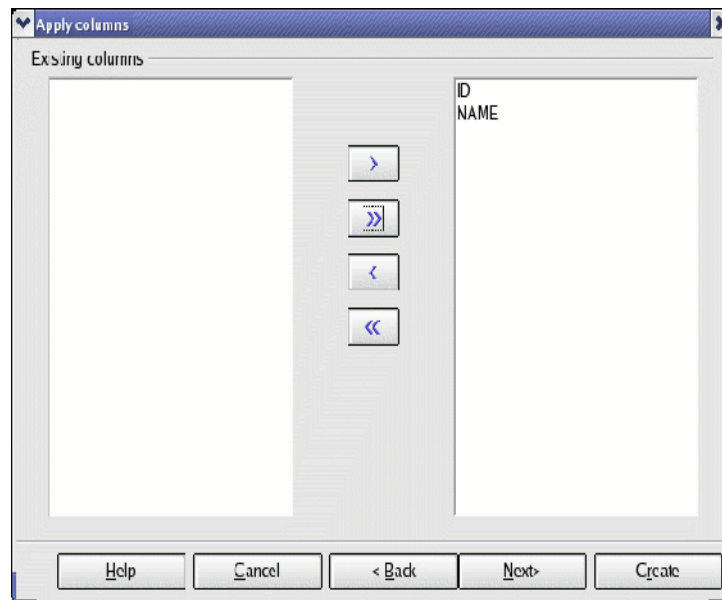


Figure 1: Use the arrows to copy the fields.

- 4) Use the Type formatting dialog to set the field types. The existing field types are fine as is, including using ID as the primary key. Click **Create** to create the ITEM table.

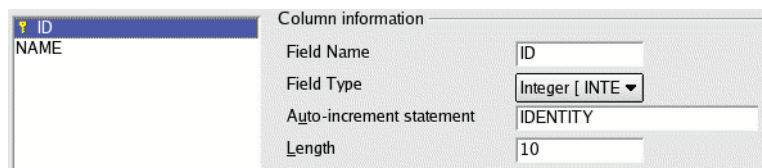


Figure 2: Set the field types.

- 5) Add the DEALER field to the ITEM table.
 - 1) Right click on the ITEM table and choose *Edit*.
 - 2) Set the *Field Name* to DEALER.
 - 3) Set the *Field Type* to Integer.
 - 4) Set *Entry required* to Yes.
 - 5) Leave *Auto value* as No.
 - 6) Set the *Default value* to 0 (zero, not a letter). ?? In OOo version 2.0, this seems to corrupt the database after it has been saved and reloaded.
 - 7) Use **File > Save** to save the table modifications to the Base document.
 - 8) Use **File > Close** to close the table design window.
- 6) Use **File > Save** for the Base document, to save the table modifications to disk.

3.2. Define the data relationships

As already stated, it is assumed that each item comes from one, and only one dealer. Use **Tools > Relationships** to open the Relationship design window. When the relationship design window opens, the Add Tables dialog also opens (see *Figure 3*). Add both the DEALER and the ITEM table to the Relationship design window by selecting each table and clicking the **Add** button. When you are finished adding tables, click the **Close** button to close the Add tables dialog.

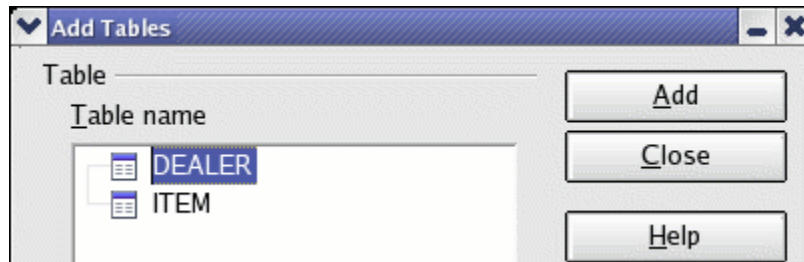


Figure 3: Add tables to the relationship design window.

To create a one-to-many relationship from, click on the ID field in the DEALER table and drag it to the DEALER field in the ITEM table. A line is drawn between the tables that illustrates the connection (see *Figure 4*).

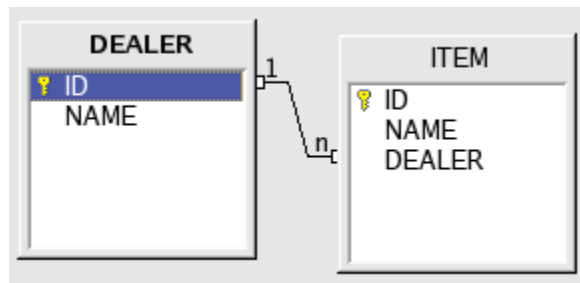


Figure 4: Create a one-to-many relationship.

The established relationship indicates that each record in the ITEM table corresponds to a single record in the dealer table. On the other hand, each record in the DEALER table corresponds to zero or more records in the ITEM table. Assume that item 7 refers to dealer 3, and you try to delete dealer 3; what happens? Double click on the relationship line to open the Relationship dialog.

The Relationship dialog allows you to specify what to do if the dealer ID is updated (changed) or deleted. The default behavior is to do nothing. If I choose to use an *Update cascade*, then if I change dealer 3 to have a new identifier, then all records in the Item table are updated to use the new value. If *Delete cascade* is set, then deleting dealer 3 causes every item that refers to dealer 3 to be deleted. Set the update action to *Update cascade* and the delete action to *Set default* (see *Figure 5*)—remember that the default value for the DEALER field in the ITEM table is 0. Use **File > Save** to save the relationships to the Base document and **File > Close** to close the relationship window.

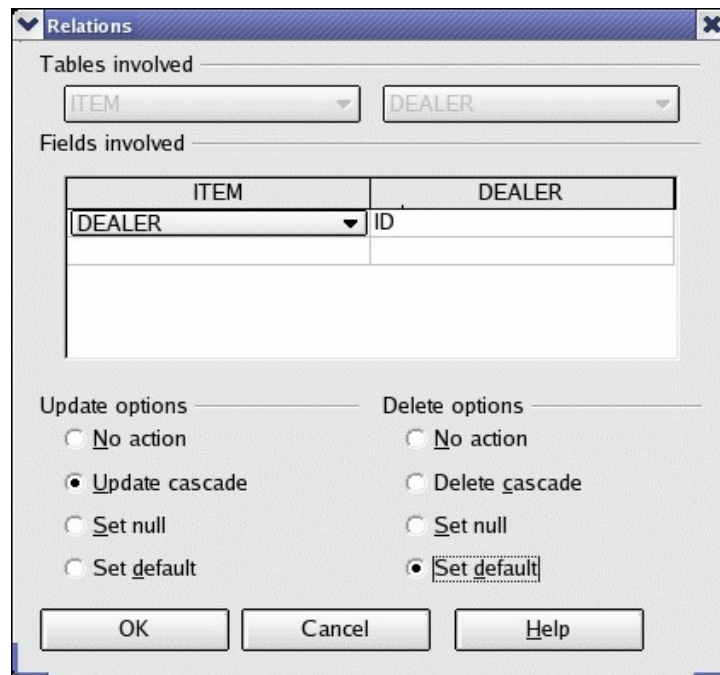


Figure 5: Set the update and delete actions.

3.3. Add data to the DEALER and ITEM tables

To provide examples, it is important that the tables contain data. Enter the sample data from Table 5 into the DEALER table.

Table 5: Sample data in the DEALER table.

ID	NAME
0	Unknown
1	Dealer 1
2	Dealer 2
3	Dealer 3
4	Dealer 4
5	Dealer 5

To add data to the DEALER, you must open the table for editing. Click on the Tables icon in the left hand column and then double click on the DEALER table (see Figure 6).



Figure 6: Open the DEALER table for editing.

The ID field is an AutoValue field, and it is shown as such in the table (see Figure 7). The table is currently empty, and a new empty record is ready to be added. You can not place the cursor in the ID field and enter a value because this is an AutoValue field. Place the cursor in the NAME field and enter the value “Unknown” and press the *Enter* key. The ID field is automatically given the value of zero. As you enter new values, the ID field will increment by one each time. While entering data using the table view, can not enter a value for the ID field. You can, however, change an ID after entering the record.

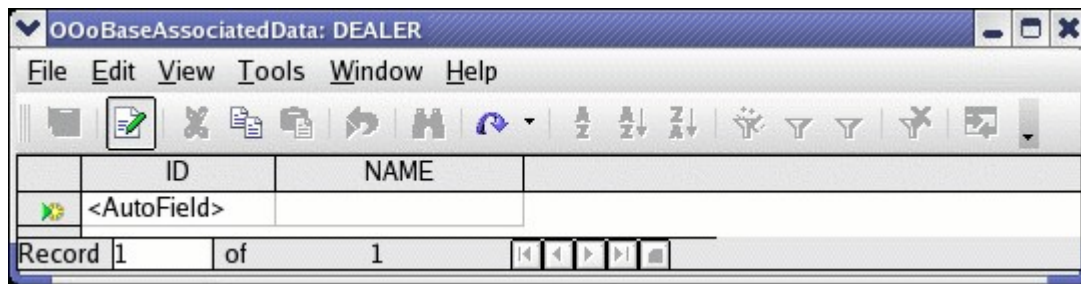


Figure 7: Empty DEALER table, waiting for data.

The internal database remembers the value used in the ID field, which is then used to provide the next automatic ID value. The important thing to remember, is that the next value is one larger than the largest value used so far, not one greater than the largest value that is currently in the database. So, if you change an entry from 4 to 100 and then back to 4, the next automatic value will be 101. With SQL, you can set the value to any value that you desire.

TIP The described behavior assumes that the Base document references an internal HSQLDB database—this is the default.

Now add example data to the ITEM table (see Table 6).

Table 6: Sample data in the ITEM table.

ID	NAME	Dealer
0	Unknown	0

ID	NAME	Dealer
1	Item 1	3
2	Item 2	1
3	Item 3	1
4	Item 4	3
5	Item 5	2

4. Forms

At a high level, a form is a Writer document containing controls attached to a database. A control can be a button, a list box, a text input box, or one of many other control types. When I speak of a form at a high level, I refer to the entire Writer document. This is what you load when you open a form.

To a form designer, the high level “form”, which is a Writer document, contains things called forms. Each form is associated with a specific table in a database. Each form can contain controls and other forms. When a form is contained in another form, it is called a sub-form.

4.1. The internal object model

Macro programmers need to understand the internal object model, which can be rather confusing—which is a shame because it is not as confusing as many people seem to think. If you are not a macro programmer, you might want to skip this section. Unfortunately, much of the interesting stuff requires the use of macros.

4.1.1. A control's shape is in the draw page

A Writer document contains a draw page. All shapes are contained in the draw page. There is a special type of shape called a control shape, which corresponds to a single control. To change the size or location of a control, find the corresponding control shape and modify it. The Form wizard groups each control with its corresponding label in a group shape object. It is, therefore, important that any code that searches for control shapes looks inside group shapes.

Listing 16: Finding control shapes in a Writer document.

```
Dim oDrawPage
Dim oShape
Dim i%
Dim sGroupShape
Dim sControlShape

sGroupShape = "com.sun.star.drawing.GroupShape"
sControlShape = "com.sun.star.drawing.ControlShape"

oDrawPage = ThisComponent.getDrawPage()
For i = 0 To oDrawPage.getCount() - 1
    oShape = oDrawPage.getByIndex(i)
    If oShape.supportsService(sGroupShape) Then
        REM The group shape supports the methods getCount()
        REM and getByIndex() to obtain the contained shapes.
        REM You should probably use a recursive routine to
        REM extract the shapes, because the contained shape
        REM may be another group shape.
        REM Dim o
        REM For j = 0 To oShape.getCount() - 1
        REM     o = oShape.getByIndex(j)
        REM     Print o.control.name
        REM Next
```

```

ElseIf oShape.supportsService(sControlShape) Then
    REM Access the control model using oShape.Control
    REM Print oShape.control.name
End If
Next

```

4.1.2. A draw page contains forms

A draw page contains forms, and forms contain controls and other forms. It is easy, therefore, to find a named form or control.

Listing 17: Find a named form or control.

```

REM oObj can be a generic draw page, a form components object, a
REM form, or a Writer document. This routine will find any
REM form component, which means a form or a control.
Function findForm(oObj, sName$)
    Dim sForm$      : sForm = "com.sun.star.form.component.Form"
    Dim sForms$     : sForms = "com.sun.star.form.FormComponents"
    Dim sComp$      : sComp = "com.sun.star.form.FormComponent"
    Dim oForm
    Dim i%
    Dim x

    REM Extract the forms from a generic draw page.
    If oObj.supportsService("com.sun.star.drawing.GenericDrawPage") Then
        findForm() = findForm(oObj.getForms(), sName$)
        Exit Function
    End If

    REM If the object is an office document, assume that it is a Writer
    REM document. If this is a Calc document, then the object has more
    REM than one draw page. I am too lazy to worry about this now.
    If oObj.supportsService("com.sun.star.document.OfficeDocument") Then
        findForm() = findForm(oObj.getDrawPage().getForms(), sName$)
        Exit Function
    End If

    REM If this is a form component, then it has a name.
    REM Check to see if the name is the search name.
    If oObj.supportsService(sComp) Then
        If oObj.getName() = sName Then
            findForm() = oObj
            Exit Function
        End If
    End If

    REM If this object contains components, then search it.
    If oObj.supportsService(sForms) Then
        REM Enumerate forms and controls.
        For i = 0 To oObj.getCount()-1
            x = oObj.getByIndex(i)
            oForm = findForm(x, sName)

```



```

        If NOT IsNull(oForm) AND NOT IsEmpty(oForm) Then
            findForm() = oForm
            Exit For
        End If
    Next
Else
    End If
End Function

```

4.1.3. A control's data model is in a form

Every control contains data such as the text that is displayed. The object that encapsulates this information is called the model. The control's model is contained in a form. Use the control's model to enable or disable the control.

The complete form/control hierarchical structure is as follows: A Writer document contains a draw page. The draw page contains shapes and forms. Each form can contain control models and other forms. Rather than provide a lengthy explanation, I provide an example macro.

Listing 18: Inspect control models in a form.

```

REM oObj can be a generic draw page, a form components object, a
REM form, or a Writer document. The lead name usually starts as "".
Function getControlNames(oObj, ByVal sLeadName$) As String
    Dim sForm$      : sForm      = "com.sun.star.form.component.Form"
    Dim sForms$     : sForms     = "com.sun.star.form.FormComponents"
    Dim sControl$   : sControl   = "com.sun.star.form.FormControlModel"
    Dim s$
    Dim i%
    Dim x

    REM Extract the forms from a generic draw page.
    If oObj.supportsService("com.sun.star.drawing.GenericDrawPage") Then
        getControlNames() = getControlNames(oObj.getForms(), sLeadName)
        Exit Function
    End If

    REM If the object is an office document, assume that it is a Writer
    REM document. If this is a Calc document, then the object has more
    REM than one draw page. I am too lazy to worry about this now.
    If oObj.supportsService("com.sun.star.document.OfficeDocument") Then
        Dim oForms : oForms = oObj.getDrawPage().getForms()
        s = getControlNames(oForms, sLeadName)
        getControlNames() = s
        Exit Function
    End If

    REM Add the current form name to the lead name if, and only if,
    REM the argument is a form.
    If oObj.supportsService(sForm) Then
        sLeadName = sLeadName & oObj.getName()
    ElseIf NOT oObj.supportsService(sForms) Then
        getControlNames() = ""
    End If
Exit Function

```

```

End If

REM Enumerate forms and controls.
For i = 0 To oObj.getCount()-1
    x = oObj.getByIndex(i)
    If x.supportsService(sControl) Then
        s = s & sLeadName & " : " & x.getName() & CHR$(10)
    ElseIf x.supportsService(sForms) Then
        If sLeadName = "" Then
            s = s & getControlNames(x, sLeadName)
        Else
            s = s & getControlNames(x, sLeadName & ".")
        End If
    End If
Next
getControlNames() = s
End Function

```

4.1.4. A control's view model is in the controller

Every visible document has a current controller. The controller handles user interaction. To cause a control to become visible, or invisible, use the control's view model returned from the controller.

The controller has the method `getControl()`, which accepts a control model as an argument. In other words, you must get a control model before you can get the view model from the controller. When a control calls an event handler, it passes an event as an argument. The source property of the event contains the control's view model. The data model is obtained by using the `getModel()` method on the view model.

Listing 19: A very simple event handle; it does nothing.

```

Sub ButtonEventHandler(oEvent)
    oEvent.Source           'Control's view model.
    oEvent.Source.getModel() 'Control's data model.
End Sub

```

The current controller is obtained from the Writer document. If you have a form, but not the parent Writer document, you can work your way back to the containing parent document as follows:

Listing 20: Get the containing parent document from a form.

```

Function getDocumentFromForm(oForm)
    Dim x
    Dim sForm$ : sForm = "com.sun.star.form.FormComponent"
    Dim sDoc$   : sDoc  = "com.sun.star.document.OfficeDocument"

    If NOT oForm.supportsService(sForm) Then
        Exit Function
    End If
    x = oForm
    Do While NOT x.supportsService(sDoc)
        x = x.getParent()
    Loop
End Function

```

```

Loop
    getDocumentFromForm() = x
End Function

```

4.1.5. Enabling and setting controls visible – an example

As a final example to illustrate how to access controls in a form, consider a macro that enables, or disables, all controls in a specific form (see *Listing 21*). First, the document is obtained from the form objects using the macro in *Listing 20*. Each control data model is obtained from the form. The data model is enabled, or disabled, depending on the `bEnable` argument. A disabled control is still visible in a form, but it is shown in a subdued color. The data model is used to obtain the view model from the document controller. The view model is then set visible, or invisible.

Listing 21: Toggle all controls in a form visible and enabled.

```

REM oForm - Form on which to work.
REM bEnable - Enable, or disable, every control in the form.
REM bVisible - Set every control visible, or invisible.
REM bRecurse - If true, recurse into subforms.
Sub EnableControls(oForm, bEnable As Boolean, _
    bVisible As Boolean, _
    bRecurse As Boolean)

    Dim sForm$      : sForm      = "com.sun.star.form.component.Form"
    Dim sForms$     : sForms     = "com.sun.star.form.FormComponents"
    Dim sControl$   : sControl   = "com.sun.star.form.FormControlModel"
    Dim oController
    Dim oDoc
    Dim i%
    Dim x

    oDoc = getDocumentFromForm(oForm)
    oController = oDoc.getCurrentController()

    For i = 0 To oForm.getCount()-1
        x = oForm.getByIndex(i)
        If x.supportsService(sControl) Then
            x.Enabled = bEnable
            oController.getControl(x).Visible = bVisible
        ElseIf x.supportsService(sForms) Then
            If bRecurse Then EnableControls(x, bEnable)
        End If
    Next
End Sub

```

4.1.6. Finding a control from an event – an example

Consider a text table with a button in the first column of every row. Each button in the table calls the same macro. The macro wants to know which cell contained the button. The solution to the problem is as follows:

- 1) Obtain the view model from the event using `oEvent.Source`.
- 2) Obtain the data model from the view model using `oEvent.Source.getModel()`.

- 3) Enumerate the shapes on the draw page. This code is not really safe because every shape is assumed to be a control shape.
- 4) Obtain the data model from the control shape using `oShape.control`.
- 5) Use `EqualUNOObjects` to compare the two data models.

The macro in *Listing 22* implements a working solution.

Listing 22: Determine which cell contains a button control.

```
Sub ButtonCall(oEvent)
    Dim i
    Dim oButton
    Dim oModel
    Dim oShape
    Dim bFound As boolean

    REM First, get the button used to call this routine.
    REM Save the button's model.
    oButton = oEvent.Source
    oModel = oButton.getModel()

    REM Iterate through the controls
    i = ThisComponent.getDrawPage().getCount()
    bFound = False
    Do While (i > 0 AND NOT bFound)
        i = i - 1
        oShape = ThisComponent.getDrawPage().getByIndex(i)
        bFound = EqualUNOObjects(oShape.Control, oModel)
    Loop
    If bFound Then
        Print "The button is in cell " & oShape.getAnchor().Cell.CellName
    End If
End Sub
```

4.1.7. Control connected to a database

A control can be bound to a database field. Unfortunately, the value in the control is not always and automatically in sync with the value in the database field.

- 1) The user interacts with the visual component.
- 2) The control model describes the look and behavior of the control.
- 3) The form field holds the data from one specific column of the result set represented by the form; did I mention that a form acts like a result set?

Code that deals directly with the control model does not interact with the underlying result set, and therefore, does not deal with the values in the database. For example, you set new values in the form (using a macro), and those values are not saved to the database (result set). Use the `commit` method on the control to explicitly commit the control/model content to the database field.

Andrew Jensen specifically recommends obtaining the control from the controller and the modifying the control rather than the model; and what Mr. Jensen says is almost always correct. Although this works in some cases where modifying the control model does not, it is not considered safe. There are issues related to which listeners are called, and what will really be updated (according to Frank Schönheit).

The safest solution is to modify the bound field directly. For example, your current code may be as follows:

```
oControl.setString( "myText" )
```

It is better if you can use the bound field:

```
oControl.BoundField.updateString( "myText" )
```

The required method is dependent on the data type, which is a drawback to this method. I am aware of at least one instance where the specific method (updateDate) failed, yet the generate updateString method properly updated the bound field.

4.1.8. Control model summary

A lot more can be said about controls, forms and their structure. I leave this discussion until it is needed for an example.

4.2. Database Forms act like a result set

A form fulfills several tasks, like storing the structure of its form components, storing the information concerning tab ordering and control grouping, and providing the event environment for its contained elements. A database form adds the ability to connect to a database.

Table 7: Some services supported by database forms.

Service	Description
Form	Specifies a form which is a group of FormComponents. The form service allows a specific form component to be identified as a form rather than a control (see <i>Listing 18</i>).
FormComponent	Allows a form to be contained in another form.
FormComponents	Allows a form to contain multiple components.
DataForm	Specifies a form that is connected to a database, displaying the results of SQL queries. A database form can add, modify, and delete records. A database form is essentially an enhanced row set, which can display and manipulate the data.
ResultSet	A ResultSet, which is usually generated by executing a Statement, maintains a cursor pointing to its current row of data. Initially the cursor is positioned before the first row. "Get" routines are provided to retrieve column values for the current row.
RowSet	A RowSet is a client side ResultSet, which combines the characteristics of a Statement and a ResultSet. A RowSet can be used to implement capabilities for a result set, which are not supported by a driver result set; caching and update capabilities, for example. A RowSet provides event notification for various changes in state.

4.2.1. Duplicate record macro

I am not aware of a built-in method to duplicate a record while entering new data, so I decided to write a simple macro to do this. First, use the form wizard to create a form to edit the DEALER table. Click on the Forms icon (see *Figure 8*).

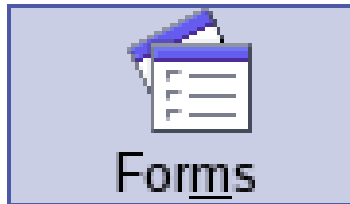


Figure 8: Forms icon

Next, click on Use Wizard to Create Form to start the Form Wizard. Select the DEALER table and then copy all Available fields to the Fields in the form and click **Next** twice.



Figure 9: Add all fields from the DEALER table to the form.

Choose a layout for the form and click **Next** three times.

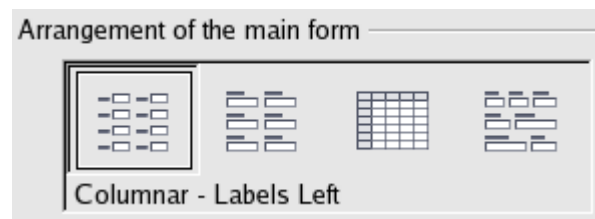


Figure 10: Use Columnar - Labels Left

Name the form DEALER_Add and click **Finish** to exit the Form Wizard. Use **File > Close** to close the newly created form. Right click on the newly created form and choose **Edit**. Use **View > Toolbars** to enable the **Form Controls** and **Form Design** toolbars. If a toolbar is displayed, there is a check mark next to the toolbar name.

Use **Tools > Macros > Organize Macros > OpenOffice.org Basic** to open the OpenOffice.org Macros dialog. Click on the **Organizer** button to open the Organizer dialog. Select the Modules tab, if it is not already selected. Find the DEALER_Add document in the Module list (see *Figure 11*). Select the Standard library, which already exists, and click the **New** button. Click **OK** to use the default new module name of Module1.

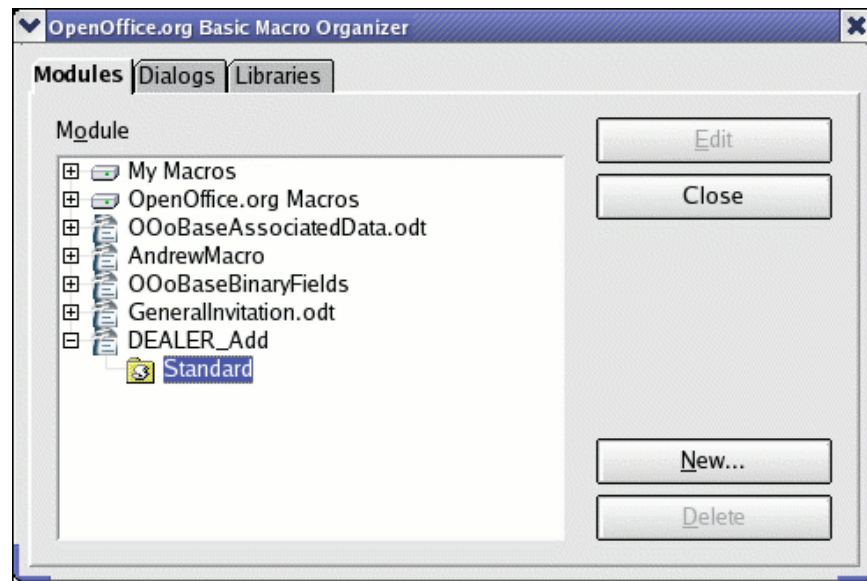


Figure 11: Add a new module to the DEALER_Add Form.

Finally, highlight Module1 and click the **Edit** button, which will open the Basic Integrated Debugging Environment (IDE). Replace the text with *Listing 23*; the included comments indicate how the macro works. Use **File > Save** to save the macro into the DEALER_Add form.

TIP

When you save a macro in a form, the macro is stored in the form document. When a form document is saved, it is saved in the Base document. Saving the form document, persists the form into the Base document, but it does not persist the Base document to disk. Be certain to save the Base document after you save a form to the Base document.

Listing 23: Macro to duplicate a DEALER record.

```
REM ***** BASIC *****
Option Explicit

Sub Main
End Sub

Sub DuplicateRecord(oEvent)
    Dim oForm      'Reference to the form containing the primary record.
    Dim lNameCol   'Column used to hold the NAME in the DEALER table.
    Dim sDealer$   'Value stored in the current NAME field.

    REM Get the button's view model from the event Source.
    REM Get the button's data model from the view model.
    REM Get the form from the data model.
    oForm = oEvent.Source.getModel().getParent()

    REM Find the column that contains the NAME field.
    lNameCol = oForm.findColumn("NAME")

    REM Not likely to happen, but check for it anyway.
```

```

If oForm.isAfterLast() Then
    Print "Hey, you are after the last element"
    Exit Sub
End If

REM Not likely to happen, but check for it anyway.
If oForm.isBeforeFirst() Then
    Print "Hey, you are before the first element"
    Exit Sub
End If


REM Get the data for the current record.
REM This is a simple table with an ID and a NAME.
REM The ID is an autovalue, so we only need to save the NAME.
sDealer = oForm.getString(lNameCol)

REM If new data has been entered into the form,
REM when I change records, I lose the data.
REM Avoid this by updating the row.
oForm.updateRow()

REM Now, move to the "Insert row" to add a new record.
oForm.moveToInsertRow()

REM Update all of the data from the original record that you stored.
REM If I only updated the control, then the form would not know
REM about it, but if I update the column in the row, then the control
REM is automatically updated.
oForm.updateString(lNameCol, sDealer)
End Sub

```

Using the form again, click on the command button icon () in the Controls toolbar and the mouse cursor will change to cross-hairs. Drag a rectangle where you want the button to be placed (see *Figure 12*).

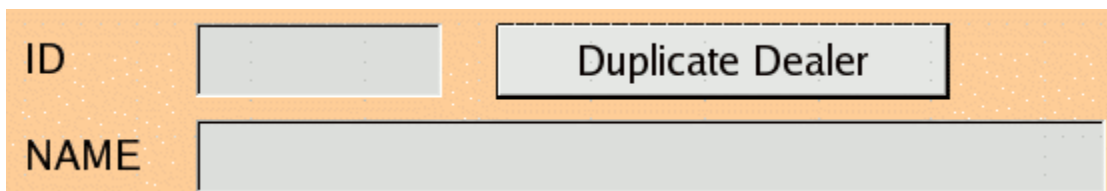



Figure 12: Completed DEALER_Add form.

Double click on the newly added button to open the button properties dialog. In the General tab, name the button PushButtonDuplicate, and set the label to Duplicate Dealer. Select the events tab. A control button supports many events. It is possible to differentiate between a mouse click, and a key press. All we care about is that the button has been clicked. Click on the three dots () next to the When initiating box to open the Assign Macro dialog. Although there is an icon next to every supported event, you can assign any event (or multiple events) to a macro. Highlight the When initiating event and click on the Assign macro. Find the DuplicateRecord macro in the DEALER_Add form, highlight it, and click **OK**.

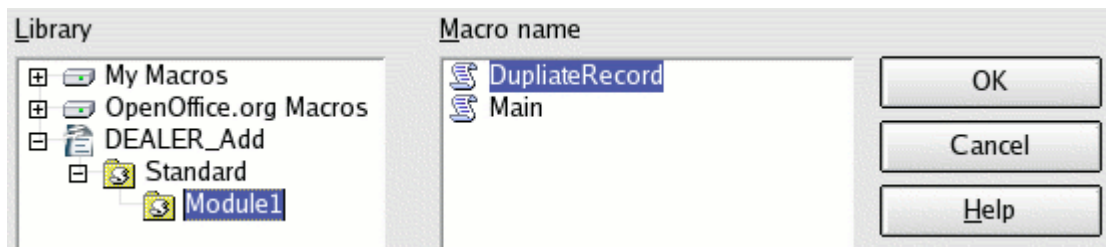



Figure 13: Use the DupliateRecord macro in the DEALER_Add form.

Click **OK** in the Assign Macro dialog, and the form is ready to test. Use **File > Save** to save the form into the Base document. Use the Design Mode On/Off icon () to toggle design mode off and test the form.

4.3. Show one item and the corresponding dealer

While displaying items one record at a time, the dealer is difficult to identify because the dealer's numeric identifier is not descriptive and easy to remember. It is, therefore, typical to show values from both tables. The form wizard can easily create the appropriate form.

Click on the Forms icon in the left column of the Base document. In the Tasks section, select *Use Wizard to Create Form* to start the Form wizard. On the first page of the wizard, select the ITEM table as the primary source of data for the main form. A query can also be used as the primary source of data. Next, move all of the fields in the “Available fields” list box to the “Fields in the form” list box. Use the >> button to move all of the fields at one time. Click the Next button to add a sub-form.

Figure 14: Select the table and fields for the main form.

A sub-form is a form that is inserted into another form. Use a sub-form to show data from tables or queries with a one-to-many relationship. Use the existing relationship. If you do not use an existing relationship, then you will be asked to explicitly specify the corresponding fields. Click the **Next** button to select which fields will be displayed from the DEALER table.

Figure 15: Create a sub-form for the DEALER record.

Select all of the fields from the DEALER table to be included in the sub-form. Click the **Next** button to choose how the controls will be arranged. Choose the *Columnar – Labels Left* format for both the form and the sub-form. Click the **Next** button to select the data entry mode. The default is to display all data and to allow modifications, deletions, and additions. Click the **Next** button to specify the look, use the default 3D look and color. Click the **Next** button to name the form. Set the name to `ITEM_Wizard_Simple`. Click the **Finish** button to save the form (see *Figure 16*). (?? Note that I had to use a development build because OOo version 2.0 failed with this).

After opening the form, you can move back and forth between the records using the first, previous (◀), next (▶), and last (▶▶) record icons. As you move through the different item records, the corresponding dealer record is displayed. If you move past the end of the last record, an empty record is displayed. This is the same as using the new record icon (▶+). If you accidentally move past the last record and, use the previous record button to get back; a new record will not be added to the table. If you accidentally add a new record, use the delete record icon (▶X).

ID	4
ITEM	Item 4
DEALER	3
ID	3
NAME	Dealer 3


Figure 16: The form contains all fields from both tables.

Although the dealer information is displayed in the sub-form, you can not move through the dealer records. Even worse, you can not use the data in the sub-form to set the dealer identifier in the ITEM table. The best solution to this problem depends on you, and the user.


4.4. Use a combo box with the dealer id

If the dealer identifier were descriptive, and simple, then a combo box can be easily used to choose the dealer. Use the form wizard to create a form for the ITEM table; do not include the DEALER field. Name the form ITEM_Dealer_ID_Combo.

After creating the simple form, right click on the form and choose **Edit**. Use **View > Toolbars** to enable the **Form Controls** and **Form Design** toolbars. If a toolbar is displayed, there is a check mark next to the toolbar name.

On the Form Controls toolbar, click on the Label Field icon (). The cursor will change to a cross-hairs shape. Draw a rectangle where you want the Dealer label; the size is not important because you can change the size later. Double click on the new label field to open the control properties window. In the control properties window, set the Name to LabelFieldDealer and set the Label to Dealer.

TIP If you click on (select) a control, green boxes are displayed around the control. A selected control can be moved, resized, copied, or deleted.

Click on the Combo Box icon () in the Form Controls toolbar and then draw a rectangle where you want the combo box. The Combo Box wizard is immediately displayed. First, you must choose from which table the combo box should obtain its values; choose the DEALER table and click **Next**. Although the Name field is more descriptive than the ID field, choose to display the ID field and click **Next**. Indicate that you want to save the value in the DEALER database field. You can only save the data in fields in the ITEM table, because the combo box is in the main form (there is only one form right now) and it is attached to the ITEM table. Click **Finish** to create the control.

You can test the form immediately by clicking on the Design Mode On/Off icon. As you move through the form, the combo box displays the correct value and you can directly set the dealer identifier by choosing a different value in the combo box. Unfortunately, this has not improved things; the dealer identifier is not descriptive.

Figure 17 shows a simple form with three fields. The first field is labeled 'ID' and contains the value '1'. The second field is labeled 'ITEM' and contains the value 'Item 1'. The third field is labeled 'Dealer' and contains the value '3' with a small downward arrow indicating a dropdown menu.

Figure 17: Simple form with a combo box

4.5. Use a list box with the dealer name

A list box provides more capabilities than a combo box. Sufficient capabilities, in fact, that this problem can be solved without writing a single macro. Right click on the `ITEM_Dealer_ID_Combo` form and choose **Copy**. Next, use **Edit > Paste** to create a new form; name the new form `ITEM_Dealer_Name_List`. Right click on the new form and choose **Edit**. Use **View > Toolbars** to enable the **Form Controls** and **Form Design** toolbars. If a toolbar is displayed, there is a check mark next to the toolbar name.

TIP In OOo version 2.0, the toolbars seem to move around a lot. Sometimes I find my toolbars floating around the screen in random locations, and sometimes they are docked to the top, bottom, or side of my current window. You can always dock a floating toolbar by dragging it where you want it.

Remove the existing combo box, it is no longer needed. In its place, insert a new list box (📋) control. After you draw out the desired location, the table selection dialog opens; select the `DEALER` table as the source of the displayed data and click the **Next** button. Select `NAME` as the displayed field and click the **Next** button. The list box allows you to display one field and link it to another field. Set the `DEALER` field from the “Value” table and the `ID` field from the “List” table (see Figure 18). Click the **Finish** button to create the control.

Figure 18 shows a dialog box for linking fields between two tables. The left table, titled 'Field from the Value Table', contains the fields `DEALER`, `ID`, `ITEM`, and `DEALER`. The right table, titled 'Field from the List Table', contains the fields `ID`, `ID`, and `NAME`. In the left table, the bottom `DEALER` field is selected. In the right table, the top `ID` field is selected.

Figure 18: Link `ITEM.DEALER` to `DEALER.ID`.

To emphasize what was just accomplished:

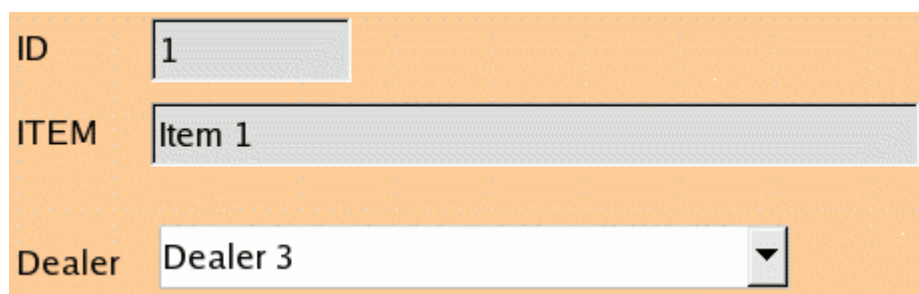
- A list box was created to display the `NAME` field from the `DEALER` table.
- The `DEALER` field in the `ITEM` table was linked to the `ID` field in the `DEALER` table.

The same information can be seen in the list box properties dialog. Double click on the newly created list box and look at the **Data** tab.

Table 8: Data properties for the list box control.

Data Property	
Data Field	Associate the list box to the DEALER field in the ITEM table.
Type	Use an SQL statement to fill the list box with data.
Content	SQL statement to use: SELECT "NAME", "ID" FROM "DEALER". Notice that the name and the id are retrieved.
Bound Field	The first field is bound, which means that the name is displayed. Notice that the name is the first field in the select statement.

Click on the General tab. Associate the Dealer label to the list box by clicking on the three dots to the right of the Label Field property and choosing the dealer label. Other notable properties include the Dropdown property, which causes the list box to be displayed as a “drop down” rather than a list box, and the multi-selection property, which prevents more than one item from being selected at a time.



The screenshot shows a form with an orange background. It contains three input fields. The first field is labeled 'ID' and contains the number '1'. The second field is labeled 'ITEM' and contains the text 'Item 1'. The third field is labeled 'Dealer' and is a dropdown menu currently displaying 'Dealer 3' with a downward arrow on the right side.

Figure 19: The dealer name displayed in a list box.

Save the form and test it. As you move through the item records, the correct dealer name is automatically displayed. Changing the dealer name in the drop down automatically updates the dealer id in the ITEM table. This is probably the solution of choice. There are a few points to keep in mind:

- 1) The list is stored in the same form as the related data. In other words, although the list box displays the dealer name, it is contained in the form displaying the item data. You can, therefore, select the item form and then insert the list box.
- 2) If wizards are enabled – there is a button in the controls toolbar to enable and disable wizards – a wizard will guide you through the process.
- 3) If you do not use a wizard, you must set the control properties manually. The data field is corresponding ID field, the contents type is SQL, and the list contents is similar to “SELECT "NAME", "ID" FROM "DEALER"”. The “bound field” entry refers to the data that is displayed from the query. With the provided SQL, the data to display is the NAME field, which corresponds to the first first column returned.

Using a listbox populated using a query provides a lot of control. You can really display almost anything.

4.6. Relations in a single table

Assume that you have Table1 with the following fields:

- ID – Integer auto-value as the key field.
- Title – Text field.
- Description – Text field.

Create a combo-box containing the values from the Title column that causes the corresponding Description text to be displayed in a text field.

4.6.1. Solution

Create a form in design view. Now, add controls to the form.

- 1) Open the form navigator.
- 2) In the form navigator, right click on Forms and choose **New > Form**. For me the form was automatically named Standard.
- 3) In the form navigator, right click on the Standard form and choose Properties.
- 4) In the form properties dialog, choose the Data tab.
- 5) Set the Content property to Table1, or what ever you named your table.
- 6) With the Standard form highlighted in the form navigator, click on the combo-box button in the Controls tool-bar. Draw a combo-box on the form. The combo-box wizard will open.
- 7) Select Table1 in the combo-box wizard and click Next.
- 8) For the display field, choose Title and click Next.
- 9) Choose “No, I only want to save the value in the form.” and click Finish.

If you toggle the design mode off to test your form, the combo-box contains values from the Title column. Toggle design mode on and then continue. Look at the properties for the combo-box, it obtains content using the following SQL:

```
SELECT DISTINCT "Title" FROM "Table1"
```

Now, add a text box to display the Description.

- 10) With the Standard form selected, add a text field.
- 11) In the Data tab of the properties dialog for the text field, set the Data field to Description.

At this point, the text field displays the Description for the current record. It is just a matter of telling the form which record to display. Add the following macro to the form:

Listing 24: *Filter a form based on text in a combo-box.*

```
Sub NewTitleSelected(oEvent)
    Dim oForm
    oForm = oEvent.Source.getModel().getParent()
    oForm.Filter = "Title=" & oEvent.Source.getText() & "="
```

```

oForm.ApplyFilter = True
oForm.reload()
End Sub

```

I tied the macro to the item status changed event, which is only called if a new item is selected using the mouse – modifying text using the keyboard does not cause an item status changed event. Another option is the text changed event, which is called when ever the text is modified, including from the keyboard. ??? I think that I tied this to the combo-box. Verify.

4.6.2. Solution characteristics

Each form is associated with a record in a table. The form containing the solution is associated with Table1. The text field is related to the current record. Moving through the records causes the text field to display the Description for the current record.

The combo-box is filled using a SQL statement and is not related to the current record. A macro filters the records that the form can visit based on the value in the combo-box.

It is possible, with a bit more work, to disassociate the text field from the current record. On the Data tab, do not associate the field to the table. The macro must now do all of the work.

Listing 25: *Set text field based on text in a combo-box.*

```

Sub NewTitleSelected(oEvent)
    Dim s$
    Dim oForm
    Dim oDoc
    Dim oContext      'Global database context.
    Dim oDataSource   'Data source for the specified database.
    Dim oCon           'Connection to a database.
    Dim sSQL$         'SQL that is executed.
    Dim oResult        'Result from an SQL statement.
    Dim oStatement     'A created statement that can execute SQL.
    Dim sQuote$

    oForm = oEvent.Source.getModel().getParent()
    oDoc = oForm

    REM Walk up the form component tree looking for
    REM the containing text document.
    s = "com.sun.star.text.TextDocument"
    Do While NOT oDoc.supportsService(s)
        oDoc = oDoc.getParent()
    Loop

    REM Now, get the containing database.
    s = "com.sun.star.sdb.OfficeDatabaseDocument"
    oDoc = oDoc.getParent()
    If IsNull(oDoc) OR IsEmpty(oDoc) Then
        Print "The form is not embedded in a Base document"
    End If
End Sub

```

```

Exit Sub
ElseIf Not oDoc.supportsService(s) Then
    Print "The form is not embedded in a Base document"
Exit Sub
End If

oContext = CreateUnoService("com.sun.star.sdb.DatabaseContext")
oDataSource = oContext.getByName(oDoc.getURL())
oCon = oDataSource.getConnection("", "")
sQuote = oCon.getMetaData().getIdentifierQuoteString()

oStatement = oCon.createStatement()
sSQL = "SELECT " & sQuote & "Description" & sQuote & " FROM " & _
    sQuote & "Table1" & sQuote & " WHERE " & _
    sQuote & "Title" & sQuote & "=" & _
    oEvent.Source.getText() & "'"

oResult = oStatement.executeQuery(sSQL)
s = ""
If NOT IsNull(oResult) AND NOT IsEmpty(oResult) Then
    If oResult.next() Then
        s = oResult.getString(1)
    End If
End If
oCon.close()
oForm.getByName("TextBox").setString(s)
End sub

```

4.7. Use a “help and fill” button

A typical solution to this problem is to provide a button or keyboard shortcut to display the values from another table. The selected value is then updated in the original table. I can think of a few possibilities on how to implement this. One solution, includes creating a form to display the DEALER entries. After choosing the dealer, an update item button is pressed on the dealer form. Paradox had a very nice feature for this, but I am too lazy to create an implementation right now.

I use a drop down list box in my example shown in Appendix A. Stuff I Own. Take a look at section A.2.2. Item One Table.

5. Many-to-many relationships

Next, setup a many to many relationship and discuss what to do with it in a manner similar to the one to many relationship. ?? Time, I lack time...

6. Database fields

A database table is a collection of records. Each record contains individual pieces of data called fields. This chapter discusses the types of data that can be stored in a database field, primarily as it relates to macro programmers.

Most major databases are built to conform to an SQL standard, which dictates numerous things, including the data types supported by the database. The primary standards are SQL89, SQL92, SQL99, and SQL2003—the number indicates the year that the standard was adopted. The data types defined by these standards are outlined in *Table 9*.

Table 9. General field types supported by SQL.

When	Type	Description
SQL99	ARRAY	Collection type.
SQL2003	ARRAY UNBOUNDED	Collection type.
SQL2003	BIGINT	Exact numeric value with precision 19.
SQL92	BINARY	Binary data of fixed length with a maximum length 255.
SQL92	BIT (Dropped SQL2003)	Fixed length n-length bit string.
SQL92	BIT VARYING (Dropped SQL2003)	Variable length bit string (up to n-bits).
SQL99	BOOLEAN	A single true or false value.
SQL89	CHAR	Character string of fixed string length with a maximum length 255.
SQL92	DATE	Year, month, and day fields, conforming to the rules of the Gregorian calendar .
SQL99	DECIMAL	Signed, exact, numeric value with a precision and scale.
SQL89	DOUBLE	Signed, approximate, numeric value with a binary precision 53.
SQL92	FLOAT	Signed, approximate, numeric value.
SQL92	INTEGER	Exact numeric value with precision 10 and scale 0.
SQL92	INTERVAL	Specify a time interval.
SQL99	INTERVAL_DAY	Number of days between two dates.
SQL99	INTERVAL_DAY_TO_HOUR	Number of days and hours between two dates and times.
SQL99	INTERVAL_DAY_TO_MINUTE	Number of days, hours, and minutes between two date and times.
SQL99	INTERVAL_DAY_TO_SECOND	Number of days, hours, minutes, and seconds between two dates and times.
SQL99	INTERVAL_HOUR	Number of hours between two dates and times.
SQL99	INTERVAL_HOUR_TO_MINUTE	Number of hours and minutes between two dates and times.

When	Type	Description
SQL99	INTERVAL_HOUR_TO_SECOND	Number of hours, minutes, and seconds between two dates and times.
SQL99	INTERVAL_MINUTE	Number of minutes between two dates and times.
SQL99	INTERVAL_MINUTE_TO_SECOND	Number of minutes and seconds between two dates and times.
SQL99	INTERVAL_MONTH	Number of months between two dates
SQL99	INTERVAL_SECOND	Number of seconds between two dates and times.
SQL99	INTERVAL_YEAR	Number of years between two dates.
SQL99	INTERVAL_YEAR_TO_MONTH	Number of years and months between two dates.
SQL92	LONG VARBINARY	Variable length binary data.
SQL92	LONG VARCHAR	Variable length character data; typically called a memo field.
SQL2003	MULTISET	Collection type.
SQL99	NUMERIC	Signed, exact, numeric value with a precision and scale.
SQL89	REAL	Signed, approximate, numeric value with a binary precision 24.
SQL89	SMALLINT	Exact numeric value with precision 5 and scale 0.
SQL92	TIME	Hour, minute, and second fields.
SQL92	TIMESTAMP	Year, month, day, hour, minute, and second fields.
SQL92	TINYINT	Exact numeric value with precision 3 and scale 0.
SQL92	VARBINARY	Variable length binary data with a maximum length 255.
SQL92	VARCHAR	Variable length character data with a maximum length 255.
SQL2003	XML	Native XML storage.

The different SQL standards build on each other. There are instances, however, where a data type is dropped; BIT was dropped by SQL2003. The most commonly supported and used data types are defined by SQL92. The data types supported by OpenOffice.org are shown in *Table 10*. The Type column contains a numeric value used internally by OpenOffice.org and is useful when modifying a database using macros. The Length column indicates the length values supported by OpenOffice.org. Many of the types defined by the SQL standard do not specify the maximum precision and length; the values are implementation specific. Precision and length information mentioned in *Table 10* is specific to the HSQLDB database implemented in OpenOffice.org.

Table 10. Field types supported by OpenOffice.org using HSQLDB.

Type Name	#	Length	Comment
Bit	-7	1	Boolean yes and no values.
TinyInt	-6	3	8 bit signed integer from -128 to 127.
BigInt	-5	19	64 bit signed integer from -9,223,372,036,854,775,808 to

Type Name	#	Length	Comment
			9,223,372,036,854,775,807.
LongVarBinary	-4	2147483647	Variable-length binary data such as an image.
VarBinary	-3	2147483647	Variable-length binary data with a maximum length.
Binary	-2	2147483647	Fixed-length binary data.
LongVarChar	-1	2147483647	Variable length character data, also known as a Memo field.
SQLNULL	0	0	Null value.
CHAR	1	1 to 2147483647	Fixed length text.
Numeric	2	1 to 646456993	Signed, exact numeric value with a specified precision and scale.
Decimal	3	1 to 646456993	Signed, exact numeric value with a specified precision and scale.
Integer	4	10	32 bit signed integer from -2,147,483,648 to 2,147,483,647.
SmallInt	5	5	16 bit signed integer from -32,768 to 32,767.
Float	6	17	Double precision (8 byte) floating point number.
Real	7	17	Double precision (8 byte) floating point number.
Double	8	17	Double precision (8 byte) floating point number.
VarChar	12	1 to 2147483647	Variable length text data.
VarChar_IgnoreCase	12	1 to 2147483647	Variable length text data that is not case sensitive. This is not a standard type.
Boolean	16	1	A numeric 0 is False and any other number is True.
Date	91	N/A	Year, month, and day fields, conforming to the rules of the Gregorian calendar.
Time	92	N/A	Hour, minute, and second fields.
TimeStamp	93	N/A	Date and time with a one second time resolution.
Other	111	21474383647	The SQL type is database-specific and is mapped to an object that can be accessed using the method XRow::getObject() .

Of the four data types—string, number, date-time, and interval—numbers have the most available data types and the greatest constraints on implementation. The SQL92 standard defines fundamental data types that mold the types found in the various SQL-based database implementations. Numbers pose the greatest risk when moving data between different database implementations because it is common to offer non-standard extensions. For example, the same type may have different default size limits in Oracle and SQL Server. It is, therefore, important that you understand the idiosyncrasies of the underlying database.

6.1. Storing numbers

All numbers have a precision, which indicates the number of significant digits. Some numeric types contain a scale value that indicates the position of the least significant digit to the right of the decimal. For example, the number 1234.56 has a precision of 6 and a scale of 2.

The SQL92 standard provides for built-in operations, such as addition, subtraction, multiplication, and division, in addition to functions that determine the length and other attributes required for value handling. The different numeric data types can be mixed during comparisons and numeric operations. In most database implementations, the result uses the data type with the greatest precision.

TIP Numeric data types are the most difficult to move between different database implementations because it is common to offer non-standard extensions. For example, support for unsigned integers or support for extended precision.

6.1.1. Integer numbers

The integer data types are used to store numbers with no decimal portion. All of the integer types use the same radix (numeric base and internal representation), round using the same rules, and are exact. The SQL standard specifies the following properties:

- Is an exact type, which store a literal representation of the number.
- Can uses decimal precision or binary precision, which is based on the number of bits used to represent the value. The represented range is implementation-specific.
- Has a scale of 0—no decimal digits.
- Has an implementation specific minimum and maximum precision.
- May have a vendor-supplied default value for the precision if no explicit value is specified.

The SQL92 standard defined the types INTEGER, SMALLINT, and TINYINT; BIGINT was introduced by SQL2003. Although each implementation is able to state the precision of the integer types, the following must be true: TINYINT ≤ SMALLINT ≤ INTEGER ≤ BIGINT. *Table 11* summarizes the differences between the integer data types implemented by the database used in OpenOffice.org.

Table 11. Integer types supported by HSQLDB.

Type	Size	Minimum value	Maximum value
TINYINT	1 byte	-128	127
SMALLINT	2 bytes	-32,767	32,767
INTEGER	4 bytes	-2,147,483,648	2,147,483,647
BIGINT	8 bytes	-9,223,372,036,854,775,808	9,223,372,036,854,775,807

6.1.2. Floating point numbers

Floating point numbers are numbers that can have numbers to the right of the decimal point—unlike an integer type, which can not. Floating point numbers are internally represented in scientific notation, for example, 1.23e-45. The Institute of Electrical and Electronics Engineers (IEEE) defined a format for storing floating point numbers. The IEEE standards are used by almost every computer that utilizes floating point numbers. Probably the most important thing to know about a particular floating point number is the number of relevant digits. For example, a double precision IEEE floating point number is accurate to roughly fifteen digits.

Floating point numbers are defined to use binary precision when rounding and are not exact. A format is not exact if some numbers are not represented exactly because the internal representation is not able to do so. At the heart of the problem is that numbers are usually represented in base 2, but the external representation is base 10. Although it is well understood that 1/3 may not be exactly representable, it is unintuitive that 0.01 is not. In IEEE single-precision format, 0.01 is represented as approximately 0.009999999776482582—the number that is stored is exactly 10737418/1073741824. It is not possible to exactly represent 0.01 using the IEEE floating point formats typically used by computers.

The SQL standard specifies that the floating point types have the following properties:

- Is an approximate numeric type, meaning that it represents an exponential format for a given value, for example, 1.23e-45. Although rounding and truncating for this type are defined largely by the manufacturer, the IEEE standards are almost always used.
- Uses binary precision when rounding.

TIP In OpenOffice.org, using HSQLDB, the three floating point types REAL, DOUBLE PRECISION, and FLOAT use the IEEE 8 byte double precision representation supporting values +/- 1.79769313486232 x 10E308.

The SQL standard defines three primary floating point data types REAL, DOUBLE PRECISION, and FLOAT. Although each implementation is able to state the precision of the floating point types, the precision of the REAL type must be less than or equal to the precision of the DOUBLE PRECISION type. In the SQL specification, the FLOAT type provides a mechanism to recommend the precision, the implementation can then choose the representation. In a typical implementation, the FLOAT type will be assigned to either the REAL or the DOUBLE PRECISION type depending on the requested precision. The HSQLDB database used internally by OpenOffice.org represents all three types as an IEEE 8 byte double precision number.

TIP In most implementations, including HSQLDB, the DOUBLE PRECISION type can be abbreviated as DOUBLE.

6.1.3. NUMERIC and DECIMAL types

The NUMERIC and DECIMAL types define exact numeric types that contain decimals. When defining a NUMERIC or a DECIMAL column, the total column length and the number of decimal digits is specified—the number of decimal digits is usually referred to as the scale. The SQL standard defines the types to have the following properties:

- Is an exact type, which store a literal representation of the number.
- Perform rounding in base 10 rather than the usual base 2. A typical implementation accomplishes this by storing the numbers as strings or in the binary coded decimal format.
- Has a total length equal to the defined precision, plus 1 if the scale is greater than 0.


Based on the SQL standard, the NUMERIC and DECIMAL types differ in their treatment of decimals. A NUMERIC type must use the number of decimal digits as specified by the scale. The DECIMAL type, however, must use at least as many decimal digits as specified by the scale, but may use more. The SQL definition allows the two types to be identical.

TIP The SQL standard uses the type NUMERIC. HSQLDB documentation and OOo frequently uses the word NUMBER rather than NUMERIC; they are the same type.

The GUI shortens numeric values, probably by manipulating them as Doubles. I tested the NUMERIC type with OOo version 2.0 release candidate 2. I created a NUMERIC column with a precision of 50 and a scale of 4. Using the GUI, I entered a 19 digit number, which the GUI shortened to 17 digits. Using SQL (see *Listing 26*), I was able to store and retrieve a long NUMERIC number, but the GUI always displayed the number to 17 significant digits.

Listing 26: Set and retrieve the value in a NUMERIC field.

```
s = "update \"Numbers\" Set \"Number2\"=1234567890123456789.1234 Where ID=3"
oStmt.executeQuery(s)
s = "select \"Number2\" from \"Numbers\" Where ID=3"
oResultSet = oStatement.executeQuery(s)
If Not IsNull(oResultSet) Then
    oResultSet.next
    Print "Return = " & oResultSet.getString(1)
End If
```

Caution  As of version 2.0, the OOo GUI does not properly update and display NUMERIC types. They data is appears to be treated as a double, which has a precision of 17 decimal digits. I tested this using both the Query designer and by editing a table directly. Be certain to wrap access to these tables in properly tested forms.

6.2. Bit and Boolean Types

HSQLDB supports the BOOLEAN type, which stores the values 'yes' and 'no'. When initialized with a numeric value, a zero value is translated to no and any other value is translated to yes. Although OpenOffice.org does not directly list the BIT data type, use the “yes/no” BOOLEAN type instead.

?? dBase you can check using yes/no, usually, you use true/false, I think that you can use 0/-1.

6.3. Date and time

Use the DATE type to store a date with no time. use the TIME type to store a time with no date. Although both the DATE and TIME type allow a display format that shows both a date and time, this is very confusing because neither type has both values. Use the TIMESTAMP type to store a date and time value.

The TIMESTAMP type stores and returns values in hundredths of a second. Although I could not set hundredths of a second using the standard GUI, I was able to use SQL to do this (see *Listing 27*). With OOo 2.0 RC 2, when the time stamp is obtained as a string, the value includes HundredthSeconds, but when obtained as a TimeStamp, it does not.

Listing 27: Set a *TIMESTAMP* to include hundredths of a second.

```
s = "update NUM Set TS='1965-03-13 01:02:03.123456789' Where ID=0"
oStmt.executeQuery(s)
oResultSet = oStatement.executeQuery("select TS from NUM Where ID=0")

If Not IsNull(oResultSet) Then
    oResultSet.next()
    oTS = oResultSet.getTimestamp(1)

    s = s & "TS1 = " & oTS.Year & "-" & oTS.Month & "-" & oTS.Day & " " & _
        oTS.Hours & ":" & oTS.Minutes & ":" & _
        oTS.Seconds & "." & oTS.HundredthSeconds & CHR$(10) & _
        "TS2 = " & oResultSet.getString(1) & CHR$(10)
    MsgBox s
End If
```

The real surprise for me is that I can not set ??

?? Comment on can not use the other type to obtain milliseconds.

?? Comment on default values and defining using DDL so that can set certain default values.

6.4. Text data

Use the CHAR type to store data with a fixed length. The advantage of a CHAR type is that it is efficient to store, index, and access. The disadvantage of the CHAR type, is that the maximum length is always used and is therefore wasteful of space. The VARCHAR type, however, is typically more efficient in storage, using only as much space as is required. The VARCHAR_IGNORECASE is a special case-insensitive non-portable type of VARCHAR.

TIP In some implementations, it is more space efficient to place the fixed length fields before the variable length fields.

Unlike the other text data types, the LONGVARCHAR type does not accept a length. The LONGVARCHAR type is frequently called a memo field because it allows for long rambling memos of no specific length—there is an implementation specific upper limit, but the limit is usually very large. In a typical implementation, the primary file contains a pointer to most or all of the “memo”, which is stored in a secondary file. The HSQLDB database stores all of the data in a single file. Some database implementations have difficulty accessing and using memo fields.

TIP I experimentally determined that the HSQLDB database uses the same amount of storage regardless of the type used. I also saw no speed advantage while searching the different text types. Despite these observations, you should choose data types realizing that the underlying implementation may change in the future.

6.5. Binary data

OpenOffice.org supports three primary binary types, BINARY, VARBINARY, and LONGVARBINARY. Based on the SQL standards, the binary types should behave similarly to the corresponding text types CHAR, VARCHAR, and LONGVARCHAR types; but binary data is stored rather than text data. The HSQLDB implementation used by OpenOffice.org, however, implements all binary types similarly to a memory field—you can not specify the length, but the length is very large.

TIP The HSQLDB implementation treats all the binary types as equivalent. You can not specify a maximum length, so the three binary types are treated as a LONGVARBINARY.

This document contains extensive examples of manipulations using binary data.

6.6. Other data type

The purpose of the OTHER data type in HSQLDB, is to store a serializable JAVA object. Objects stored as an OTHER type are always considered equal unless one of the objects is NULL. Finally, OTHER data can not be searched or joined other than tests for NULL.

6.7. Database sequences and auto-value fields

A database index is a lookup table that relates the value of fields to their location in the database. The value of the fields on which an index is based provide is called the index key. It is typically very fast to search an index for a specific value, and then the index returns a pointer into the database that allows the corresponding record to be obtained. If a large database is frequently accessed based on a person's last name, then an index should probably be created based on the last name field.

A key is a set of columns that can be used to identify or access a particular row or rows. A unique key is a key that is constrained so that no two values are equal. The primary key is a unique key that is selected to be the most important key for a table—each table can have only one primary key. A primary index is based on the primary key and a secondary index is based on columns that are not the primary key.

Although OpenOffice.org can access a database that does not contain a primary key, OOo can not (in general) update a database without a primary key. Neither a CVS text file nor a Calc document contain a primary key. The dBase access code is an exception to the primary key rule; dBase files do not contain primary keys, yet OOo is able to update these files.

It is very important, therefore, that add a primary key field to every table that you create. When no obvious primary key exists, the typical solution is to create an integer field that contains a sequence of integers. OpenOffice.org calls a field an auto-value field, if it can automatically set itself to the next available sequence number when a new row is inserted into the table.

TIP The auto-increment statement for an integer field in HSQLDB is “IDENTITY”.

HSQLDB, included with OOo, implements an Identity statement used to generate a sequence of integers. An auto-value integer field can be implicitly set with an automatically generated number—insert a null value into the auto-value column to implicitly insert the next sequence number. You can explicitly set the value stored in an auto-value field. If the explicitly entered value is greater than the next available sequence number, then the sequence is reset to the explicit value. *Listing 28* demonstrates implicitly and explicitly setting an auto-value field while inserting new rows in a database.

Listing 28. *Implicitly and explicitly set an auto-value field.*

```
CALL IDENTITY()                ** Assume returns 50
INSERT INTO `Table1` VALUES (NULL, 'x0')    ** Insert (50, 'x1')
INSERT INTO `Table1` VALUES (137, 'x1')      ** Insert (137, 'x2')
INSERT INTO `Table1` VALUES (130, 'x2')      ** Insert (130, 'x3')
INSERT INTO `Table1` VALUES (130, 'x3')      ** Failure, 130 exists.
INSERT INTO `Table1` VALUES (NULL, 'x4')     ** Insert (138, 'x0')
INSERT INTO `Table1` VALUES (NULL, IDENTITY()) ** Insert (139, 138)
```

7. A few easy database definitions

Although database terminology is not difficult, some words have a different meaning depending on the context and database system. Words that are likely to be used while creating a new database are shown and defined in *Table 12* as they are used in this document. A database power user will already know and be comfortable with the definitions.

Note Many of OpenOffice.org's APIs accept arguments that specify the schema and catalog, which is why I chose to define them. The definitions shown in Table 5 are intended to provide a broad view for an inexperienced user. The terms schema and catalog are generally not required while using a simple database. In other words, don't panic, and don't spend a lot of time memorizing definitions.

A database table is a collection of records. Each record contains individual pieces of data called fields. The SQL 92 standard introduces the terms catalog and schema, which refer to the organization of data in database systems. Fields are contained in records, records are contained in tables, tables are contained in schemas, and schemas are contained in catalogs. Finally, all of the elements are stored in a database. A schema is also likely to contain views, aliases, indexes, triggers, and structured types.

Table 12. Simplistic database definitions

Term	Simplistic Definition
Field	Individual piece of data such as a first name, or date.
Record	Collection of related fields, treated as a single unit; a record is a single row of fields.
Table	Collection of records.
Schema	Collection of tables, views, aliases, indexes, triggers, and structured types. Frequently used to mean the structure of a group of tables.
Catalog	Collection of schemas.
Database	Encapsulates all of the items, including catalogs, schemas, tables, records, and fields.

Many database systems do not support catalogs or schemas, especially smaller database systems. Larger database systems are more likely to support catalogs and schemas, but the names and precise definitions differ between different database systems. For example, in some systems a table is called a catalog and in others a table is called a schema—the different usages were in place before the SQL92 standard provided a common definition.

TIP Internally, the HSQLDB supports schemas, but not catalogs. Schema support is new, and was added shortly before the release of OOo version 2.0. The initial release of OOo version 2.0 does not fully support schemas when used with their internal database due to time constraints; better support will be added later.

7.1. Schema

According to the dictionary, a schema refers to a structure that represents some aspect of the world. The SQL92 definition effectively organizes an entire database hierarchy into schemas. A common usage for a schema is to store table definitions, relationships, and access rights in one schema, and the general data tables in a different schema. While working with an unfamiliar database, I sometimes search the information schema to determine existing table names—assuming that I have access to the information schema.

With regards to database systems, the word schema frequently refers to the database structure, and is frequently represented as a table or a graphic. Although you must determine the meaning based on the context, this is usually easy to do.

8. Database connections

8.1. Obtain a database context

The first step in accessing a database, is to open the database context (see *Listing 29*). The database context is used to manipulate registered data sources and to obtain a data source for a database that is not registered. *Table 13* provides a brief description of methods supported by the database context.

Listing 29. Create a global database context.

```
oBaseContext = CreateUnoService( "com.sun.star.sdb.DatabaseContext" )
```

TIP	Do not dispose a database context; the database context is a global service and there is only one instance, which is always returned. If you dispose a database context, the disposed unusable context will be returned. You must restart OpenOffice.org before you can obtain another database context.
------------	--

Table 13. Some methods supported by a DatabaseContext.

Method	Description
addContainerListener(listener)	Registering a listener to be notified when new data source is created or removed.
createEnumeration()	Return an enumeration of registered data sources.
dispose()	Dispose this global object. Do not use this method!
getByName(name)	Retrieve a data source, registered or not, based on the registered name or database URL.
getElementNames()	Return an array of registered data source names.
getRegisteredObject(name)	Return the registered data source.
hasByName(name)	Return True if the specified name is a registered data source.
hasElements()	Return True if there is at least one registered data source.
registerObject(name, data_source)	Register the supplied data source.
removeContainerListener(listener)	Remove a registered container listener.
revokeObject(name)	Unregister the named data source.

8.1.1. Registered data sources

Use the database context to obtain a data source for either a registered or an unregistered data source. Some methods relate only to registered data sources. For example, the method `hasByName(name)` returns `True` if the database context contains a registered a database with the specified name, but `getByName(url)` returns a data source even if it is not registered. The macro in *Listing 30* displays the currently registered data sources and then demonstrates how to enumerate them.

Listing 30. List the currently registered database services.

```
Sub ListRegisteredDataSources()
    Dim oBaseContext 'Global database context.
    Dim oEnum         'Enumeration of registered data sources.
    Dim oDataSource   'Database source

    oBaseContext = CreateUnoService("com.sun.star.sdb.DatabaseContext")
    If NOT oBaseContext.hasElements() Then
        Print "There are no registered data sources."
        Exit Sub
    End If

    REM Use hasByName(name) to verify that a data source is registered.
    REM use getElementNames() to obtain a list of registered data sources.
    REM Calling oBaseContext.getByName(name) returns a data source.
    MsgBox Join(oBaseContext.getElementNames(), CHR$(10)), 0, _
        "Registered Sources"

    REM Enumerate the currently registered data sources.
    oEnum = oBaseContext.createEnumeration()
    Do While oEnum.hasMoreElements()
        oDataSource = oEnum.nextElement()
    Loop
End Sub
```

8.1.2. Unregistering a data source

While registered, an OOo Base file is locked, preventing it from being deleted or otherwise manipulated. Deleting a data source from the data source view will unregister the data source; use **F4** to open the data source view. A data source can also be unregistered using a macro (see *Listing 31*).

Listing 31. Unregister a data sources.

```
oBaseContext = CreateUnoService("com.sun.star.sdb.DatabaseContext")
oBaseContext.revokeObject("MemTest")
```

8.1.3. Registering a data source

Prior to OpenOffice.org version 2.0, a data source needed to be registered before it could be accessed; this is no longer true. A data source can be obtained based on the document's URL, but this will not register the data source. *Listing 32* demonstrates how to obtain and register a data source. A data source can be obtained based on the URL, even if it is already registered by another name—notice the use of `GetSourceCodeDir()` from *Listing 57*.

Listing 32. Register a data source.

```
oBaseContext = CreateUnoService("com.sun.star.sdb.DatabaseContext")
sName = GetSourceCodeDir() & "MemTest.odt"
oDataSource = oBaseContext.getByURL(sName)

REM Register the object if you want, but this is not required for use.
oBaseContext.registerObject("MemTest", oDataSource)
```

TIP You can manipulate a database without a registered data source. The data source is the database document. *Listing 32* demonstrates how to obtain a data source based on its URL and how to register the data source.

A data source always corresponds to a database document; even with the “Bibliography” data source. The data source is available from the `DatabaseContext` using the URL of the database document. The data source contains a reference to the database document using the `DatabaseDocument` property, which can be opened the same way as any other OpenOffice.org document. *Listing 62* demonstrates how to find a currently open document based on the document's URL. The primary difference between loading a document directly, and obtaining the data source from the `DatabaseContext` (see *Listing 32*), is the absence of a visible user interface when using a `DatabaseContext`.

8.2. Connect to a database

A database connection represents a session with a specific database. To manipulate or modify the data contained in a database, a database connection is required. The connection provides the ability to execute SQL statements to modify and obtain information. *Listing 33* demonstrates how to open a connection to the Bibliography database, which is included with OOo.

Listing 33. Open a connection to the Bibliography database.

```
Dim oBaseContext 'Global database context.
Dim oDataSource 'Data source for the specified database.
Dim oCon 'Connection to a database.
Dim sUser$ 'User name while connecting.
Dim sPassword$ 'Password while connections.

REM Set the user name and password for connection.
REM There is no user or password required so set to an empty string.
sUser = ""
sPassword = ""

oBaseContext = CreateUnoService("com.sun.star.sdb.DatabaseContext")
```

```
oDataSource = oBaseContext.getByName("Bibliography")
oCon = oDataSource.getConnection(sUser, sPassword)
oCon.close()
```

TIP Although connections are automatically closed when the macro finishes, it is considered good practice to close a connection when you are finished. If the macro exits abnormally, the connection may not close.

8.3. Connect using an interaction handler

The code in Listing 33 demonstrates how to open a connection to a database by supplying both the user name and password. The Bibliography database does not require a user name and a password, so empty strings are used. OpenOffice.org provides a mechanism to automatically prompt for a password if it is required (see Listing 34).

Listing 34. Use an interaction handler to open the database.

```
Dim oBaseContext 'Global database context.
Dim oDataSource 'Data source for the specified database.
Dim oHandler 'Interaction handler in case a password is required.
Dim oCon 'Connection to a database.

oBaseContext = CreateUnoService("com.sun.star.sdb.DatabaseContext")
oDataSource = oBaseContext.getByName("Bibliography")
oHandler = createUnoService("com.sun.star.sdb.InteractionHandler")
oCon = oDataSource.ConnectWithCompletion(oHandler)
oCon.close()
```

8.4. Connections

A connection represents a session with a specific database. A connection is used to execute SQL statements and obtain results. A database connection is able to provide meta data concerning the connection.

Table 14. Methods supported by the com.sun.star.sdbc.Connection service.

Method	Description
createStatement()	Return a new statement object for executing SQL. SQL statements that are executed many times should use a prepared statement instead.
prepareStatement(sql)	Return a prepared statement object for executing parameterized SQL statements. Use '?' as the parameter place holder.
prepareCall(sql)	Return a callable statement object for calling database stored procedures. Use '?' as the parameter place holder.
nativeSQL(sql)	A driver may transform SQL before sending it to the database, the nativeSQL() method returns the transformed SQL without sending it to the database.
setAutoCommit(boolean)	Enable (True) or disable (False) a connection's auto-commit mode. In auto-commit mode, all SQL statements are executed and committed as individual transactions. Otherwise, SQL statements are grouped into transactions that are terminated by a call to either commit() or rollback(). A commit occurs when a statement completes or the next execute occurs, whichever comes first. If a result set is obtained, the statement completes when the last row from the

Method	Description
	result set is retrieved or the result set is closed.
getAutoCommit()	Return the current auto-commit state. By default, new connections are in auto-commit mode.
close()	Close the connection. The connection is automatically closed when it is no longer referenced.
commit()	Cause all changes made since the last commit or rollback to become permanent, releasing database locks held by the connection—use only when auto-commit mode is disabled.
rollback()	Discard all changes made since the last commit or rollback, releasing database locks held by the connection—use only when auto-commit mode is disabled.
isClosed()	Return True if the connection is closed.
getMetaData()	Return the meta-data for the connection's database.
setReadOnly(boolean)	Enable (True), or disable, read-only mode as a hint to enable database optimizations.
isReadOnly()	Return True if the connection is in read-only mode.
setCatalog(catalog_name)	Set a catalog name to select a subspace of this connection's database in which to work. If the driver does not support catalogs, the request is ignored.
getCatalog()	Return the name of the connection's current catalog.
setTransactionIsolation(long)	Change the transaction isolation level.
getTransactionIsolation()	Return the connection's current transaction isolation level.
getTypeMap()	Return the type map, if any, associated with this connection.
setTypeMap(map)	Install the given type map as the type map for this connection. The type map is used for the custom mapping of SQL structured types and distinct types. Not all drivers support type maps.
getWarnings()	Return the first warning or error. The warning object contains a link to the following warnings.
clearWarnings()	Clears all warnings.

Transactions prevent data from being read in an inconsistent state. For example, reading a row changed by one transaction that is later rolled back. *Table 15* lists the three primary types of “bad” reads that are preventable using transactions.

Table 15. Read types that produce inconsistent data.

Read	Description
Dirty	Reading a row with uncommitted changes.
Non-repeatable	Reading a row before and after a second transaction modifies the row.
Phantom	A phantom read requires the following sequence of events: A transaction using a WHERE clause reads rows before and after, a second transaction inserts a row that matches the WHERE clause. The extra rows read after the insertion are called phantom rows.

The transaction isolation level is set based on the TransactionIsolation constants; not all drivers support all transaction isolation level.

Table 16. Constants in the *com.sun.star.sdbc.TransactionIsolation* constant group.

Name	#	Description
NONE	0	Transactions are not supported.
READ_UNCOMMITTED	1	Rows that are not yet committed can be read; allows dirty, non-repeatable, and phantom reads.
READ_COMMITTED	2	Prevent dirty reads but allows non-repeatable and phantom reads.
REPEATABLE_READ	4	Prevents dirty and non-repeatable reads, but allows phantom reads.
SERIALIZABLE	8	Prevents dirty, non-repeatable and phantom reads.

8.4.1. Extended SDB connections

Every connection supports the *com.sun.star.sdbc.Connection* service. Some connections support the *com.sun.star.sdb.Connection* service, which provides access to the data definitions of a connected database. The extended connection can be used to control the access rights on database objects. Access to the tables, using *getTables()* is always supported and optionally, the methods *getViews()*, *getUsers()*, and *getGroups()* may also be present.

The SDB connection implements the *prepareCommand(sql\$, CommandType)* method (see Table 19), which returns a *PreparedStatement* object for sending parameterized SQL statements to the database. A SQL statement with or without IN parameters can be pre-compiled and stored in a *PreparedStatement* object. This object can then be used to efficiently execute the statement multiple times.

Table 17. Values for the *com.sun.star.sdb.CommandType* constant group.

Value	Name	Description
0	TABLE	The command contains a table name, which can be used to process a command like "SELECT * FROM tablename".
1	QUERY	The command contains a name of a query component, which contains a certain statement.
2	COMMAND	The command is an SQL statement.

8.4.2. Meta-data

Use a connection's *getMetaData()* method to obtain comprehensive information about the database as a whole. Some meta-data object methods return simple data types, and others return a result set with multiple rows. Different result set types exist. Some methods supported by the meta-data object accept a result set type as an argument; *insertsAreDetected()* for example.

Table 18. Values for the *com.sun.star.sdbc.ResultSetType* constant group.

Value	Name	Description
1003	FORWARD_ONLY	The row set cursor can move only forward.

Value	Name	Description
1004	SCROLL_INSENSITIVE	The row set cursor is scrollable but generally not sensitive to changes made by others.
1005	SCROLL_SENSITIVE	The row set cursor is scrollable and generally sensitive to changes made by others.

Table 19. Values for the *com.sun.star.sdbc.ResultSetConcurrency* constant group.

Value	Name	Description
1007	READ_ONLY	Concurrency mode for a result set that may not be updated.
1008	UPDATABLE	Concurrency mode for a result set that may be updated.

Routines that accept string patterns, such as a table name, support simple regular expressions. The string, “%” means match any substring of 0 or more characters, and “_” means match any one character.

Caution



The API documentation indicates that a NULL reference for a search pattern causes that argument's criteria to be dropped from the search. Unfortunately, it is not valid to pass a NULL reference using UNO. Some platforms generate a run time error and some platforms silently allow it. Do not allow assume that a NULL value is valid.

Table 20. Methods supported by the *com.sun.star.sdbc.XDatabaseMetaData* interface.

Method	Description
<code>allProceduresAreCallable()</code>	True, if the current user can call every procedure returned by the <code>getProcedures()</code> method.
<code>allTablesAreSelectable()</code>	True, if the current user can SELECT every table returned by the <code>getTables()</code> method.
<code>dataDefinitionCausesTransactionCommit()</code>	True, if a data definition statement within a transaction can force the transaction to commit.
<code>dataDefinitionIgnoredInTransactions()</code>	True, if a data definition statement within a transaction is ignored.
<code>deletesAreDetected(ResultSetType)</code>	True, if the <code>rowDeleted()</code> method of the result set can detect a row deletion.
<code>doesMaxRowSizeIncludeBlobs()</code>	True if <code>getMaxRowSize()</code> includes LONGVARCHAR and LONGVARBINARY blobs.
<code>getBestRowIdentifier(catalog, schema, table, scope, nullable)</code>	Result set describing a table's optimal set of columns that uniquely identify a row.
<code>getCatalogs()</code>	Result set of the available catalog names.
<code>getCatalogSeparator()</code>	The separator between a catalog and table name.
<code>getCatalogTerm()</code>	The database vendor's preferred term for “catalog”.
<code>getColumnPrivileges(catalog, schema, table, column)</code>	Result set with a description of the access rights for a table's columns.

Method	Description
getColumns(catalog,schema, table, column)	Result set with a description of the table columns. I used this to find table names as well.
getConnection()	Connection that produced this meta-data object.
getCrossReference(primaryCatalog, primarySchema, primaryTable, foreignCatalog, foreignSchema, foreignTable)	Result set with a description of the foreign key columns in the foreign key table that reference the primary key columns of the primary key table—describe how one table imports another's key.
getDatabaseProductName()	Name of the database product.
getDatabaseProductVersion()	Version of the database product.
getDefaultTransactionIsolation()	Default database transaction isolation level.
getDriverMajorVersion()	Major version number of the JDBC driver.
getDriverMinorVersion()	Minor version number of the JDBC driver.
getDriverName()	Name of the JDBC driver.
getDriverVersion()	Version number of the JDBC driver.
getExportedKeys(catalog, schema, table)	A row set that describes the foreign key columns that reference a table's primary key columns—the foreign keys exported by a table.
getExtraNameCharacters()	All the "extra" characters that can be used in unquoted identifier names (those beyond a-z, A-Z, 0-9 and _).
getIdentifierQuoteString()	The string used to quote SQL identifiers. A space " " is returned if identifier quoting is not supported.
getImportedKeys(catalog, schema, table)	A row set that describes the primary key columns that are referenced by a table's foreign key columns—the primary keys imported by a table.
getIndexInfo(catalog, schema, table, unique, approximate)	A row set that describes a table's indexes and statistics.
getMaxBinaryLiteralLength()	Maximum number of hex characters in a binary literal.
getMaxCatalogNameLength()	Maximum length of a catalog name.
getMaxCharLiteralLength()	Maximum length for a character literal.
getMaxColumnNameLength()	Maximum length of a column name.
getMaxColumnsInGroupBy()	Maximum number of columns in a "GROUP BY" clause.
getMaxColumnsInIndex()	Maximum number of columns allowed in an index.
getMaxColumnsInOrderBy()	Maximum number of columns in an "ORDER BY" clause.
getMaxColumnsInSelect()	Maximum number of columns in a "SELECT" list.
getMaxColumnsInTable()	Maximum number of columns in a table.
getMaxConnections()	Maximum number of concurrent active connections.
getMaxCursorNameLength()	Maximum length of a cursor name.
getMaxIndexLength()	Maximum index length (in bytes).
getMaxProcedureNameLength()	Maximum length of a procedure name.
getMaxRowSize()	Maximum length of a single row.

Method	Description
getMaxSchemaNameLength()	Maximum length allowed for a schema name.
getMaxStatementLength()	Maximum length of an SQL statement.
getMaxStatements()	Maximal number of open concurrent active statements.
getMaxTableNameLength()	Maximum length of a table name.
getMaxTablesInSelect()	Maximum number of tables in a SELECT statement.
getMaxUserNameLength()	Maximum length of a user name.
getNumericFunctions()	Comma-separated list of math functions.
getPrimaryKeys(catalog, schema, table)	A row set that describes a table's primary key columns.
getProcedureColumns(catalog, schema, procedure, column)	A row set that describes the stored procedure's parameters and result columns.
getProcedures(catalog, schema, procedure)	A row set that describes the stored procedures in a catalog.
getProcedureTerm()	The database vendor's preferred term for "procedure".
getSchemas()	A row set that describes the available schema names.
getSchemaTerm()	The database vendor's preferred term for "schema".
getSearchStringEscape()	The string used to escape wild-card characters '_' or '%'.
getSQLKeywords()	Comma-separated list of SQL keywords not in SQL92.
getStringFunctions()	Comma-separated list of string functions.
getSystemFunctions()	Comma-separated list of system functions.
getTablePrivileges(catalog, schema, table)	A row set that describes the access rights for each table.
getTables(catalog, schema, table, types())	A row set that describes the available tables.
getTableTypes(catalog, schema, table, column)	A row set with one column with the available table types.
getTimeDateFunctions()	Comma-separated list of time and date functions.
getTypeInfo()	A row set that describes the supported standard SQL types.
getUDTs(catalog, schema, type_name, types())	A row set that describes the user-defined types. Supported types include OBJECT, STRUCT, and DISTINCT.
getURL()	Directory and filename of the database.
getUserName()	User name for the connection.
getVersionColumns(catalog, schema, table)	A row set that describes a table's columns that are automatically updated when any value in a row is updated.
insertsAreDetected(ResultSetType)	True, if the rowInserted() method of the result set can detect a row insertion.
isCatalogAtStart()	True, if a catalog appears at the start of a qualified table name; otherwise it appears at the end.
isReadOnly()	True, if the database is in read-only mode.
nullPlusNonNullIsNull()	True, if concatenations between NULL and non-NULL values are NULL.

Method	Description
<code>nullsAreSortedAtEnd()</code>	True, if NULL values are sorted at the end.
<code>nullsAreSortedAtStart()</code>	True, if NULL values are sorted at the start.
<code>nullsAreSortedHigh()</code>	True, if NULL values are sorted high.
<code>nullsAreSortedLow()</code>	True, if NULL values are sorted low.
<code>othersDeletesAreVisible(ResultSetType)</code>	True, if deletes made by others are visible.
<code>othersInsertsAreVisible(ResultSetType)</code>	True, if inserts made by others are visible.
<code>othersUpdatesAreVisible(ResultSetType)</code>	True, if updates made by others are visible.
<code>ownDeletesAreVisible(ResultSetType)</code>	True, if a result set's own deletes are visible.
<code>ownInsertsAreVisible(ResultSetType)</code>	True, if a result set's own inserts are visible.
<code>ownUpdatesAreVisible(ResultSetType)</code>	True, if a result set's own updates are visible.
<code>storesLowerCaseIdentifiers()</code>	True, if the database treats mixed case unquoted SQL identifiers as case insensitive and stores them in lower case.
<code>storesLowerCaseQuotedIdentifiers()</code>	True, if the database treats mixed case quoted SQL identifiers as case insensitive and stores them in lower case.
<code>storesMixedCaseIdentifiers()</code>	True, if the database treats mixed case unquoted SQL identifiers as case insensitive and stores them in mixed case.
<code>storesMixedCaseQuotedIdentifiers()</code>	True, if the database treats mixed case quoted SQL identifiers as case insensitive and stores them in mixed case.
<code>storesUpperCaseIdentifiers()</code>	True, if the database treats mixed case unquoted SQL identifiers as case insensitive and stores them in upper case.
<code>storesUpperCaseQuotedIdentifiers()</code>	True, if the database treats mixed case quoted SQL identifiers as case insensitive and stores them in upper case.
<code>supportsAlterTableWithAddColumn()</code>	True, if the Database supports "ALTER TABLE" with add column.
<code>supportsAlterTableWithDropColumn()</code>	True, if the Database supports "ALTER TABLE" with drop column.
<code>supportsANSI92EntryLevelSQL()</code>	True, if the database supports ANSI92 entry level SQL grammar.
<code>supportsANSI92FullSQL()</code>	True, if the database supports ANSI92 full SQL grammar.
<code>supportsANSI92IntermediateSQL()</code>	True, if the database supports ANSI92 intermediate SQL grammar.
<code>supportsBatchUpdates()</code>	True, if the driver supports batch updates.
<code>supportsCatalogsInDataManipulation()</code>	True, if a catalog name can be used in a data manipulation statement.
<code>supportsCatalogsInIndexDefinitions()</code>	True, if a catalog name can be used in an index definition statement.
<code>supportsCatalogsInPrivilegeDefinitions()</code>	True, if a catalog name can be used in a privilege

Method	Description
	definition statement.
supportsCatalogsInProcedureCalls()	True, if a catalog name be used in a procedure call statement.
supportsCatalogsInTableDefinitions()	True, if a catalog name be used in a table definition statement.
supportsColumnAliasing()	True, if the Database supports column aliasing.
supportsConvert(fromType, toType)	True, if the Database can convert between the two SQL types. The types are represented as an integer.
supportsCoreSQLGrammar()	True, if the database supports ODBC Core SQL grammar.
supportsCorrelatedSubqueries()	True, if correlated sub-queries are supported.
supportsDataDefinitionAndDataManipulationTransactions()	True, if the Database supports both data definition and data manipulation statements within the same transaction.
supportsDataManipulationTransactionsOnly()	True, if only data manipulation statements within a transaction are supported.
supportsDifferentTableCorrelationNames()	True, if table correlation names are restricted to names different from the actual table names.
supportsExpressionsInOrderBy()	True, if expressions in "ORDER BY" lists are supported.
supportsExtendedSQLGrammar()	True, if the ODBC Extended SQL grammar is supported.
supportsFullOuterJoins()	True, if full outer joins are supported.
supportsGroupBy()	True, if some form of the "GROUP BY" clause is supported.
supportsGroupByBeyondSelect()	True, if a "GROUP BY" clause can add columns not in the SELECT—assuming it specifies all columns in the SELECT.
supportsGroupByUnrelated()	True, if a "GROUP BY" clause can use columns not in the SELECT.
supportsIntegrityEnhancementFacility()	True, if the SQL Integrity Enhancement Facility is supported.
supportsLikeEscapeClause()	True, if the escape character is supported in "LIKE" clauses.
supportsLimitedOuterJoins()	True, if limited outer joins are supported.
supportsMinimumSQLGrammar()	True, if the ODBC Minimum SQL grammar is supported.
supportsMixedCaseIdentifiers()	True, if unquoted SQL identifiers are case sensitive.
supportsMixedCaseQuotedIdentifiers()	True, if quoted SQL identifiers are case sensitive.
supportsMultipleResultSets()	True, if multiple result sets from a single execute are supported.
supportsMultipleTransactions()	True, if multiple transactions on different connections can be open at once.
supportsNonNullableColumns()	True, if columns be defined as not nullable.
supportsOpenCursorsAcrossCommit()	True, if cursors can remain open across commits.

Method	Description
supportsOpenCursorsAcrossRollback()	True, if cursors can remain open across rollbacks.
supportsOpenStatementsAcrossCommit()	True, if statements can remain open across commits.
supportsOpenStatementsAcrossRollback()	True, if statements can remain open across rollbacks.
supportsOrderByUnrelated()	True, if an "ORDER BY" clause use columns not in the SELECT statement.
supportsOuterJoins()	True, if outer joins are supported.
supportsPositionedDelete()	True, if positioned DELETE is supported.
supportsPositionedUpdate()	True, if positioned UPDATE is supported.
supportsResultSetConcurrency(ResultSetType, concurrency)	Does the database support the concurrency type in combination with the given result set type.
supportsResultSetType(ResultSetType)	True, if the given result set type is supported.
supportsSchemasInDataManipulation()	True, if a schema name can be used in a data manipulation statement.
supportsSchemasInIndexDefinitions()	True, if a schema name can be used in an index definition statement.
supportsSchemasInPrivilegeDefinitions()	True, if a schema name can be used in a privilege definition statement.
supportsSchemasInProcedureCalls()	True, if a schema name can be used in a procedure call statement.
supportsSchemasInTableDefinitions()	True, if a schema name be used in a table definition statement.
supportsSelectForUpdate()	True, if SELECT for UPDATE is supported.
supportsStoredProcedures()	True, if stored procedure calls using the stored procedure escape syntax are supported.
supportsSubqueriesInComparisons()	True, if sub-queries in comparison expressions are supported.
supportsSubqueriesInExists()	True, if sub-queries in 'exists' expressions are supported.
supportsSubqueriesInIns()	True, if sub-queries in 'in' statements are supported.
supportsSubqueriesInQuantifieds()	True, if sub-queries in quantified expressions are supported.
supportsTableCorrelationNames()	True, if table correlation names are supported.
supportsTransactionIsolationLevel(level)	True, if the given transaction isolation level is supported.
supportsTransactions()	True, if transactions are supported.
supportsTypeConversion()	True, if the CONVERT function between SQL types is supported.
supportsUnion()	True, if the SQL UNION statement is supported.
supportsUnionAll()	True, if the SQL UNION ALL statement is supported.
updatesAreDetected(ResultSetType)	True, if a row update can be detected by calling the method rowUpdated() on the result set object.
usesLocalFilePerTable()	True, if one local file is used for each table.
usesLocalFiles()	True, if the local files are used to save the tables.

8.4.3. Inspecting the meta-data

To demonstrate how to access the meta data, I wrote a macro that retrieves the meta-data by calling the methods in *Listing 35*, and then stores the data in a Calc document. First, a new Calc document is created to contain the resulting data.

Listing 35. Inspect the meta data from a connection.

```
Function InspectMetaData(ByVal oMeta)
    Dim s$           'Utility string variable.
    Dim oResult      'Result from an SQL statement.
    Dim i As Long    'General index variable.
    Dim j As Long    'General index variable.
    Dim k As Long    'General index variable.
    Dim oDoc         'Calc document that will contain the presentation data.
    Dim sNewUrl$     'URL for a new calc document.
    Dim oTables()    'Contains the primary data tables.
    Dim oData1()     'Generic data variable.
    Dim oData2()     'Generic data variable.
    Dim oData3()     'Generic data variable.
    Dim oData4()     'Generic data variable.
    Dim oNull As Object

    If IsNull(oMeta) OR IsEmpty(oMeta) Then
        Print "The meta-data is null or empty"
        Exit Function
    End If

    REM Open a new calc document to hold the data!
    sNewUrl = "private:factory/scalc"
    oDoc = StarDesktop.loadComponentFromURL(sNewUrl, "_blank", 0, Array())

    REM Start by obtaining the list of regular tables.
    oResult = oMeta.getTables(oNull, "%", "%", Array("TABLE"))
    Redim oTables()
    Do While oResult.next()
        AppendToArray(oTables(), oResult.getString(3))
    Loop

    RunMultipleCallsAppendToDoc(oDoc, oMeta, _
        Array("allProceduresAreCallable", _
            "allTablesAreSelectable", _
            "dataDefinitionCausesTransactionCommit", _
            "dataDefinitionIgnoredInTransactions", _
            "doesMaxRowSizeIncludeBlobs"))

    REM The arguments are (catalog, schema, table, scope, nullable)
    REM The scope can be any value from the com.sun.star.sdbc.BestRowScope
    REM constants including TEMPORARY (0), TRANSACTION (1),
    REM and SESSION (2).
    For i = LBound(oTables()) To UBound(oTables())
        oResult = oMeta.getBestRowIdentifier(oNull, "%", oTables(i), _
            com.sun.star.sdbc.BestRowScope.SESSION, True)
        AddResultSetToDoc(oResult, oDoc, _
```

```

        "getBestRowIdentifier(" & oTables(i) & ")")
Next

REM There is a single column CATALOG returned.
oResult = oMeta.getCatalogs()
AddResultSetToDoc(oResult, oDoc, "getCatalogs")

RunMultipleCallsAppendToDoc(oDoc, oMeta, _
    Array("getCatalogSeparator", "getCatalogTerm"))

REM getColumnPrivileges(catalog,schema, table, columnPattern)
oResult = oMeta.getColumnPrivileges(oNull, "%", "%", "%")
AddResultSetToDoc(oResult, oDoc, "getColumnPrivileges")

REM The NULL reference can not be replaced with
REM "%" as with getColumnPrivileges().
oResult = oMeta.getColumns(oNull, "%", "%", "%")
ResultSetToData(oResult, oDoc, "getColumns", oData1())
SafeArrayColumns(11, oData1(), _
    Array("NO_NULLS", "NULLABLE", "NULLABLE_UNKNOWN"), True)
AppendDataToCalcDoc(oDoc, oData1())

RunMultipleCallsAppendToDoc(oDoc, oMeta, _
    Array("getDatabaseProductName", "getDatabaseProductVersion"))

oData2() = Array("getDefaultTransactionIsolation", "unknown")
Select Case oMeta.getDefaultTransactionIsolation()
Case com.sun.star.sdbc.TransactionIsolation.NONE
    oData2(1) = "0 = None"
Case com.sun.star.sdbc.TransactionIsolation.READ_UNCOMMITTED
    oData2(1) = "1 = Read Uncommitted"
Case com.sun.star.sdbc.TransactionIsolation.READ_COMMITTED
    oData2(1) = "2 = Read Committed"
Case com.sun.star.sdbc.TransactionIsolation.REPEATABLE_READ
    oData2(1) = "4 = Repeatable Read"
Case com.sun.star.sdbc.TransactionIsolation.SERIALIZABLE
    oData2(1) = "8 = Serializable"
Case Else
    oData2(1) = "invalid"
End Select
AppendDataToCalcDoc(oDoc, Array(oData2()))

RunMultipleCallsAppendToDoc(oDoc, oMeta, _
    Array("getDriverMajorVersion", "getDriverMinorVersion", _
        "getDriverName", "getDriverVersion"))

ReDim oData3(5 To 7)
oData3(5) = "INITIALLY_DEFERRED"
oData3(6) = "INITIALLY_IMMEDIATE"
oData3(7) = "NONE"

For i = LBound(oTables()) To UBound(oTables())

```

```

oResult = oMeta.getExportedKeys(oNull, "%", oTables(i))
ResultSetToData(oResult, oDoc, _
    "getExportedKeys(" & oTables(i) & ")", _
    oData1())
SafeArrayColumns(10, oData1(), Array("CASCADE", "RESTRICT", _
    "SET_NULL", "NO_ACTION", _
    "SET_DEFAULT"), True)
SafeArrayColumns(11, oData1(), Array("CASCADE", "RESTRICT", _
    "SET_NULL", "NO_ACTION", _
    "SET_DEFAULT"), True)

SafeArrayColumns(14, oData1(), oData3(), True)
AppendDataToCalcDoc(oDoc, oData1())
Next

For i = LBound(oTables()) To UBound(oTables())
    oResult = oMeta.getIndexInfo(oNull, "%", oTables(i), False, True)
    ResultSetToData(oResult, oDoc, "getIndexInfo(" & oTables(i) & ")", _
        oData1())

    SafeArrayColumns(7, oData1(), _
        Array("STATISTIC", "CLUSTERED", "HASHED", "OTHER"), True)
    AppendDataToCalcDoc(oDoc, oData1())
Next

RunMultipleCallsAppendToDoc(oDoc, oMeta, _
    Array("getMaxBinaryLiteralLength", "getMaxCatalogNameLength", _
    "getMaxCharLiteralLength", "getMaxColumnNameLength", _
    "getMaxColumnsInGroupBy", "getMaxColumnsInIndex", _
    "getMaxColumnsInOrderBy", "getMaxColumnsInSelect", _
    "getMaxColumnsInTable", "getMaxConnections", _
    "getMaxCursorNameLength", _
    "getMaxIndexLength", "getMaxProcedureNameLength", "getMaxRowSize", _
    "getMaxSchemaNameLength", "getMaxStatementLength", _
    "getMaxStatements", _
    "getMaxTableNameLength", "getMaxTablesInSelect", _
    "getMaxUserNameLength"))

AppendDataToCalcDoc(oDoc, _
    Array(Split("getNumericFunctions," & _
        oMeta.getNumericFunctions(), ",")))

For i = LBound(oTables()) To UBound(oTables())
    oResult = oMeta.getPrimaryKeys(oNull, "%", oTables(i))
    AddResultSetToDoc(oResult, oDoc, "getPrimaryKeys(" & _
        oTables(i) & ")")
Next

RunMultipleCallsAppendToDoc(oDoc, oMeta, Array("getProcedureTerm"))

oResult = oMeta.getSchemas()
AddResultSetToDoc(oResult, oDoc, "getSchemas")

```

```

RunMultipleCallsAppendToDoc(oDoc, oMeta, _
    Array("getSchemaTerm", "getSearchStringEscape"))
AppendDataToCalcDoc(oDoc, _
    Array(Split("getSQLKeywords," & oMeta.getSQLKeywords(), ",")))
AppendDataToCalcDoc(oDoc, _
    Array(Split("getStringFunctions," & oMeta.getStringFunctions(), _
        ",")))
AppendDataToCalcDoc(oDoc, _
    Array(Split("getSystemFunctions," & _
        oMeta.getSystemFunctions(), ",")))

REM Obtain the following columns:
REM TABLE_CAT, TABLE_SCHEM, TABLE_NAME, GRANTOR, GRANTEE, PRIVILEGE
oResult = oMeta.getTablePrivileges(oNull, "%", "%")
AddResultSetToDoc(oResult, oDoc, "getTablePrivileges")

oResult = oMeta.getTables(oNull, "%", "%", Array())
AddResultSetToDoc(oResult, oDoc, "getTables")

REM One column with the supported table types
oResult = oMeta.getTableTypes(oNull, "%", "%")
AddResultSetToDoc(oResult, oDoc, "getTableTypes")

AppendDataToCalcDoc(oDoc, _
    Array(Split("getTimeDateFunctions," & _
        oMeta.getTimeDateFunctions(), ",")))

oResult = oMeta.getTypeInfo()
ResultSetToData(oResult, oDoc, "getTypeInfo", oData1())
SafeArrayColumns(7, oData1(), _
    Array("NO_NULLS", "NULLABLE", "NULLABLE_UNKNOWN"), True)
SafeArrayColumns(9, oData1(), _
    Array("NONE", "CHAR", "BASIC", "FULL"), True)
AppendDataToCalcDoc(oDoc, oData1())

REM The outlook driver generates an exception with a call to getUDTS()
'oResult = oMeta.getUDTS(oNull, "%", "%", _
'    Array(com.sun.star.sdbc.DataType.OBJECT, _
'          com.sun.star.sdbc.DataType.STRUCT , _
'          com.sun.star.sdbc.DataType.DISTINCT ))
'AddResultSetToDoc(oResult, oDoc, "getUDTS")

RunMultipleCallsAppendToDoc(oDoc, oMeta, _
    Array("getURL", "getUserName"))

For i = LBound(oTables()) To UBound(oTables())
    oResult = oMeta.getVersionColumns(oNull, "%", oTables(i))
    ResultSetToData(oResult, oDoc, "getVersionColumns(" & _
        oTables(i) & ")", _
        oData1())
    SafeArrayColumns(8, oData1(), _
        Array("UNKNOWN", "NOT_PSEUDO", "PSEUDO"), True)

```

```

AppendDataToCalcDoc(oDoc, oData1())
Next

RunMultipleCallsAppendToDoc(oDoc, oMeta, _
    Array("isCatalogAtStart", "isReadOnly", "nullPlusNonNullIsNull", _
        "nullsAreSortedAtEnd", "nullsAreSortedAtStart", _
        "nullsAreSortedHigh", "nullsAreSortedLow"))

For i = 1003 To 1005
    RunMultipleCallsAppendToDoc(oDoc, oMeta, _
        Array("insertsAreDetected", _
            "othersDeletesAreVisible", "othersInsertsAreVisible", _
            "othersUpdatesAreVisible", "ownDeletesAreVisible", _
            "ownInsertsAreVisible", "ownUpdatesAreVisible"), True, i)
Next

RunMultipleCallsAppendToDoc(oDoc, oMeta, _
    Array("storesLowerCaseIdentifiers", _
        "storesLowerCaseQuotedIdentifiers", _
        "storesMixedCaseIdentifiers", _
        "storesMixedCaseQuotedIdentifiers", _
        "storesUpperCaseIdentifiers", _
        "storesUpperCaseQuotedIdentifiers", _
        "supportsAlterTableWithAddColumn", _
        "supportsAlterTableWithDropColumn", _
        "supportsANSI92EntryLevelSQL", "supportsANSI92FullSQL", _
        "supportsANSI92IntermediateSQL", "supportsBatchUpdates", _
        "supportsCatalogsInDataManipulation", _
        "supportsCatalogsInIndexDefinitions", _
        "supportsCatalogsInPrivilegeDefinitions", _
        "supportsCatalogsInProcedureCalls", _
        "supportsCatalogsInTableDefinitions", "supportsColumnAliasing"))

oData1() = Array(-7, -6, -5, -5, -4, -3, -2, -1, 0, _
    1, 2, 3, 4, 6, 7, 8, 12, 91, 92, 93, 111, _
    2000, 2001, 2002, 2003, 2004, 2005, 2006)

oData2() = Array("supportsConvert", "BIT", "TINYINT", "BIGINT", _
    "LONGVARBINARY", "VARBINARY", "BINARY", "LONGVARCHAR", _
    "SQLNULL", "CHAR", "NUMERIC", "DECIMAL", "INTEGER", _
    "SMALLINT", "FLOAT", "REAL", "DOUBLE", "VARCHAR", "DATE", _
    "TIME", "TIMESTAMP", "OTHER", "OBJECT", "DISTINCT", _
    "STRUCT", "ARRAY", "BLOB", "CLOB", "REF")

oData3() = DimArray(UBound(oData2()))
oData3(0) = oData2()
For i = LBound(oData1()) to UBound(oData1())
    oData4() = DimArray(UBound(oData2()))
    oData4(0) = "supportsConvert(" & oData2(i+1) & ", ...)"
    For j = LBound(oData1()) to UBound(oData1())
        oData4(j+1) = CStr(oMeta.supportsConvert(oData1(i), oData1(j)))
    Next
Next

```

```

    oData3(i+1) = oData4()
Next
AppendDataToCalcDoc(oDoc, oData3())

RunMultipleCallsAppendToDoc(oDoc, oMeta, _
    Array( "supportsCoreSQLGrammar", "supportsCorrelatedSubqueries", _
        "supportsDataDefinitionAndDataManipulationTransactions", _
        "supportsDataManipulationTransactionsOnly", _
        "supportsDifferentTableCorrelationNames", _
        "supportsExpressionsInOrderBy", "supportsExtendedSQLGrammar", _
        "supportsFullOuterJoins", "supportsGroupBy", _
        "supportsGroupByBeyondSelect", "supportsGroupByUnrelated", _
        "supportsIntegrityEnhancementFacility", _
        "supportsLikeEscapeClause", _
        "supportsLimitedOuterJoins", "supportsMinimumSQLGrammar", _
        "supportsMixedCaseIdentifiers", _
        "supportsMixedCaseQuotedIdentifiers", _
        "supportsMultipleResultSets", "supportsMultipleTransactions", _
        "supportsNonNullableColumns", _
        "supportsOpenCursorsAcrossCommit", _
        "supportsOpenCursorsAcrossRollback", _
        "supportsOpenStatementsAcrossCommit", _
        "supportsOpenStatementsAcrossRollback", _
        "supportsOrderByUnrelated", _
        "supportsOuterJoins", "supportsPositionedDelete", _
        "supportsPositionedUpdate"))

For i = 1003 To 1005
    RunMultipleCallsAppendToDoc(oDoc, oMeta, _
        Array( "supportsResultSetConcurrency"), True, i, 1007)
    RunMultipleCallsAppendToDoc(oDoc, oMeta, _
        Array( "supportsResultSetConcurrency"), True, i, 1008)
Next

For i = 1003 To 1005
    RunMultipleCallsAppendToDoc(oDoc, oMeta, _
        Array( "supportsResultSetType"), True, i)
Next

RunMultipleCallsAppendToDoc(oDoc, oMeta, _
    Array( "supportsSchemasInDataManipulation", _
        "supportsSchemasInIndexDefinitions", _
        "supportsSchemasInPrivilegeDefinitions", _
        "supportsSchemasInProcedureCalls", _
        "supportsSchemasInTableDefinitions", "supportsSelectForUpdate", _
        "supportsStoredProcedures", "supportsSubqueriesInComparisons", _
        "supportsSubqueriesInExists", "supportsSubqueriesInIns", _
        "supportsSubqueriesInQuantifieds", _
        "supportsTableCorrelationNames"))

REM com.sun.star.sdbc.TransactionIsolation constants
oData1() = Array(0, 1, 2, 4, 8)

```

```

For i = Lbound(oData1()) to UBound(oData1())
    RunMultipleCallsAppendToDoc(oDoc, oMeta, _
        Array("supportsTransactionIsolationLevel"), True, oData1(i))
Next

RunMultipleCallsAppendToDoc(oDoc, oMeta, _
    Array("supportsTransactions", "supportsTypeConversion", _
        "supportsUnion", "supportsUnionAll"))

For i = 1003 To 1005
    RunMultipleCallsAppendToDoc(oDoc, oMeta, _
        Array("updatesAreDetected"), True, i)
Next

RunMultipleCallsAppendToDoc(oDoc, oMeta, _
    Array("usesLocalFilePerTable", "usesLocalFiles"))
InspectMetaData() = oDoc
End Function

```

The macro in *Listing 36* demonstrates how to inspect the meta-data from a connection.

DB Meta Data Use the **DB Meta Data** button to select a Base document and display the associated meta data in a Calc document.

Listing 36. InspectHSQLMetaData is contained in SC04.

```

Sub InspectDBMetaData()
    Dim oBaseContext 'Global database context service.
    Dim oDataBase     'Database obtained from the database context.
    Dim oCon          'Database connection.
    Dim sURL$

    LoadDBLibs()
    sURL = ChooseAFile$(OOoBaseFilters(), True)
    If sURL = "" Then Exit Sub

    oBaseContext = CreateUnoService("com.sun.star.sdb.DatabaseContext")
    oDataBase = oBaseContext.getByName(sURL)
    oCon = oDataBase.getConnection("", "")
    InspectMetaData(oCon.getMetaData())
    oCon.close()
End Sub

```

The macro in *Listing 37* demonstrates how to inspect the meta-data from a connection.

HSQLDB Meta Data Use the **HSQLDB Meta Data** button to use the binary database document created at the beginning of this document. This macro may take a little while to run!

Listing 37. InspectHSQLMetaData is contained in SC04.

```

Sub InspectHSQLMetaData()
    Dim oBaseContext 'Global database context service.
    Dim oDataBase     'Database obtained from the database context.
    Dim oCon          'Database connection.
    Dim sDBUrl$
    LoadDBLibs()

```

```

sDBUrl = GetSourceCodeDir() & sDBBaseName
oBaseContext = CreateUnoService("com.sun.star.sdb.DatabaseContext")
oDataBase = oBaseContext.getByNamed(sDBUrl)
oCon = oDataBase.getConnection("", "")
InspectMetaData(oCon.getMetaData())
oCon.close()
End Sub

```

8.4.4. GetBestRowIdentifier

The method `getBestRowIdentifier()` returns a description of the optimal set of columns that uniquely identify a row. In other words, use the columns identified by the method `getBestRowIdentifier()` to most quickly access a row in a table. The arguments are described in *Table 21*.

Table 21. Arguments for `getBestRowIdentifier()`.

Argument	Description
catalog	Retrieve rows that use the specified catalog name; an empty string ("") retrieves rows without a catalog and NULL means ignore the catalog name.
schema	Retrieve rows that use the specified schema name; an empty string ("") retrieves rows without a schema.
table	Retrieve rows that use the specified table name; wild-cards are not supported.
scope	Retrieve rows with the specified scope. Use the <code>com.sun.star.sdbc.BestRowScope</code> constant group, which supports values of TEMPORARY (0), TRANSACTION (1), and SESSION (2).
nullable	If True, return columns that are allowed to be null.

Many databases support special “hidden” columns that are not explicitly defined by the user in the table definition. These pseudo-columns generally provide the fastest access to the data because they typically are pointers to the exact location of the record. *Table 22* contains the constants used to identify a column as a pseudo-column.

Table 22. Values for the `com.sun.star.sdbc.BestRowType` constant group.

Value	Name	Description
0	UNKNOWN	The best row identifier may or may not be a pseudo-column.
1	NOT_PSEUDO	The best row identifier is not a pseudo-column.
2	PSEUDO	The best row identifier is a pseudo-column.

The `getBestRowIdentifier()` method returns a result set with the columns shown in *Table 23*.

Table 23. Columns returned by `getBestRowIdentifier()`.

#	Column	Type	Description
1	SCOPE	short	Scope, based on the <code>BestRowScope</code> constant group (see the scope argument in <i>Table 21</i>).
2	COLUMN_NAME	string	Column name.

#	Column	Type	Description
3	DATA_TYPE	short	SQL data type based on the java.sql.Types (see <i>Table 10</i>).
4	TYPE_NAME	string	Data source dependent type name.
5	COLUMN_SIZE	long	The precision (length) of the column.
6	BUFFER_LENGTH	long	Length of column value in bytes.
7	DECIMAL_DIGITS	short	Scale; for numerical columns.
8	PSEUDO_COLUMN	short	Identify the column as a pseudo-column (see <i>Table 22</i>).

Listing 35 calls `getBestRowIdentifier()`. Some drivers return a NULL result set rather than a result set that contains no rows. In my testing using OOo 2.0 pre-beta(?? retest), a flat file returned a result set that contained no rows and an HSQLDB database returned NULL.

8.4.5. GetColumnPrivileges

The result set from `getColumnPrivileges()` describes the access rights for a table's columns; the supported arguments are shown in *Table 24*.

Table 24. Arguments for getColumnPrivileges().

Argument	Description
catalog	Retrieve rows that use the specified catalog name; "" retrieves rows without a catalog and NULL means ignore the catalog name.
schema	Retrieve rows that use the specified schema name; "" retrieves rows without a schema.
table	Retrieve rows that use the specified table name; use "%" to retrieve all tables.
column	Retrieve rows that use the specified column name; use "%" to retrieve all columns.

The method `getColumnPrivileges()` returns a result set with the columns as shown in *Table 25*. System tables are also returned, so hundreds of rows may be returned, even on a small database.

Table 25. Columns returned by getColumnPrivileges().

#	Column	Type	Description
1	TABLE_CAT	string	Table catalog name (may be NULL).
2	TABLE_SCHEM	string	Table schema (may be NULL).
3	TABLE_NAME	string	Table name.
4	COLUMN_NAME	string	Column name.
5	GRANTOR	string	Granter of access (may be NULL).
6	GRANTEE	string	Grantee of access.
7	PRIVILEGE	string	Name of access (SELECT, INSERT, UPDATE, REFERENCES, ...).
8	IS_GRANTABLE	string	"YES" if grantee is permitted to grant to others; "NO" if not; NULL if unknown.

8.4.6. GetColumns

The `getColumns()` method returns a description of the available columns. The arguments to `getColumns()`, catalog, schema, table, and column, are shown in *Table 24*. The returned row set uses the `ColumnValue` constant group (see *Table 26*) to identify if a column can contain a NULL value.

Table 26. Values for the com.sun.star.sdbc.ColumnValue constant group

Value	Name	Description
0	NO_NULLS	A column does not allow NULL values.
1	NULLABLE	A column allows NULL values.
2	NULLABLE_UNKNOWN	The null-ability of the column is unknown.

The `getColumns()` method returns a result set with the columns shown in *Table 27*.

Table 27. Columns returned by getColumns().

#	Column	Type	Description
1	TABLE_CAT	string	Table catalog name.
2	TABLE_SCHEM	string	Table schema name.
3	TABLE_NAME	string	Table name.
4	COLUMN_NAME	string	Column name.
5	DATA_TYPE	short	SQL data type based on the java.sql.Types (see <i>Table 10</i>).
6	TYPE_NAME	string	Data source dependent type name.
7	COLUMN_SIZE	long	The precision (length) of the column.
8	BUFFER_LENGTH	long	Length of column value in bytes. Not used for all drivers.
9	DECIMAL_DIGITS	long	Scale for numerical columns.
10	NUM_PREC_RADIX	long	Radix (typically either 10 or 2).
11	NULLABLE	long	Can the column contain a NULL value (see <i>Table 26</i>).
12	REMARKS	string	Comment describing column (may be NULL).
13	COLUMN_DEF	string	Default value for the column (may be NULL).
14	SQL_DATA_TYPE	long	The value of the SQL data type as it appears in the SQL_DESC_TYPE field of the descriptor. This is frequently the same as the data type and it may be NULL.
15	SQL_DATETIME_SUB	long	For a date-time or interval type, this contains the date-time/interval sub-code. This may be NULL.
16	CHAR_OCTET_LENGTH	long	Maximum number of bytes in the column (for char columns).
17	ORDINAL_POSITION	int	Column's Index in the table (starting with 1).
18	IS_NULLABLE	string	“NO” means no, “YES” means maybe, and “”

#	Column	Type	Description
			means unknown.

Most database systems store information in system tables. For example, it is typical to have a table that lists users, and another that lists the defined tables. When run against a simple database using the HSQLDB format, the `getColumns()` method returned roughly 400 rows from numerous system tables. Use `getColumns()` to investigate the available columns and rows, but in practice, you will probably want to limit the returned values to the tables of interest.

8.4.7. GetExportedKeys

The method `getExportedKeys()` returns a result set describing the foreign key columns that reference a table's primary key columns (the foreign keys exported by a table). Each foreign key contains an update and delete rule described by the `KeyRule` constant group (see *Table 28*). The behavior may be different based on whether the rule is used as an update rule or a delete rule.

Table 28. Values for the com.sun.star.sdbc.KeyRule constant group.

Value	Name	Description
0	CASCADE	As an update rule, when the primary key is updated, the foreign key (imported key) is also changed. As a delete rule, when the primary key is deleted, rows that imported that key are deleted.
1	RESTRICT	As an update rule, a primary key may not be updated if it is imported by another table as a foreign key. As a delete rule, a primary key may not be deleted if it is imported by another table as a foreign key.
2	SET_NULL	If the primary key is updated or deleted, the foreign key (imported key) is changed to NULL.
3	NO_ACTION	An imported primary key cannot be updated or deleted.
4	SET_DEFAULT	If the primary key is updated or deleted, the foreign key (imported key) is set to the default value.

The Deferrability constant group (see *Table 29*) controls whether a constraint can be deferred. A constraint that is not deferrable is checked immediately after every command. A deferred constraint may be postponed until the end of the transaction.

Table 29. Values for the com.sun.star.sdbc.Deferrability constant group.

Value	Name	Description
5	INITIALLY_DEFERRED	Foreign key verification may be deferred.
6	INITIALLY_IMMEDIATE	Foreign key verification is done after each command.
7	NONE	Foreign key verification is not performed.

The columns returned by the `getExportedKeys()` method are shown in *Table 30*.

Table 30. Columns returned by *getExportedKeys()*.

#	Column	Type	Description
1	PKTABLE_CAT	string	Primary key table catalog (may be NULL).
2	PKTABLE_SCHEM	string	Primary key table schema (may be NULL).
3	PKTABLE_NAME	string	Primary key table name.
4	PKCOLUMN_NAME	string	Primary key column name.
5	FKTABLE_CAT	string	Foreign key table catalog being exported (may be NULL).
6	FKTABLE_SCHEM	string	Foreign key table schema being exported (may be NULL).
7	FKTABLE_NAME	string	Foreign key table name being exported.
8	FKCOLUMN_NAME	string	Foreign key column name being exported.
9	KEY_SEQ	short	Sequence number in the foreign key.
10	UPDATE_RULE	short	What happens to the foreign key when the primary is updated (see KeyRule in Table 28).
11	DELETE_RULE	short	What happens to the foreign key when the primary is deleted (see KeyRule in Table 28).
12	FK_NAME	string	Foreign key name (may be NULL).
13	PK_NAME	string	Primary key name (may be NULL).
14	DEFERRABILITY	short	When are foreign key constraints evaluated (see Table 29)?

The arguments for *getExportedKeys()* are shown in Table 31.

Table 31. Arguments for *getExportedKeys()*.

Argument	Description
catalog	Retrieve rows that use the specified catalog name; "" retrieves rows without a catalog and NULL means ignore the catalog name.
schema	Retrieve rows that use the specified schema name; "" retrieves rows without a schema.
table	Retrieve rows that use the specified table name; wild-cards are not supported.

8.4.8. GetIndexInfo

The *getIndexInfo()* method returns a row set that provides information related to each index for the database. Each index has a type (see Table 32). A clustered index is usually used when there are a limited number of unique values, such as a state column that contains only 50 unique state codes. A clustered index is also frequently used if queries using operators such as BETWEEN, >, >=, <, and <= are used. A hashed based index is optimized for equality based searches.

Table 32. Values for the *com.sun.star.sdbc.IndexType* constant group.

Value	Name	Description
0	STATISTIC	Table statistics that are returned in conjunction with a table's index

Value	Name	Description
		description.
1	CLUSTERED	The index is a clustered index.
2	HASHED	The index is a hashed index.
3	OTHER	This index some other type.

All of the entries in *Table 32* define a type of index except for STATISTIC, which changes the meaning of certain rows. For example, the PAGES column refers to the number of pages used for the index unless the type is STATISTIC, in which case it is the number of pages used for the table (see *Table 33*).

Table 33. Columns returned by getIndexInfo().

#	Column	Type	Description
1	TABLE_CAT	string	Table catalog (may be NULL).
2	TABLE_SCHEM	string	Table schema (may be NULL).
3	TABLE_NAME	string	Table name.
4	NON_UNIQUE	boolean	True, if the index values can be non-unique, False otherwise; this is always False for the statistic type.
5	INDEX_QUALIFIER	string	Index catalog (may be NULL); NULL for type statistic.
6	INDEX_NAME	string	Index name. This is always NULL when the type is statistic.
7	TYPE	short	Index type (see <i>Table 32</i>).
8	ORDINAL_POSITION	short	Column sequence number in the index; zero for type statistic.
9	COLUMN_NAME	string	Column name; NULL for type statistic.
10	ASC_OR_DESC	string	Column sort sequence; “A” means ascending and “D” means descending. This is NULL if a sort sequence is not supported and for type statistic.
11	CARDINALITY	long	Number of unique values in the index. If the type is statistic, then this is the number of rows in the table.
12	PAGES	long	Number of pages used for the index. If the type is statistic, then this is the number of pages used for the table.
13	FILTER_CONDITION	string	Filter condition, if any (may be NULL).

The arguments for getIndexInfo() are shown in *Table 34*.

Table 34. Arguments for getIndexInfo().

Argument	Description
catalog	Retrieve rows that use the specified catalog name; "" retrieves rows without a catalog and NULL means ignore the catalog name.
schema	Retrieve rows that use the specified schema name; "" retrieves rows without a schema.

Argument	Description
table	Retrieve rows that use the specified table name; wild-cards are not supported.
unique	If True, return only indexes for unique values, otherwise, return all indexes.
approximate	If True, the returned values may contain approximate or out of data values. If False, results must be accurate.

8.4.9. GetPrimaryKeys

The `getPrimaryKeys()` method returns a result set that describes a table's primary key columns. The returned rows are ordered by the column name. The accepted arguments—catalog, schema, and table—are described in *Table 26* and the returned rows are shown in *Table 35*.

Table 35. Columns returned by `getPrimaryKeys()`.

#	Column	Type	Description
1	TABLE_CAT	string	Catalog name (may be NULL).
2	TABLE_SCHEM	string	Table schema (may be NULL).
3	TABLE_NAME	string	Table name.
4	COLUMN_NAME	string	Column name.
5	KEY_SEQ	short	Sequence number within primary key.
6	PK_NAME	string	Name of the primary key (may be NULL).

8.4.10. GetTablePrivileges

The method `getTablePrivileges()` returns a result set that describes the access rights for each table in the catalog. Each table privilege applies to one or more columns in the table, but not necessarily to all of the columns in the table. The accepted arguments are described in *Table 26* and the returned columns are described in *Table 36*—wild-cards are supported for the table name.

Table 36. Columns returned by `getTablePrivileges()`.

#	Column	Type	Description
1	TABLE_CAT	string	Catalog name (may be NULL).
2	TABLE_SCHEM	string	Table schema (may be NULL).
3	TABLE_NAME	string	Table name.
4	GRANTOR	string	Person who granted access (may be NULL).
5	GRANTEE	string	Person to whom access was granted.
6	PRIVILEGE	string	Access type (SELECT, INSERT, UPDATE, REFERENCES, ...).
7	IS_GRANTABLE	string	"YES" if grantee is permitted to grant to others; "NO" if not; NULL if unknown

8.4.11. GetTables

The `getTables()` method returns a row set that describes the tables contained in the database. The columns returned by the row set are described in *Table 37*.

Table 37. Columns returned by `getTables()`.

#	Column	Type	Description
1	TABLE_CAT	string	Catalog name (may be NULL).
2	TABLE_SCHEM	string	Table schema (may be NULL).
3	TABLE_NAME	string	Table name.
4	TABLE_TYPE	string	Table type; typical values include TABLE, VIEW, SYSTEM TABLE, GLOBAL TEMPORARY, LOCAL TEMPORARY, ALIAS, and SYNONYM.
5	REMARKS	string	Comment that explains the table.

The arguments for `getTables()` are shown in *Table 38*.

Table 38. Arguments for `getTables()`.

Argument	Description
catalog	Retrieve rows that use the specified catalog name; "" retrieves rows without a catalog and NULL means ignore the catalog name.
schema	Retrieve rows that use the specified schema name; "" retrieves rows without a schema.
table	Retrieve rows that use the specified table name; use "%" to retrieve all tables.
types()	Array of table types to return; Although NULL returns all types, this is not supported on some systems; use an empty array instead.

8.4.12. GetTypeInfo()

The `getTypeInfo()` method returns a row set that describes the standard SQL types supported by this database. Each column supports different types of searches based on the data type and the database implementation (see *Table 10*).

Table 39. Values for the `com.sun.star.sdbc.ColumnSearch` constant group.

Value	Name	Description
0	NONE	WHERE search clauses are not supported for this type.
1	CHAR	The only supported search clause is WHERE...LIKE.
2	BASIC	Supports all supported search clauses except WHERE...LIKE.
3	FULL	All WHERE search clauses can be based on this type.

The columns returned by the row set are shown in *Table 40*.

Table 40. Columns returned by *getTypeInfo()*.

#	Column	Type	Description
1	TYPE_NAME	string	Type name.
2	DATA_TYPE	short	SQL data type from java.sql.Types (see Table 10).
3	PRECISION	long	Maximum precision.
4	LITERAL_PREFIX	string	Prefix used to quote a literal (may be NULL).
5	LITERAL_SUFFIX	string	Suffix used to quote a literal (may be NULL).
6	CREATE_PARAMS	string	Parameters used in creating the type (may be NULL).
7	NULLABLE	short	Indicates if this type supports NULL values (see Table 26).
8	CASE_SENSITIVE	boolean	True if this type is case sensitive.
9	SEARCHABLE	short	Indicates the searches supported by this type: Table 39.
11	FIXED_PREC_SCALE	boolean	If True, this type can represent a monetary value.
12	AUTO_INCREMENT	boolean	If True, this type can be used for an auto-increment value.
13	LOCAL_TYPE_NAME	string	Localized version of type name (may be NULL).
14	MINIMUM_SCALE	short	Minimum scale supported.
15	MAXIMUM_SCALE	short	Maximum scale supported.
16	SQL_DATA_TYPE	long	The value of the SQL data type as it appears in the SQL_DESC_TYPE field of the descriptor. This is frequently the same as the data type and it may be NULL.
17	SQL_DATETIME_SUB	long	For a date-time or interval type, this contains the date-time/interval sub-code. This may be NULL for many drivers.
18	NUM_PREC_RADIX	long	Radix (typically either 10 or 2).

8.4.13. GetUDTS

The *getUDTS()* method returns a row set that describes the user-defined types contained in the database. The arguments for *getUDTS()* are shown in Table 41. In my testing, calling the *getUDTS()* method causes a runtime error for some connection types; Outlook, for example.

Table 41. Arguments for *getUDTS()*.

Argument	Description
catalog	Retrieve rows that use the specified catalog name; "" retrieves rows without a catalog and NULL means ignore the catalog name.
schema	Retrieve rows that use the specified schema name; "" retrieves rows without a schema.
type name	Retrieve rows that use the specified fully qualified name; use "%" to retrieve all UDTs.
types()	Array of UDT types to return; supported types include strings with values of OBJECT, STRUCT, and DISTINCT.

The columns returned by the row set are shown in *Table 42*.

Table 42. Columns returned by getUDTS().

#	Column	Type	Description
1	TYPE_CAT	string	Type's catalog (may be NULL).
2	TYPE_SCHEM	string	Type's schema (may be NULL).
3	TYPE_NAME	string	Type name.
4	CLASS_NAME	string	Java class name or service name.
5	DATA_TYPE	string	Type value. One of OBJECT, STRUCT, or DISTINCT.
6	REMARKS	string	Explanatory comment on the type.

8.4.14. GetVersionColumns

The method `getVersionColumns()` returns a row set that describes a table's columns that are automatically updated when any value in a row is updated. Many databases support special “hidden” columns that are not explicitly defined by the user in the table definition. These pseudo-columns generally provide the fastest access to the data because they typically are pointers to the exact location of the record. *Table 43* contains the constants used to identify a column as a pseudo-column.

Table 43. Values for the com.sun.star.sdbc.ColumnType constant group.

Value	Name	Description
0	UNKNOWN	The best row identifier may or may not be a pseudo-column.
1	NOT_PSEUDO	The best row identifier is not a pseudo-column.
2	PSEUDO	The best row identifier is a pseudo-column.

The arguments accepted by `getVersionColumns()`—catalog, schema, and table—are described in *Table 31*—wild-cards are not supported for the table argument—and the returned rows are shown in *Table 44*.

Table 44. Columns returned by getVersionColumns().

#	Column	Type	Description
1	SCOPE	short	Unused.
2	COLUMN_NAME	string	Column name.
3	DATA_TYPE	short	SQL data type based on the java.sql.Types (see <i>Table 10</i>).
4	TYPE_NAME	string	Data source dependent type name.
5	COLUMN_SIZE	long	The precision (length) of the column.
6	BUFFER_LENGTH	long	Length of column value in bytes.
7	DECIMAL_DIGITS	short	Scale; for numerical columns.
8	PSEUDO_COLUMN	short	Identify the column as a pseudo-column (see <i>Table 43</i>).

8.5. Connections

A method is usually obtained as a three step process (see *Listing 38*):

1. Get a reference to the DatabaseContext service.
2. Get/load the database from the DatabaseContext.
3. Get a connection from the database.

Listing 38. *Standard method to obtain a database connection.*

```
oBaseContext = CreateUnoService("com.sun.star.sdb.DatabaseContext")
oDB = oBaseContext.getByName(dbURL)
oCon = oDB.getConnection(" ", " ")
```

The `getConnection` method returns a proxy object for the connection. Every proxy object obtained in this way share the same physical connection. Therefore, operations done on one proxy should cause side effects on all the others.

Another method, `getIsolatedConnection`, returns an isolated connection that is not shared. Although an isolated connection uses more resources, this is required if a separate connection is required to the database. OOo is not a multi-user application so a standard connection is usually what you want.

If an existing connection already has exclusive access to the database, the returned connection will be (I think it will be) read-only. For example, if you open two isolated connections, the second connection is likely to be read-only.

TIP If you forget to close a connection, the next connection may be read-only, even if the connection is isolated. So, if you obtain a read-only connection that you did not expect, check your code for a *connection leak*.

8.6. Connections without a data source

Although the new database document type, which corresponds to a data source, is desirable, sometimes the extra functionality of prepared queries, forms, and reports is not required. It is possible to connect to a database by using an appropriate database driver rather than a data source—so an external Base file is not required. OpenOffice.org contains a driver manager that manages the database drivers supported by OOo. *Listing 39* demonstrates how to retrieve the database driver manager.

Listing 39. *Retrieve the global database driver manager.*

```
oManager = CreateUnoService("com.sun.star.sdbc.DriverManager")
```

The driver manager identifies the driver types based on a URL (see *Table 45*). The structure of the URL consists of a protocol name, followed by the driver specific sub-protocol. The data source administration dialog shows the latest supported protocols. Some protocols are platform dependent. For example, ADO is only supported on Windows.

Table 45. Database URL types supported by OpenOffice.org.

Driver	URL	Comment
JDBC	jdbc:subprotocol:	JDBC connections.
ODBC 3.5	sdbc:odbc:datasource name	ODBC connections.
Adabas D	sdbc:adabas:database name	Adabas connections.
ADO	sdbc:ado:ADO specific	Available only on Windows.
dBase	sdbc:dbase:Location of folder or file	Read-only access to dBase files.
Flat file format (csv)	sdbc:flat:Location of folder or file	Read-only access to delimited text files (see <i>Listing 42</i>).
OpenOffice.org Calc	sdbc:calc:Location of OpenOffice.org Calc file	Read-only access to Calc documents (see <i>Listing 47</i>).
Address Book	sdbc:address:Kind of address book.	Mozilla, Outlook, and LDAP address books (see <i>Listing 48</i>).

The driver manager supports methods to obtain connections, based on a URL and to enumerate the supported drivers (see *Table 46*).

Table 46. Some methods supported the driver manager.

Method	Description
createEnumeration()	Return an enumeration of the available drivers.
getConnection(url)	Return a connection to the given database URL. The manager selects an appropriate driver from the registered drivers.
getConnectionWithInfo(url, args())	Return a connection to the given database URL. The manager selects an appropriate driver from the registered drivers.
getDriverByURL(url)	Return a driver that can handle the specified URL.
getLoginTimeout()	Get the maximum time, in seconds, that a driver will wait while attempting to connect to a database.
hasElements()	Return True if there are any available drivers.
setLoginTimeout(seconds)	Set the maximum time, in seconds, that a driver will wait while attempting to connect to a database.

Listing 40 demonstrates how to enumerate the drivers supported by OpenOffice.org and is shown in *Figure 20*; my Linux computer has all of the drivers shown in *Figure 20*, and it also has a driver for evolution. **Supported DB Drivers** Use the **Supported DB Drivers** button to display a list of drivers supported on this computer.

Listing 40. Database drivers supported by OOo.

```

Sub SupportedDBDrivers()
    Dim oManager 'Connection driver manager.
    Dim oEnum    'Enumeration of supported drivers.
    Dim oDriver  'An individual driver.
    Dim s$      'Utility string variable.

```

```

oManager = CreateUnoService("com.sun.star.sdbc.DriverManager")
oEnum = oManager.createEnumeration()
Do While oEnum.hasMoreElements()
    oDriver = oEnum.nextElement()
    s = s & oDriver.getImplementationName() & CHR$(10)
Loop
MsgBox s, 0, "Supported Database Drivers"
End Sub

```

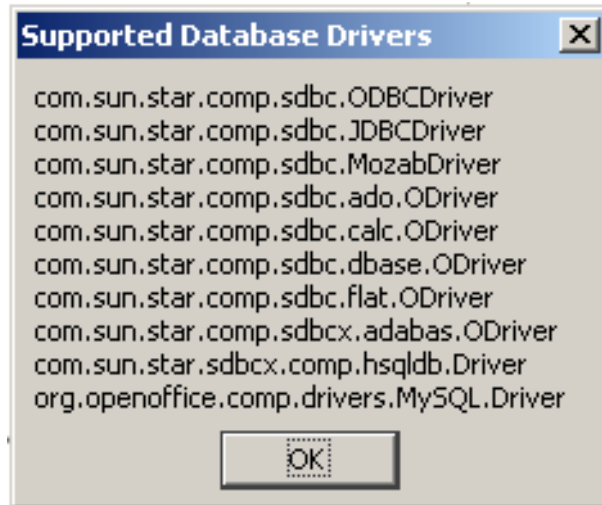


Figure 20. Drivers supported on a Windows computer.

Use `getDriverByURL()` to obtain the driver used to open a specific database. You can also verify that a suitable driver exists for a specific database type. After obtaining a specific driver, use the methods in Table 47.

Table 47. Methods supported by a database driver.

Method	Description
<code>connect(url, args())</code>	Return a connection to the given database URL. NULL is returned if the driver can not connect to the specified database.
<code>acceptsURL(url)</code>	Return True if the driver thinks that it can open a connection to the given URL; drivers return True if they understand the sub-protocol in the URL.
<code>getPropertyInfo(url, args())</code>	Return the properties supported by this driver.
<code>getMajorVersion()</code>	Return the driver's major version number. Initially this should be 1.
<code>getMinorVersion()</code>	Return the driver's minor version number. Initially this should be 0.

The `connect()` method for a driver (see *Table 47*) and `getConnectionWithInfo()` for the driver manager (see *Table 46*) both accept a URL to the database and an array of arguments. The supported arguments differ based on the database and the driver used. Each driver supports the method `getPropertyInfo()` (see *Table 47*) that returns the properties supported by a specific driver. The `getPropertyInfo()` method is intended to allow a generic GUI tool to discover the supported properties and then prompt for connection information. The returned list may change depending on the properties, which is why `getPropertyInfo()` accepts an array of properties. As such, it may be necessary to iterate through several calls to `getPropertyInfo()`.

The macro in *Listing 41* demonstrates the use of `getPropertyInfo()` for a CSV file.

Flat Driver Args Use the **Flat Driver Args** button to display the supported arguments for a CSV file (see *Figure 21*). Notice that the driver displays arguments for the file `foo.csv`, which does not exist. The file name is used to find the driver. The macro in *Listing 41* does not actually open the file.

Listing 41. Show the arguments supported by the CSV database driver.

```
Sub ShowFlatDriverArgs()
    LoadDBLibs()
    Call DriverArgs("sdbc:flat:" & GetSourceCodeDir() & "foo.csv")
End Sub

REM The specified database is not required to exist.
REM Required arguments are enclosed in parenthesis "()".
REM Optional arguments are enclosed in square brackets "["].

Sub DriverArgs(sURL$)
    Dim oManager    'Connection driver manager.
    Dim oDriver     'An individual driver.
    Dim oProps()    'Supported properties.
    Dim oProp       'A specific property.
    Dim s$          'Utility string variable.
    Dim i%          'Utility index variable.

    oManager = CreateUnoService("com.sun.star.sdbc.DriverManager")

    REM Obtain a driver that supports the specified URL
    oDriver = oManager.getDriverByURL(sURL)
    If IsNull(oDriver) Then
        Print "Sorry, no driver available for " & sURL
        Exit Sub
    End If
    oProps() = oDriver.getPropertyInfo(sURL, ARRAY())
    For i = LBound(oProps()) To UBound(oProps())
        oProp = oProps(i)
        If NOT oProp.IsRequired Then
            s = s & "[" & oProp.Name & ", " & oProps(i).Value & ", " & _
                oProps(i).Description & "]" & CHR$(10)
        Else
            s = s & "(" & oProp.Name & ", " & oProps(i).Value & ", " & _
                oProps(i).Description & ")" & CHR$(10)
        End If
    Next
End Sub
```

```
MsgBox s, 0, "Properties for " & oDriver.getImplementationName()  
End Sub
```

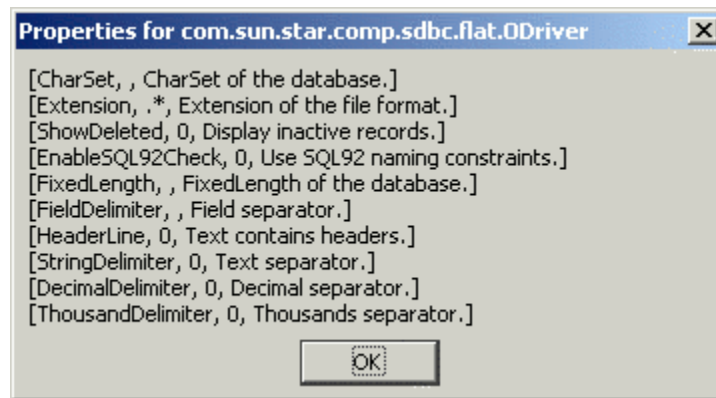


Figure 21. Arguments for a flat file driver.

Figure 21 shows the properties supported by the flat driver for a CSV file as generated by Listing 41. Although all of the listed properties are listed as optional, the default values for some properties are not reasonable. For example, the text separator and field separator are not set. The supported properties for the database drivers are shown in Table 48.

Table 48. Documented connection properties.

Property	Description	Driver
AutoRetrievingStatement	Specifies the statement to executed when asking an insert statement for the XGeneratedResultSet interface.	JDBC, ODBC
Charset	Character set used to fetch data.	ODBC, dBase, Flat
DecimalDelimiter	A one character delimiter to separate the decimal.	Flat
Extension	Extension of the files to be used; txt, csv, and sdf.	Flat
FieldDelimiter	A one character delimiter to separate the fields.	Flat
FixedLength	True when all occurrences of "?" as a parameter name will be replaced by a valid parameter name. This is necessary, because some drivers mix the order of the parameters.	Flat
HeaderLine	True when the file contains a header line otherwise false	Flat
IgnoreDriverPrivileges	Ignore privileges from the database driver.	ODBC JDBC
IsAutoRetrievingEnabled	If True, the statement will support the XGeneratedResultSet interface when it is supported in the future.	JDBC, ODBC
JavaDriverClass	JAVA class for the JDBC driver.	JDBC
ParameterNameSubstitution	If True, the parameter "?" in a prepared statement is replaced with a distinct name.	ODBC
password	Password for the specified user name.	all
ShowDeleted	True when deleted rows should be shown, otherwise false	dBase
Silent	If True, no user interaction will be used while creating the connection.	ODBC
StringDelimiter	A one character delimiter to separate the strings.	Flat

Property	Description	Driver
SystemDriverSettings	Driver settings.	ODBC
ThousandDelimiter	A one character delimiter to separate the thousands.	Flat
Timeout	Timeout value, in seconds, to use for this connection.	ODBC
UseCatalog	If True, the driver should support a catalog.	ODBC
user	User name to access the database.	all

I inspected the source code and found properties that are not in *Table 48*:

AddIndexAppendix	DataCacheSize	PortNumber
AppendTableAliasName	DataCacheSizeIncrement	ParameterNameSubstitution
AutoIncrementCreation	EnableSQL92Check	ShutdownDatabase
BaseDN	FixedLength	SuppressVersionColumns
BooleanComparisonMode	HostName	UseSchemaInSelect
ControlPassword	MaxRowCount	UseCatalogInSelect
ControlUser	NoNameLengthLimit	

8.6.1. Delimited text files

The flat file driver provides read-only connections to text files. The most popular text file format is the CSV (Comma-Separated Values) file. A CSV file contains the values for a single table as a series of ASCII text lines organized so that each column value is separated by a delimiter—usually a comma, semicolon, or tab—from the next column's value and each row starts a new line. The macro in *Listing 42* opens a connections to a CSV file, inspects the connection's meta-data, and then reads all of the data in the file.

Listing 42. Read a CSV file, print the meta-data and the data.

```
REM sDir - Path, as a URL, to the database file.
REM sFile - File name without the file extension; also the table name.
REM sExt - File extension, probably csv.
Sub ReadCSVData(sDir$, sFile$, sExt$)
    Dim oManager 'Connection driver manager.
    Dim oDriver 'An individual driver.
    Dim sURL$ 'DB URL including "sdbc:flat:..."
    Dim oCon 'Connection object.
    Dim sSQL$ 'SQL that is executed.
    Dim oResult 'Result from an SQL statement.
    Dim oStatement 'A created statement that can execute SQL.
    Dim oDoc 'Calc document that will contain the presentation data.
    Dim oData1() 'Generic data variable.
    Dim oParms() As New com.sun.star.beans.PropertyValue

    sFile = sFile
    sExt = sExt
    sURL = "sdbc:flat:" & sDir & sFile & "." & sExt

    REM Obtain a driver that supports the specified URL
    oManager = CreateUnoService("com.sun.star.sdbc.DriverManager")
    oDriver = oManager.getDriverByURL(sURL)
```

```

If IsNull(oDriver) Then
    Print "Sorry, no driver available for a " & sExt & " file"
    Exit Sub
End If

REM Assume that there is a header line!
AppendProperty(oParms(), "Extension", sExt)
AppendProperty(oParms(), "HeaderLine", True)
AppendProperty(oParms(), "FieldDelimiter", ",")
AppendProperty(oParms(), "StringDelimiter", "\"")
AppendProperty(oParms(), "DecimalDelimiter", ".")
AppendProperty(oParms(), "ThousandDelimiter", ",")

oCon = oManager.getConnectionWithInfo(sURL, oParms())
oDoc = InspectMetaData(oCon.getMetaData())

oStatement = oCon.createStatement()
sSQL = "SELECT * FROM " & DBQuoteName(sFile, oCon)
oResult = oStatement.executeQuery(sSQL)
ResultSetToData(oResult, oDoc, "SELECT *", oData1())
AppendDataToCalcDoc(oDoc, oData1())
oCon.close()
End Sub

```

Read CSV Data Use the **Read CSV Data** button to call the macro in *Listing 43*. First, you must select an existing CSV file. The macro will open a new Calc document and then fill the Calc document with properties from the meta data, followed by the data in the CSV file.

Listing 43. Prompt for a CSV file, and then display the data.

```

Sub CallReadCSVData()
    Dim oFilters()
    Dim sFile$
    Dim sURL$
    Dim sExt$
    Dim sDir$

    REM Load my libraries.y
    LoadDBLibs()

    REM Use some methods from the Tools library.
    If NOT GlobalScope.BasicLibraries.isLibraryLoaded("Tools") Then
        GlobalScope.BasicLibraries.LoadLibrary("Tools")
    End If

    REM List .csv first so it will be used by default.
    oFilters() = Array("*.csv Flat files", "*.csv", "All Files", "**.*")
    sURL = ChooseAFile$(oFilters(), True)
    If sURL = "" Then Exit Sub

    REM Call methods in the Tools library to parse the path.
    sExt = GetFileNameExtension(sURL)
    sFile = GetFileNameWithoutExtension(sURL, "/" )

```



```
sDir = DirectoryNameoutofPath(sURL, "/" ) & "/"

REM Now, display the meta-data and the regular data.
ReadCSVData(sDir$, sFile$, sExt$)
End Sub
```

The database driver may choose to convert table and column names to upper case and then fail because the names are not in mixed or lower-case. Quote the table and column names to avoid this problem. The quote string may be different for each driver. The macro in *Listing 77* uses the quote string from the connection meta-data to safely quote identifiers.

Each flat file contains a single table. The name of the table is the name of the file without the file extension. If the HeaderLine property is True, the column names are set based on the first row of data. If the HeaderLine property is False, however, the columns names default to the letter C followed by the column number; C1, C2, C3, C4.

TIP The table name in a flat file is the base file name. If the first row does not contain the column names, then the columns default to C1, C2, C3. It is always possible to retrieve columns based on its index.

I determined this by inspecting the meta data from the connection. This is the same way that I determined the table names in a Calc document. Inspecting the meta data from the connection is a very enlightening experience.

8.6.2. Fixed width text files

Fixed width text files are frequently called SDF (System Data Format). An SDF file is an ASCII text file in which records have a fixed length and end with a carriage return and line feed. Although the SDF file format is mentioned in the developer's guide as a supported format, I have only been able to read fixed text files by using the CSV file extension. Fields are not delimited. Unfortunately, fixed width files can not be opened directly as a database connection.

TIP OOo uses the SDF extension for files output from the localization tools. The localization files are also sometimes labeled GSI; GSI is short for “Gutschmidt invented” after the inventor of the format. The SDF localization files are not fixed width text files.

It is possible to open fixed width data into a Calc document using the “Text - txt - csv (StarCalc)” import/export filter. The Calc document can then be used as the data source. For Calc to recognize a flat file, the file extension must be CSV. *Listing 44* is a code snippet that will import a fixed width text file.

Listing 44. Import a fixed width text file.

```
Dim args(1) as new com.sun.star.beans.PropertyValue
args(0).Name = "FilterName"
args(0).Value = "Text - txt - csv (StarCalc)"
args(1).Name = "FilterOptions"
args(1).Value = "FIX,34,0,1,0/1/10/2/30/10"

sURL = GetSourceCodeDir() & "fix_len.csv"
oDoc = StarDesktop.LoadComponentFromUrl(sURL, "_blank", 0, args())
```

The FilterOptions argument directs the importing and exporting of data through the Calc CSV filter. The format of the argument is shown in *Listing 45*. Each option is separated by a comma. The first option determines if the CSV file is a fixed width file or a delimited file (see *Table 49*).

Listing 45. Filter options for “Text - txt - csv (StarCalc)” filter.

```
field_separator,text_delimiter,character_set,first_line,field_specifier
```

Table 49. Initial arguments for the “Text - txt - csv (StarCalc)” filter.

#	Name	Description
1	field separator	The ASCII value of the field separator. If multiple separators are used, separate them with a slash. For example, 44/9 indicates that a comma or a tab can separate each field. Use the text “FIX” for fixed width fields.
2	text delimiter	ASCII value of the text delimiter. In a CSV file, the text portions are usually surrounded by either double quotation marks (") or single quotation marks ('). To recognize multiple delimiters, separate them with a slash; for example 34/39 specifies both a double and a single quote.
3	character set	Numeric index that identifies which character set to use. The value of 0 represents the system character set. This might also be called the code page.
4	first line	First line to import. The value 1 indicates the first line. The value 2, causes the first column to be ignored.
5	field specifier	The field specifier identifies the columns or fields to import.

If the filter options start with a field separator, then the field specifier is in the format shown in *Listing 46*. The field_num is an integer that identifies a field, where 1 is the leftmost field. For example, given the fields “one”, “two”, and “three”, a field_num of 2 refers to “two”. The format is an integer that identifies the format of the field (see *Table 50*).

Listing 46. Delimited-text-format string for the CSV filter.

```
field_num/format/field_num/format/field_num/format/...
```

If the filter options start with the text “Fix”, as is shown in *Listing 44*, then the field specifier is in the format shown in *Listing 47*. The column value specifies the first character in a field. A column of 0 refers to the leftmost character of the text. The format is an integer that identifies the format of the text (see *Table 50*).

Listing 47. Delimited-text-format string for the CSV filter.

```
column/format/column/format/column/format/...
```

Table 50. CSV field format values.

Format	Description
1	Standard format allows the import filter to guess the type.
2	Text
3	MM/DD/YY
4	DD/MM/YY
5	YY/MM/DD
9	Do not import; ignore this field.

Format	Description
10	Import a number formatted in the US-English locale regardless of the locale.

To access a fixed width text file using a database connection, open the file as a Calc document and then either access the Calc document as a database, or export the data as a delimited file and then open the connection to the delimited text file. The macro in *Listing 47* demonstrates how to read a fixed width text file by transforming the file to a Calc document and a delimited CSV file as follows:

- 1) Create the fixed width data file delme1.csv.
- 2) The fixed width data is imported into a Calc document using the CSV import filter.
- 3) The Calc document is saved as a Calc document (delme1.ods).
- 4) The Calc document is exported to a delimited CSV file.
- 5) A connection is created to the delimited file.
- 6) A connection is made to the Calc document.
- 7) The created files are deleted.

Read Fixed Width File Use the button **Read Fixed Width File** to created and read the fixed width files.

Listing 48. Created and read fixed width files and a Calc document.

```
Sub ReadFixedWidthFile()
    Dim sFileName$ 'The base file name.
    Dim n As Integer 'The file number.
    Dim i As Integer 'General index variable.
    Dim s As String 'Temporary string.
    Dim sURLInitial$ 'URL of the CSV file.
    Dim sURLCalc$ 'URL of the Calc document.
    Dim sOut() 'Output text initially written to the CSV file.
    Dim oManager 'Connection driver manager.
    Dim oCon 'Connection object.
    Dim sSQL$ 'SQL that is executed.
    Dim oResult 'Result from an SQL statement.
    Dim oStatement 'A created statement that can execute SQL.
    Dim oDoc 'Calc document used to read the fixed width data.
    Dim oParms() As New com.sun.star.beans.PropertyValue

    REM First, create the following text file:
    REM Date      Name      Number
    REM 01/01/05 Danny Bot   17
    REM 12/25/99 Shelly Girt 13
    REM 03/13/65 Fred Krank  7

    sOut() = Array("Date      Name      Number", _
                   "01/01/05 Danny Bot   17", _
                   "12/25/99 Shelly Girt 13", _
```

```

"03/13/65Fred Krank 7 ")

sFileName = "delme1"
sURLInitial = GetSourceCodeDir() & sFileName & ".csv"

n = FreeFile() 'Next free file number

REM Open for read/write
Open sURLInitial For Output Access Read Write As #n
For i = 0 To UBound(sOut())
    Print #n, sOut(i)
Next
Close #n

REM Open the fixed width text file into a Calc document.
AppendProperty(oParms(), _
    "FilterName", "Text - txt - csv (StarCalc)")
AppendProperty(oParms(), "FilterOptions", "FIX,34,0,1,0/3/8/2/19/10")
oDoc = StarDesktop.LoadComponentFromUrl(sURLInitial, "_blank", _
    0, oParms())

REM Save the document as a Calc document!
ReDim oParms()
AppendProperty(oParms(), "Overwrite", "True")
sURLCalc = GetSourceCodeDir() & sFileName & ".ods"
oDoc.storeAsURL(sURLCalc, oParms())

REM Delete the initial fixed width file.
Kill(sURLInitial)

REM Write the file as a comma delimited text file. Notice that only the
REM text columns contain the double quote text delimiter.
REM "Date","Name","Number"
REM 01/01/2005,"Danny Bot",17
REM 12/25/1999,"Shelly Girt",13
REM 03/13/1965,"Fred Krank",7
ReDim oParms()
AppendProperty(oParms(), _
    "FilterName", "Text - txt - csv (StarCalc)")
AppendProperty(oParms(), "FilterOptions", "44,34,0,1,1/1/2/2/3/10")
oDoc.storeAsURL(sURLInitial, oParms())
oDoc.close(True)

oManager = CreateUnoService("com.sun.star.sdbc.DriverManager")

REM Open the CSV file.
REM Notice that I the header line is ignored.
REM If the header line is NOT ignored, then all
REM columns are recognized as type VarChar.
ReDim oParms()
AppendProperty(oParms(), "Extension", "csv")
AppendProperty(oParms(), "HeaderLine", True)

```

```

AppendProperty(oParms(), "FieldDelimiter", ",")
AppendProperty(oParms(), "StringDelimiter", "\"")
AppendProperty(oParms(), "DecimalDelimiter", ".")
AppendProperty(oParms(), "ThousandDelimiter", ",")

oCon = oManager.getConnectionWithInfo("sdbc:flat:" & _
    sURLInitial, oParms())

oStatement = oCon.createStatement()
sSQL = "SELECT * FROM " & DBQuoteName(sFileName, oCon)
oResult = oStatement.executeQuery(sSQL)

Do While oResult.next()
    s = s & "CSV File: Date = " & _
        CStr(UNODateToDate(oResult.getDate(1))) & _
        " Name = '" & oResult.getString(2) & "'" & _
        " Number = " & CStr(oResult.getLong(3)) & CHR$(10)
Loop
oCon.close()

oCon = oManager.getConnection("sdbc:calc:" & sURLCalc)
oStatement = oCon.createStatement()
sSQL = "SELECT * FROM " & DBQuoteName(sFileName, oCon)
oResult = oStatement.executeQuery(sSQL)
s = s & CHR$(10)
Do While oResult.next()
    s = s & "Calc File: Date = " & _
        CStr(UNODateToDate(oResult.getDate(1))) & _
        " Name = '" & oResult.getString(2) & "'" & _
        " Number = " & CStr(oResult.getLong(3)) & CHR$(10)
Loop
oCon.close()
MsgBox s, 0, "Data from the flat file"

REM Delete the created files.
Kill(sURLCalc)
Kill(sURLInitial)
End Sub

```

TIP It is very important that you use the correct filter name. When I used “scalc: Text - txt - csv (StarCalc)”, rather than “Text - txt - csv (StarCalc)”. The macro worked fine on one computer, but on the other computer, the CSV file loaded into a text document rather than a Calc document—thanks to Paolo Mantovani for providing the correct solution.

Figure 22 demonstrates that the data read from the Calc document is the same as the data read from the CSV file.

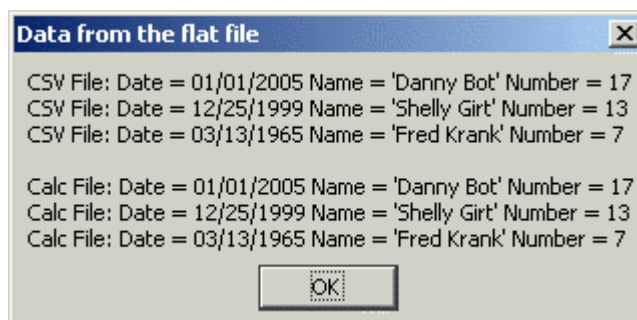


Figure 22. The CSV and Calc files contain the same data.

It is easier to import a Calc document than a delimited text file. Therefore, I recommend that you read fixed width data files as Calc documents rather than delimited files. The flat file database connection guesses the column types based on data in the first row. If the header is included with the CSV file, then all fields are considered text fields. The Calc CSV import filter allows you to specify the data types. Also, connections to Calc documents do not require connection arguments.

8.6.3. Help, I still can not import my CSV file

It can be difficult to obtain the correct values to open a CSV file. The most common problems are:

- 1) The file extension must be “.csv”; you can not open a “.txt” file into a calc document as a comma delimited file.
- 2) Using the wrong filter name.
- 3) Determining the correct character set, sometimes referred to as the code page (see *Table 51*).
- 4) Determining the correct field widths.

If you do not know the character set, try some of the values shown in *Table 51*. I am likely to try 0, 65535, and then 1252.

Table 51. Character sets for use with CSV import.

Character Set	Comment
0	The system character set frequently works (tested).
65535	Unicode is very popular these days (tested).
437	Original DOS Latin character set(not tested).
850	DOS Latin 1 (not tested).
851	Dos Latin 2 (not tested).
1252	Windows Latin 1 (not tested).
1250	Windows Latin 2 (not tested).

If you still have difficulties, try importing the file using the GUI and then inspect the document to see which import arguments were used.

Listing 49. Inspect document arguments.

```
Sub InspectDocArgs
    Dim args()
    Dim x, i%, s$
    args() = ThisComponent.getArgs()
    On Error Resume Next
    For i = LBound(args()) To UBound(args())
        x = args(i)
        s = s & x.Name & " : "
        s = s & CStr(x.Value)
        s = s & CHR$(10)
    Next
    MsgBox s
End Sub
```

After inspecting the arguments (see *Listing 49*), you can see the filter and the filter arguments (see *Figure 23*). If you can figure out how to import the document using the GUI, you should be able to import the file from a macro.

Figure 23: Arguments from a CSV file.

8.6.4. Address books

The address book connection provides access to Mozilla, Outlook, Outlook Express, and LDAP address books. The macro in *Listing 50* demonstrates how to read the names and email addresses supported by the local Outlook address book. Each address book type (see *Table 45*) implements a different set of tables and columns. I inspect the connection meta-data to determine the name of the tables and supported columns. The Outlook address book uses the “OP Contacts” table, and the Mozilla address book has two tables, “Personal Address book” and “Collected Addresses”. I checked this macro on a windows computer and it displayed the email addresses that I personally added, not the global addresses.

Listing 50. Inspect an Outlook address book.

```
Sub InspectOutlookAddress()
    Dim oManager    'Connection driver manager.
    Dim oCon        'Connection object.
    Dim sSQL$       'SQL that is executed.
    Dim oResult     'Result from an SQL statement.
    Dim oStatement  'A created statement that can execute SQL.
    Dim s$          'General string variable.
    Dim nCount&

    oManager = CreateUnoService("com.sun.star.sdbc.DriverManager")
    oCon = oManager.getConnection("sdbc:address:outlook:")

    oStatement = oCon.createStatement()
    sSQL = "SELECT " & DBQuoteName("First Name", oCon) & ", " & _
        DBQuoteName("Last Name", oCon) & ", " & _
        DBQuoteName("E-mail", oCon) & " FROM " & _
        DBQuoteName("OP Contacts", oCon)
    oResult = oStatement.executeQuery(sSQL)

    REM Limit the returned values to 50!
    nCount = 0
    Do While oResult.next() AND nCount < 50
        s = s & oResult.getString(1) & " " & oResult.getString(2) & _
            " ==> " & oResult.getString(3) & CHR$(10)
    Loop
    MsgBox s, 0, "Public e-mail"
    oCon.close()
End Sub
```

8.6.5. MySQL using JDBC

Connecting to a MySQL database using JDBC is easy if you know proper connection URL. The macro in *Listing 51* connects to a the “stamps” MySQL database running on my local computer (local host) using the default port (3306). Setting the Java driver class property is optional and it defaults to com.mysql.jdbc.Driver.

Listing 51. Connect to a MySQL database using JDBC.

```
sUser$ = "user"
sPass$ = "password"
```



```

sURL$ = "sdbc:mysql:jdbc:localhost:3306/stamps"

oManager = CreateUnoService("com.sun.star.sdbc.DriverManager")
AppendProperty(oParms(), "user", sUser)
AppendProperty(oParms(), "password", sPass)
AppendProperty(oParms(), "JavaDriverClass", "com.mysql.jdbc.Driver")
oCon = oManager.getConnectionWithInfo(sURL, oParms())

```

TIP *Listing 51* assumes that Java is installed, the JDBC driver is installed, and the OpenOffice.org has been properly configured to use the driver.

8.6.6. Paradox using ODBC

ODBC (Open DataBase Connectivity), pronounced as separate letters, is a standard database access method developed in 1992 by the SQL Access group. ODBC is a middle layer between a DBMS (database management system) and an application. ODBC translates data queries into commands that the DBMS understands. SDBC knows how to talk to an ODBC data source.

An ODBC data source must be registered before it can be used. Use the ODBC Data Source Administrator to add and configure an ODBC data source on Windows. Use the control panel (Start | Settings | Control Panel) to open the Control Panel. With Microsoft Windows 2000 or newer, use Administrative Tools | Data Sources (ODBC) to open the ODBC Data Source Administrator (see *Figure 24*). For older versions of Windows, use 32-bit ODBC or ODBC.

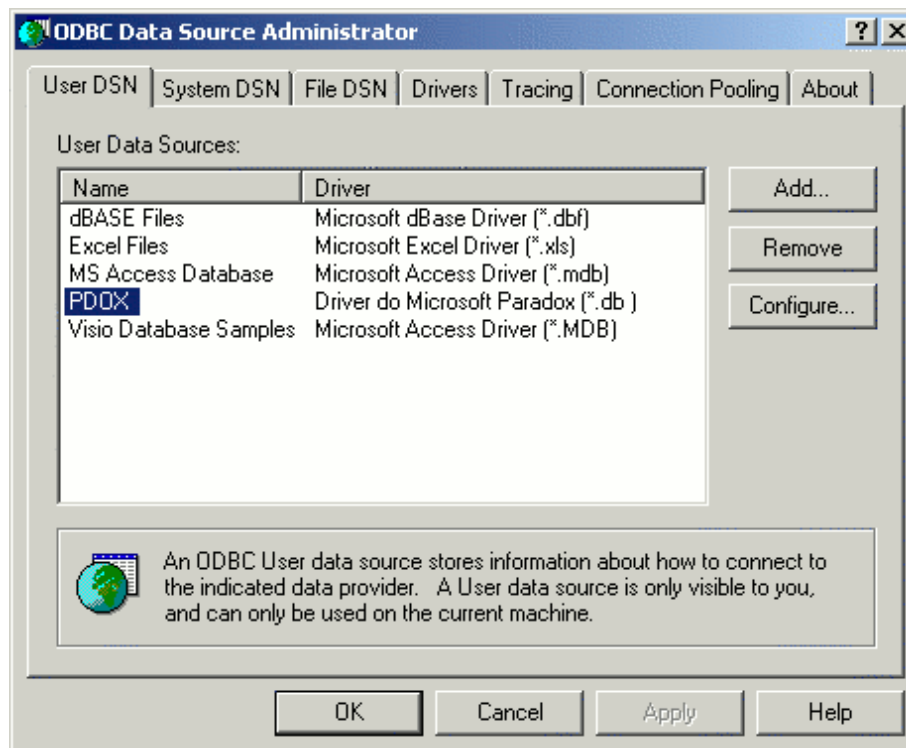


Figure 24. ODBC data sources on Windows XP Professional.

I used ODBC to connect to a set of existing Paradox tables. I was very happy to find out that Windows XP Professional came with drivers for Paradox, and many other database systems, already installed. The macro in *Listing 52* connects to the defined PDOX data source and accesses the COUNTRY table.

Listing 52. Connect to a Paradox database using ODBC.

```
Sub UsePDoxODBC
    Dim oManager      'Connection driver manager.
    Dim sURL$         'DB URL including "sdbc:flat:..."
    Dim oCon          'Connection object.
    Dim sSQL$         'SQL that is executed.
    Dim oResult       'Result from an SQL statement.
    Dim oStatement    'A created statement that can execute SQL.
    Dim oDoc           'Calc document that will contain the presentation data.
    Dim oData()       'Generic data variable.
    Dim s$            'General utility string.

    sURL = "sdbc:odbc:PDOX"
    oManager = CreateUnoService("com.sun.star.sdbc.DriverManager")
    oCon = oManager.getConnection(sURL)
    oStatement = oCon.createStatement()
    sSQL = "SELECT * FROM " & DBQuoteName("COUNTRY", oCon)
    oResult = oStatement.executeQuery(sSQL)
    Do While oResult.next()
        s = s & oResult.getString(1) & " " & oResult.getString(2) & CHR$(10)
    Loop
```

```
oCon.close()  
MsgBox s  
End Sub
```

8.6.7. Conclusion

Determining the connection URL is probably the most difficult part of connecting to an external database. I frequently create a connection and then inspect the meta data to determine the format of the URL. This chapter provides sufficient information to connect to a database and sets the stage for more advanced applications.

9. Connecting to MySQL using JDBC

To connect to MySQL using JDBC, Java must be installed on your computer. From OOo, use **Tools > Options > OpenOffice.org > Java** to open the Java tab of the options dialog. Verify that a Java runtime environment is configured for use with OpenOffice.org. If Java is not installed, visit <http://java.sun.com/> to download and install Java. I installed J2SE 5.0, which is sometimes referred to as version 1.5. Developers usually install the JDK (Java Development Kit) version so that they can create software using Java. Non-developers, however, usually install the JRE (Java Runtime Environment) version because it is smaller and they do not require the developer tools. After installing Java, you may need to restart OOo so that it can recognize Java.

Download the latest JDBC driver from <http://www.mysql.com>; the main MySQL web page. From the main MySQL page, click **Products > Connectors > MySQL Connector/J** to open the JDBC download page. Select and download an appropriate driver. Unpack the driver using a zip or tar utility and make a note of where you placed the driver. You must set the class path so that OOo can find the MySQL JDBC drivers. Click the Class Path button, from the Java options page, to open the Class Path dialog. Click the Add Folder button to add the directory containing the JDBC driver. You must restart OpenOffice.org before the new class path will be recognized.

Use **File > New > Database** to start the Database Wizard. Select Connect to an existing database and choose the MySQL Database type. Click Next to setup the MySQL connection. Choose the JDBC connector and click Next to setup the JDBC connection.

In MySQL, you connect to a database, which is collection of tables. The JDBC connection in *Figure 25* is configured to connect to the stamps database on the current computer using the default port.

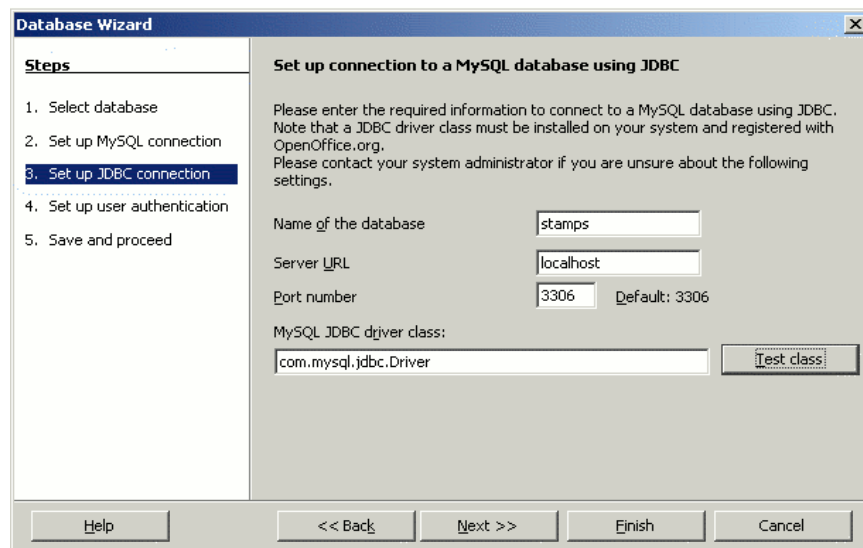


Figure 25. Configure and test the JDBC connection.

The default JDBC driver is `com.mysql.jdbc.Driver`. Click the Test class button to verify that the JDBC driver can be loaded. If the driver can not be loaded, verify that the class path is properly configured in OOo. Click Next to setup user authentication

Enter the user name and choose to use a password if required. Click the Test Connection button to verify that a connection can be made to the database using the configured parameters. You will be prompted for the password, if one is required. To connect to a MySQL database, the MySQL server must be running.

Click Next to move to the final dialog in the Database Wizard. You do not need to register the database unless you want to reference the data source based on a name rather than using the URL of the Database document. Click the Finish button to finish and save the document. I chose to save the document using the name StampsJDBC.

OoO contains a dedicated MySQL/JDBC bridge, which handles certain MySQL peculiarities such as parameter handling. The connection returned from the GUI uses the special MySQL connection. While using the connection manager in a macro, however, you must be certain to specify the correct type (see *Listing 51*).

The "sdbc:mysql:jdbc:localhost:3306/stamps" connection string uses the SDBC MySQL driver, which is the desired connection. Using "jdbc:mysql://localhost:3306/stamps", uses a standard JDBC connection, that does not contain the "special sauce", so to speak.

TIP Always use "sdbc:mysql:jdbc" rather than "jdbc:mysql".

10. Mailmerge

??

Insert content from chapter 10 as it is. Also, use the new merge to email capability in 2.01.

11. Copying an entire database

There is an example here: <http://www.ofoforum.org/forum/viewtopic.phtml?t=22858&view=previous>

12. General utility macros

This section introduces general utility macros and establishes the foundation on which the database related macros are built. Many interesting and useful macros that are not directly related to database programming are used in this document. The macros also demonstrate some useful programming techniques.

Macros are considered an advanced topic that are an absolute necessity for some and never required for others. The macro coverage in this book assumes a working knowledge of macros so no preliminary macro information is discussed. For more information on macro programming, see my book “OpenOffice.org Macros Explained.”

OOo supports different documents types such as Writer (text document) and Calc (spreadsheet document). Each unique document type supports at least one service that is supported only by that document type (see *Table 52*). You can determine the type of document by checking if it supports one of these services.

Table 52. Fields in the binary table.

Document Type	Service
Drawing	com.sun.star.drawing.DrawDocument
Writer	com.sun.star.text.TextDocument
Writer/Web	com.sun.star.text.WebDocument
Master Document	com.sun.star.text.GlobalDocument
Calc	com.sun.star.sheet.SpreadsheetDocument
Math	com.sun.star.formula.FormulaProperties
Presentation (Impress)	com.sun.star.presentation.PresentationDocument
Database (Base)	com.sun.star.sdb.DatabaseDocument

A Database document implements the DatabaseDocument service. The macro in *Listing 53* checks to see if the variable oDoc references a database document.

Listing 53: Check if oDoc references a database document.

```
If oDoc.supportsService("com.sun.star.sdb.DatabaseDocument") Then
```

Unlike other document types, a database document can not contain macros, which is probably why ThisComponent is not set to reference a database document. To obtain the currently focused component, including a database document, use the getCurrentComponent() method of the desktop object (see *Listing 54*). ThisComponent references the most recently focused document that is not a database document. Unfortunately, getCurrentComponent() also returns components such as the Basic IDE.

Listing 54: Get the current component and ThisComponent.

```
oDoc = ThisComponent  
oComp = StarDesktop.getCurrentComponent()
```

TIP

Later versions of OpenOffice.org may allow database documents to contain macros. Also, in future versions, ThisComponent may be set to reference a database document; until then, you

need to use `getCurrentComponent()`.

12.1. Choose a directory

The PathSettings service provides an easy method to get the user's document directory.

Work Directory Use the **Work Directory** button to print your working directory.

Listing 55: Get the user's document directory from the PathSettings service.

```
Function GetWorkDir() As String
    Dim oPathSettings
    oPathSettings = CreateUnoService("com.sun.star.util.PathSettings")
    GetWorkDir() = oPathSettings.Work
End Function
```

The FolderPicker service returns a dialog to select directories. The actual service that is returned depends on the operating system and options specified in the Options dialog. **Choose Director** Use the **Choose Directory** button to call the macro in *Listing 56*; the starting directory should be your work directory.

Caution



In OOO version 2.0, the Win32FolderPicker has a bug, which prevents you from setting the initial directory. Use **Tools > Options > OpenOffice.org > General** to open Options dialog, and then check the *Use OpenOffice.org Dialogs* checkbox. Baring that, you can explicitly use the OfficeFolderPicker service rather than the FolderPicker service.

Listing 56: Select a directory.

```
REM sInPath specifies the initial directory. If the initial directory
REM is not specified, then the user's default work directory is used.
REM The selected directory is returned as a URL.
Function ChooseADirectory(Optional sInPath$) As String
    Dim oDialog As Object
    Dim oSFA As Object
    Dim s As String

    Rem You can also use com.sun.star.ui.dialogs.OfficeFolderPicker
    oDialog = CreateUnoService("com.sun.star.ui.dialogs.FolderPicker")
    oSFA = createUnoService("com.sun.star.ucb.SimpleFileAccess")

    If IsMissing(sInPath) Then
        oDialog.setDisplayDirectory(GetWorkDir())
    ElseIf oSFA.Exists(sInPath) Then
        oDialog.setDisplayDirectory(sInPath)
    Else
        s = "Directory '" & sInPath & "' Does not exist"
        If MsgBox(s, 33, "Error") = 2 Then Exit Function
    End If

    If oDialog.Execute() = 1 Then
        ChooseADirectory() = oDialog.getDirectory()
```

```
End If
End Function
```

12.2. Get a document's directory

Many of the macros in this document assume that the working directory is the directory that contains this document. The macro in *Listing 57* demonstrates how to obtain the directory portion of a URL. **Print Doc Dir** Use the **Print Doc Dir** button to call the macro in *Listing 57*.

Listing 57: Select a directory.

```
REM Determine the directory used by this document.
REM It is assumed that the rest of the interesting data files
REM are in the same directory.
REM Returns the document's directory as a URL with a trailing "/".
Function GetSourceCodeDir(Optional useDoc) As String
    Dim oDoc 'The document on which to work.
    Dim s$   'Temporary string variable.

    If NOT IsMissing(useDoc) Then
        oDoc = useDoc
    End If
    If IsEmpty(oDoc) OR IsNull(oDoc) Then
        oDoc = ThisComponent
    End If
    If oDoc.HasLocation() Then
        GlobalScope.BasicLibraries.LoadLibrary("Tools")
        REM This is assumed to be in URL notation!
        s = DirectoryNameoutofPath(oDoc.getLocation(), "/")
        GetSourceCodeDir() = s & "/"
    Else
        Print "Warning, current document has no location"
    End If
End Function
```

12.3. Choose a file

Use the `FilePicker` service to select a file. There are many configurable options for setting the default behavior using `TemplateDescription` constants. If the optional input path is set and contains a file name, then that file name is selected by default.

Listing 58: Select a file.

```
REM sInPath specifies the initial directory. If the initial directory
REM is not specified, then the user's default work directory is used.
REM The selected file is returned as a URL.
Function ChooseAFile$(sFilters(), bOpen As Boolean, Optional sInPath$)
    Dim oDialog As Object
    Dim sPath As String
    Dim oSFA As Object
    Dim s As String
    Dim i As Integer

    oDialog = CreateUnoService("com.sun.star.ui.dialogs.FilePicker")
```

```

oSFA = createUnoService("com.sun.star.ucb.SimpleFileAccess")

REM See the TemplateDescription constants to see what other
REM values are supported.
If bOpen Then
    i = com.sun.star.ui.dialogs.TemplateDescription.FILEOPEN_SIMPLE
Else
    REM When choosing a file that already exists, you will be asked
    REM if you want to overwrite the file.
    i = com.sun.star.ui.dialogs.TemplateDescription.FILESAVE_SIMPLE
End If
oDialog.initialize(Array(i))

If IsMissing(sInPath) Then
    oDialog.setDisplayDirectory(GetWorkDir())
ElseIf oSFA.Exists(sInPath) Then
    oDialog.setDisplayDirectory(sInPath)
Else
    s = "Directory '" & sInPath & "' Does not exist"
    If MsgBox(s, 33, "Error") = 2 Then Exit Function
End If

For i = LBound(sFilters()) To UBound(sFilters()) Step 2
    Dim sFilterName$
    Dim sFilterValue$
    sFilterValue = sFilters(i+1)
    sFilterName = sFilterValue & " - " & sFilters(i)
    oDialog.appendFilter(sFilterName, sFilterValue)
Next

If oDialog.Execute() = 1 Then
    sPath = oDialog.Files(0)
    ChooseAFile() = sPath
End If
End Function

```

The macro in *Listing 58* expects a list of filters to limit the files displayed to the user. The macros in *Listing 59* demonstrate how to build the list.

Listing 59: Build filters for selecting graphics files or Base documents.

```

Function OOoBaseFilters()
    OOoBaseFilters() = Array("All Files", "*.*", _
        "OOo Base", "*.odb")
End Function

Function GraphicFilters()
    GraphicFilters() = Array("All Files", "*.*", _
        "Graphic Interchange Format", "*.gif", _
        "Joint Photographic Experts Group", "*.jpg", _
        "Tag Image File Format", "*.tif", _
        "Windows Bit Map", "*.bmp", _
        "Gimp Files", "*.xcf", _

```

```

    "Portable Network Graphics", "*.png")
End Function

```

Find File Use the **Find File** button to call the macro in *Listing 60*. The initial filename and directory are not specified, so the your default working directory is used.

Listing 60: Obtain a graphic filename for input.

```

Sub ChooseGraphicFileForOpen()
    LoadDBLibs()
    Print "You Chose: " & ChooseAFile$(GraphicFilters(), True)
End Sub

```

As nice as the FilePicker is, it can be very annoying, or very friendly, depending on your desires. If you choose to save a file, then you are asked if you want to overwrite the file. On the other hand, if you are opening a file, then you can not specify a file that does not exist. This behavior can be a curse, or a blessing, depending on your intentions.

12.4. Finding a (loaded) OOo document

I want a reference to a document that is already loaded. To solve this problem, obtain an enumeration of open components from the desktop object and then check the URL for each component. The enumeration includes all open components, including the Basic IDE and the OOo help, which should be ignored. Also, if a document has not been saved, then it does not have a URL—check this by calling the `hasLocation()` method.

Listing 61: Safely get the URL from an OOo component.

```

REM Care must be taken while enumerating components to only obtain a URL
REM from components that support them.
Function GetDocURL(oDoc) As String
    GetDocURL() = ""
    If NOT HasUNOInterfaces(oDoc, "com.sun.star.frame.XStorable") Then
        REM The OOo help does not support the XStorable interface, but the
        REM Basic IDE does. A truly paranoid person, therefore, might check
        REM for the com.sun.star.document.OfficeDocument service.
        REM I could also check for the com.sun.star.script.BasicIDE service.
    ElseIf NOT oDoc.hasLocation() Then
        REM This document has never been saved, so there is no URL
        REM to compare against.
    Else
        GetDocURL() = oDoc.getURL()
    End If
End Function

```

If an OOo document is already loaded, the macro in *Listing 62* will find it based on its URL or file name. If the file name is used rather than the full URL, the file extension is optional but safer—the files `Util.odt` and `Util.odt` look the same without a file extension. If a document is not found in the enumeration returned by the desktop object, then it is loaded if the second argument is true.

Caution Use the `StarDesktop.loadComponentFromURL()` to load an OOo document. In OOo version 1.x, a NULL value is returned if the document does not exist. In OOo version 2.0, however, an illegal argument exception is thrown instead.



Listing 62: Find an OOo document if it is already loaded.

```

Function FindComponentWithURL(sName$, bLoadIfNotFound As Boolean)
    Dim oDocs      ' Enumeration of the loaded components.
    Dim oDoc        ' A single enumerated component.
    Dim sDocURL$    ' URL of the component that we are checking.

    REM Use some methods from the Tools library.
    If NOT GlobalScope.BasicLibraries.isLibraryLoaded("Tools") Then
        GlobalScope.BasicLibraries.LoadLibrary("Tools")
    End If

    oDocs = StarDesktop.getComponents().createEnumeration()
    Do While oDocs.hasMoreElements()
        oDoc = oDocs.nextElement()
        sDocURL = GetDocURL(oDoc)

        REM Just in case the name contains the full URL.
        REM If the name is an empty string, then return an unsaved document.
        If sName = sDocURL Then
            FindComponentWithURL() = oDoc
            Exit Function
        End If

        REM This will only work if the name contains the file extension.
        If InStr(sDocURL, "/") > 0 Then
            If FileNameoutofPath(sDocURL, "/") = sName Then
                FindComponentWithURL() = oDoc
                Exit Function
            End If
        End If
    Loop

    REM The document was not found,
    REM perhaps the name did not contain a file extension.
    oDocs = StarDesktop.getComponents().createEnumeration()
    Do While oDocs.hasMoreElements()
        oDoc = oDocs.nextElement()
        sDocURL = GetDocURL(oDoc)

        If InStr(sDocURL, "/") > 0 Then
            If GetFileNameWithoutExtension(sDocURL, "/") = sName Then
                FindComponentWithURL() = oDoc
                Exit Function
            End If
        End If
    Loop

```

```

REM The name was still not found, check to see if a document exists
REM with the specified URL.
REM In OOo version 1.x, loadComponentFromURL() returned NULL
REM if the document did not exist. Starting with version 2.x,
REM an illegal argument exception is thrown instead.
If bLoadIfNotFound AND FileExists(sName) Then
    oDoc = StarDesktop.loadComponentFromURL(sName, "_blank", 0, Array())
    FindComponentWithURL() = oDoc
Else
    ' FindComponentWithURL = NULL
End If
End Function

```

12.5. Append to an array

While accumulating data, it is convenient to add data to an array without worrying about the size of the array—it may not be efficient, but it is convenient. The macro in *Listing 63* increases the size of an array by one, and then appends data to the end of the array. This technique is invaluable when space and complexity are more important than runtime—appending data to an array is similar to pushing data onto a stack.

Listing 63. Append data to an array.

```

Sub AppendToArray(oData(), ByVal x)
    Dim iUB As Integer 'The upper bound of the array.
    Dim iLB As Integer 'The lower bound of the array.
    iUB = UBound(oData()) + 1
    iLB = LBound(oData())
    ReDim Preserve oData(iLB To iUB)
    oData(iUB) = x
End Sub

```

TIP The first version of AppendToArray() passed the second argument as a reference. I then wrote a loop that used a temporary variable to collect data. At each iteration, the reference to the temporary variable was pushed onto the end of the array. The final result was that every value in the array was the same.

12.6. Compare data in an array

I frequently convert data into data arrays and then copy the data arrays directly into a spreadsheet. I initially wrote this routine to check individual arrays to see if they contained duplicate data.

Listing 64. Check to see if two arrays contain the same text.

```

REM check to see if two arrays contain the same data.
REM Both arrays are assumed to be one dimensional arrays.
REM The data in both arrays are assumed to be directly
REM comparable data.

```

```

Function ArraysAreSame(oData1(), oData2(), nMinCol As Long) As Boolean
    Dim n As Long
    ArraysAreSame() = False

    REM First, compare the upper dimension.
    If UBound(oData1()) <> UBound(oData2()) Then Exit Function
    If LBound(oData1()) <> LBound(oData2()) Then Exit Function
    If LBound(oData1()) < nMinCol Then Exit Function

    For n = LBound(oData1()) To UBound(oData1())
        If oData1(n) <> oData2(n) Then
            Exit Function
        End If
    Next
    ArraysAreSame() = True
End Function

```

12.7. Create a property

I have always found it annoying to create an array of properties to pass as arguments to the UNO API; create a properly dimensioned array, set the property values, realize that an extra property is required so I need to start over. The `CreateProperty()` method in *Listing 65* simplifies the creation of properties. accepts a property name and a property value—a new property is created and returned. The real strength of the `CreateProperty()` method is seen when used in conjunction with the `AppendProperty()` method, which creates a new property and appends it to an array (see *Listing 65*).

Listing 65. Create a PropertyValue structure and append one to an array.

```

Function CreateProperty(sName$, oValue) As com.sun.star.beans.PropertyValue
    Dim oProperty As New com.sun.star.beans.PropertyValue
    oProperty.Name = sName
    oProperty.Value = oValue
    CreateProperty() = oProperty
End Function

Sub AppendProperty(oProperties(), sName As String, ByVal oValue)
    AppendToArray(oProperties(), CreateProperty(sName, oValue))
End Sub

```

12.8. Create a Point and a Size

Many routines require a point or size object. It is easier to create and return an item in a function.

Listing 66. Create a Point and Size.

```

Function CreatePoint(xPos, yPos) As New com.sun.star.awt.Point
    Dim oPoint As New com.sun.star.awt.Point
    oPoint.X = xPos
    oPoint.Y = yPos
    CreatePoint() = oPoint
End Function

Function CreateSize(iWidth, iHeight) As New com.sun.star.awt.Size

```

```

Dim oSize As New com.sun.star.awt.Size
oSize.Width = iWidth
oSize.Height = iHeight
CreateSize() = oSize
End Function

```

12.9. Append a data array to a Calc document

I frequently write test macros that display data in a dialog. A dialog is not sufficient to display a large amount of data, so I sometimes use a Calc document instead. The macro in *Listing 67* accepts a data array, which is added to the Calc document at the current cursor position. After inserting the data, the cursor is moved below the last inserted row. Repeated calls to `AppendDataToCalcDoc()` append more data to the end of the document and reposition the cursor.

Listing 67. Add data array to a Calc document at the cursor position.

```

Sub AppendDataToCalcDoc(oDoc, oData())
    Dim oSheet      'The first sheet in the document.
    Dim oAddr       'Address of the current cursor.
    Dim iNumRows%   'Number of rows to add.
    Dim iNumCols%   'Number of columns to add.
    Dim x()         'Utility variable used as a single row.
    Dim oRange      'Range to add the data and focus the cursor.

    REM Get the address of the currently selected cell.
    REM Use the address to obtain the current sheet.
    oAddr = oDoc.getCurrentSelection().getCellAddress()
    oSheet = oDoc.getSheets().getByIndex(oAddr.Sheet)

    REM Determine the number of rows in the data.
    REM Next, obtain the first row and see how many columns it has.
    iNumRows = UBound(oData()) - LBound(oData())
    x() = oData(LBound(oData()))
    iNumCols = UBound(x()) - LBound(x())

    oRange = oSheet.getCellRangeByPosition(oAddr.Column, oAddr.Row, _
        oAddr.Column + iNumCols, oAddr.Row + iNumRows)
    oRange.setDataArray(oData())
    oRange.getColumns().optimalWidth = True
    oRange = oSheet.getCellByPosition(oAddr.Column, oAddr.Row + iNumRows+1)
    oDoc.getCurrentController().select(oRange)
End Sub

```

The `setDataArray()` method for a Calc sheet is used because it is fast and efficient, but the data array must be a single dimension array. Each element in the data array corresponds to a single row of data and is represented by a one dimensional array(see *Table 53*).

Table 53. Data arrangement for use in setDataArray()

Correct: Each row is an array	Incorrect: A two dimension array
Dim oData(3) oData(0)=Array(1, 2, 3) oData(1)=Array(4, 5, 6) oData(2)=Array(7, 8, 9)	Dim oData(3, 3) oData(0, 0)=1 : oData(0, 1)=2 : oData(0, 2)=3 oData(1, 0)=4 : oData(1, 1)=5 : oData(1, 2)=6 oData(2, 0)=7 : oData(2, 1)=8 : oData(2, 2)=9

12.10. Dynamically call object methods

The macros introduced in this section are very advanced and easily skipped by a novice. Before examining these macros, it is important to understand the problem. Assume that you have an object and the name of the method (as a string) that you want to call. If you know the name of the method when you write your macro, you can directly call the method. If you do not know the method name ahead of time, however, then it is not possible to directly call the method on an object based on its name. For example, if you want to write a macro that asks the user which method to call. The macro in *Listing 68* demonstrates a function called “MyInsertedFunc”. The “MyInsertedFunc” function returns the value returned by the getName() method on the oObj object.

TIP Although you can not dynamically call a method based on its name, most of an objects properties are available through the property set information object available through the method getPropertySetInfo(). At runtime, you can discover the name of an objects properties and then obtain the values using the property set information object. You can also discover the names of an objects methods at runtime, but you can not call them based on the name.

Listing 68. Call the getName() function on the object oObj.

```
Option Explicit
Function MyInsertedFunc(ByRef oObj)
    MyInsertedFunc = oObj.getName()
End Function
```

The macro in *Listing 68* is so trivial and obvious that it seems superfluous until you realize that you can write a macro to generate the macro in *Listing 68* based on the method name. I created a function named RunFromLib(). When RunFromLib() is called as shown in *Listing 69*, it performs the following tasks:

- Create a library named xlib in the global libraries object.
- Create a module named TMod in the xlib library.
- Create the function named MyInsertedFunc (see *Listing 68*), which calls the function getName() on the oDoc object.
- Call the function MyInsertedFunc(); this works because the function name MyInsertedFunc never changes.
- Perform cleanup.

Listing 69. Call the `getName()` function on the object `oDoc`.

```
oLib = GlobalScope.BasicLibraries
RunFromLib(oLib, "xLib", "TMod", oDoc, "getName", True)
```

The arguments for `RunFromLib()` are shown in *Table 54*.

Table 54. Arguments for the `RunFromLib()` function.

Argument	Description
<code>oLibs</code>	BasicLibraries container to contain the newly created module.
<code>sLName</code>	Name of the library to use, this library can already exist. If the library does not exist, it is created.
<code>sMName</code>	Name of the module to use; the module is replaced if it already exists. The module is created to contain code similar to that shown in <i>Listing 68</i> .
<code>oObj</code>	Object that contains the method to call.
<code>sCall</code>	Name of the function to call.
<code>bClean</code>	If True, the created library and module are deleted after the call. The library is not removed if it existed before the call.
<code>x</code>	Optional argument that, if it exists, is passed to the called method.
<code>y</code>	Optional argument that, if it exists, is passed to the called method.

The `oLibs` argument is a `BasicLibraries` container for either a document or the application. The library container contains the newly created module. In my testing, I was able to use the `BasicLibraries` property from the current document and from the application, but not from a newly created document. You can not create a library in a document that is not the current document, because the object returned by the `getLibraryContainer()` method is not usable for this purpose. *Listing 70* contains a working implementation.

Listing 70. Call a named method on an object.

```
Function RunFromLib(oLibs, sLName$, sMName$, oObj, sCall$, _
                    bClean As Boolean, Optional x, optional y)
    Dim oLib      'The library to use to run the new function.
    Dim s$        'Generic string variable.
    Dim bAddedLib As Boolean

    REM If the library does not exist, then create it.
    bAddedLib = False
    If NOT oLibs.hasByName(sLName) Then
        bAddedLib = True
        oLibs.createLibrary(sLName)
    End If
    oLibs.loadLibrary(sLName)
    oLib = oLibs.getByName(sLName)

    If oLib.hasByName(sMName) Then
        oLib.removeByName(sMName)
    End If

    s = "Option Explicit" & CHR$(10) & _
        "Function MyInsertedFunc(ByRef oObj"
```

```

If NOT IsMissing(x) Then s = s & ", x"
If NOT IsMissing(y) Then s = s & ", y"
s = s & ")" & CHR$(10) & "MyInsertedFunc = oObj." & sCall & "("
If NOT IsMissing(x) Then s = s & "x"
If NOT IsMissing(y) Then s = s & ", y"
s = s & ")" & CHR$(10) & "End Function"

oLib.insertByName(sMName, s)

If IsMissing(x) Then
    RunFromLib = MyInsertedFunc(oObj)
ElseIf IsMissing(y) Then
    RunFromLib = MyInsertedFunc(oObj, x)
Else
    RunFromLib = MyInsertedFunc(oObj, x, y)
End If

If bClean Then
    oLib.removeByName(sMName)
    If bAddedLib Then
        oLibs.removeLibrary(sLName)
    End If
End If
End Function

```

While investigating meta-data, I wanted to call multiple methods on the same object. To shorten the macro listings, I created the RunMultipleCalls() method (see *Listing 71*) to call RunFromLib() multiple times. The arguments for RunMultipleCalls() are shown in *Table 55*.

Table 55. Arguments to the RunMultipleCalls() method.

Argument	Description
oObj	Object that contains the methods to call.
oData()	As each function is called, the return value is added to this array.
funcs()	Array of function names to call.
bPrintOptional	If True, optional arguments are printed with the function names.
x	Optional argument that, if it exists, is passed to the called method.
y	Optional argument that, if it exists, is passed to the called method.

The macro in *Listing 71* accepts an array of function names that are assumed to return a simple data type and accept zero, one, or two arguments. RunFromLib() is called once for each function name in the funcs() array. The oData() array is modified to contain the method name and the return value in a form suitable for use in the setDataArray() method in a Calc sheet.

Listing 71. Call multiple named methods on an object.

```

Sub RunMultipleCalls(oObj, oData(), funcs(), _
    Optional bPrintOptional, Optional x, optional y)
    Dim i%          'General index variable.

```

```

Dim z          'General utility variable.

REM First, dimension the array to hold the data.
ReDim oData(LBound(funcs()) To UBound(funcs()))

REM For each function, create a module that contains a function,
REM which is used to call the requested function on the object oObj.
For i = LBound(funcs()) To UBound(funcs())
    If IsMissing(x) Then
        z = RunFromLib(GlobalScope.BasicLibraries, _
            "xyzzylib", "TestMod", oObj, funcs(i), True)
    ElseIf IsMissing(y) Then
        z = RunFromLib(GlobalScope.BasicLibraries, _
            "xyzzylib", "TestMod", oObj, funcs(i), True, x)
    Else
        z = RunFromLib(GlobalScope.BasicLibraries, _
            "xyzzylib", "TestMod", oObj, funcs(i), True, x, y)
    End If

    If IsMissing(bPrintOptional) OR IsMissing(x) Then
        oData(i) = Array(funcs(i), CStr(z))
    ElseIf NOT bPrintOptional Then
        oData(i) = Array(funcs(i), CStr(z))
    ElseIf IsMissing(y) Then
        oData(i) = Array(funcs(i) & "(" & CStr(x) & ")", CStr(z))
    Else
        oData(i) = Array(funcs(i) & "(" & CStr(x) & ", " & CStr(y) & _
            ")", CStr(z))
    End If
Next
End Sub

```

The method `RunMultipleCallsAppendToDoc()` (see *Listing 72*) is a wrapper to call `RunMultipleCalls()` and append the returned data array to a Calc document.

Listing 72. Call named methods and append the output to a document.

```

Sub RunMultipleCallsAppendToDoc(oDoc, oObj, funcs(), _
    Optional bPrintOptional, _
    Optional x, optional y)
    Dim oData() 'Holds the data array that is set.
    If IsMissing(x) OR IsMissing(bPrintOptional) Then
        RunMultipleCalls(oObj, oData(), funcs())
    ElseIf IsMissing(y) Then
        RunMultipleCalls(oObj, oData(), funcs(), bPrintOptional, x)
    Else
        RunMultipleCalls(oObj, oData(), funcs(), bPrintOptional, x, y)
    End If
    AppendDataToCalcDoc(oDoc, oData())
End Sub

```

12.11. Display numeric constants as meaningful text

The OOo API contains many constant groups. The constants can be referenced as either the numerical value, or the full name. For example, `com.sun.star.sdbc.ColumnValue.NO_NULLS` is equivalent to the number 0. It is much easier to recognize the text name than it is to remember the numerical values.

Most constant groups used in OpenOffice.org UNO programming are sequentially numbered. For example, the `com.sun.star.sdbc.ColumnValue` constant group contains the constants `NO_NULLS`, `NULLABLE`, and `NULLABLE_UNKNOWN`, which are numbered 0, 1, and 2. The code snippet in *Listing 73* demonstrates an easy way to convert sequentially numbered constants to text; the constant value is used to index into an array containing a textual representation.

Listing 73. Convert the ColumnValue constants to text.

```
x() = Array( "NO_NULLS", "NULLABLE", "NULLABLE_UNKNOWN" )
Print x(com.sun.star.sdbc.ColumnValue.NO_NULLS)
Print x(0)
```

The macro in *Listing 74* generalizes the method demonstrated in *Listing 73* to convert the numeric value to a user friendly text entry. The first argument, `n`, is the constant value to convert. The second argument, `x()`, is an array of descriptive entries that describe the constant values. If the constants range from -3 to 5, then so should `x()`. If `bIncN` is `True`, then the returned name is preceded by numeric value. The final argument is optional, and is used as the default default descriptive entry if the constant group value does not properly index into the array.

Listing 74. Obtain data from an array without an out-of-bounds error.

```
Function SafeArray(ByVal n As Long, x(), bIncN As Boolean, _
    Optional DValue)
    Dim s As String
    If bIncN Then
        s = CStr(n) & " - "
    Else
        s = ""
    End If
    If LBound(x()) <= n AND n <= UBound(x()) Then
        SafeArray() = s & CStr(x(n))
    ElseIf NOT IsMissing(DValue) Then
        SafeArray() = s & CStr(DValue)
    Else
        SafeArray() = s & "Invalid"
    End If
End Function
```

The purpose of the `SafeArray()` function in *Listing 74* is to convert a single numeric value into a human readable text representation of the value. The macro in *Listing 75* replaces the numeric values in an entire column with a human readable text representation. The final usage of *Listing 75* reduces the amount of code required to evaluate an entire result set to a data array, and then replace a single column with a nice textual representation.

Listing 75. Find element in parallel arrays.

```

Sub SafeArrayColumns(ByVal nCol As Long, oData(), x(), _
    bIncN As Boolean, Optional DVal)
    Dim oRow()
    Dim i As Long
    Dim n As Long
    Dim dValue

    If IsMissing(DVal) Then
        dValue = "Invalid"
    Else
        dValue = DVal
    End If

    For i = LBound(oData()) + 1 To UBound(oData())
        oRow() = oData(i)
        If LBound(oRow()) <= nCol AND nCol <= UBound(oRow()) Then
            oRow(nCol) = SafeArray(CLng(oRow(nCol)), x(), bIncN, dValue)
        End If
    Next i
End Sub

```

12.12. Select from a list in a list box

The macro in *Listing 76* creates a dialog containing a list box. The data in the `oData()` array is added into the list box. The user then chooses a single item from the list box. The chosen item is returned as a string. If nothing is selected, an empty string is returned. An interesting thing about this macro, is that it dynamically builds the dialog and the controls.

Listing 76. Create a dialog with a list box to select one item.

```

REM Create a dialog with a list box.
REM Populate the list box with the data and return the selected item.
Function DialogSelectItem(oData(), sTitle$) As String
    Dim oDlg          'The created dialog.
    Dim oDlgModel      'The created dialog's model.
    Dim oModel         'Model for a control.
    Dim oControl       'References a control.
    Dim iTabIndex%     'The current tab index while creating controls.
    Dim s$             'Generic temporary string variable.

    DialogSelectItem() = ""

    REM Create the dialog's model
    oDlgModel = CreateUnoService("com.sun.star.awt.UnoControlDialogModel")
    oDlgModel.PositionX = 50 : oDlgModel.PositionY = 50
    oDlgModel.Width = 350 : oDlgModel.Height = 300
    oDlgModel.Title = sTitle$

    iTabIndex = 0
    s = "com.sun.star.awt.UnoControlListBoxModel"
    oModel = oDlgModel.createInstance(s)

```

```

oModel.Name = "ListBox"
oModel.TabIndex = iTabIndex
oModel.PositionX = CInt(10) : oModel.PositionY = 10
oModel.Width = 350 - 20 : oModel.Height = 300 - 60
oModel.Dropdown = False
oModel.MultiSelection = False
oDlgModel.InsertByName("ListBox", oModel)
iTabIndex = iTabIndex + 1

s = "com.sun.star.awt.UnoControlButtonModel"
oModel = oDlgModel.CreateInstance(s)
oModel.Name = "OKButton"
oModel.TabIndex = iTabIndex
oModel.PositionX = CInt(350 / 2 - 75)
oModel.PositionY = 300 - 20
oModel.Width = 50
oModel.Height = 15
oModel.Label = "OK"
oModel.PushButtonType = com.sun.star.awt.PushButtonType.OK
oDlgModel.InsertByName("OKButton", oModel)
iTabIndex = iTabIndex + 1

oModel = oDlgModel.CreateInstance(s)
oModel.Name = "CANCELButton"
oModel.TabIndex = iTabIndex
oModel.PositionX = CInt(350 / 2 + 25)
oModel.PositionY = 300 - 20
oModel.Width = 50
oModel.Height = 15
oModel.Label = "Cancel"
oModel.PushButtonType = com.sun.star.awt.PushButtonType.CANCEL
oDlgModel.InsertByName("CANCELButton", oModel)
iTabIndex = iTabIndex + 1

REM Create the dialog and set the model
oDlg = CreateUnoService("com.sun.star.awt.UnoControlDialog")
oDlg.setModel(oDlgModel)

REM Add the items.
oDlg.getControl("ListBox").addItems(oData(), 0)

REM Create a window and then tell the dialog to use the created window.
Dim oWindow
oWindow = CreateUnoService("com.sun.star.awt.Toolkit")
oDlg.createPeer(oWindow, null)

Dim i%
REM Finally, execute the dialog.
i = oDlg.execute()
If i = 0 Then
    Exit Function
End If

```

```
    DialogSelectItem() = oDlg.getControl("ListBox").getSelectedItem()  
End Function
```


13. Database utility macros

13.1. Quoting table and field names

If a field name contains a space or lower case letters, then the field name must be quoted. The character used for quoting changes depending on the database in question. Common quote characters include a back-tic (`), single quote ('), or double quote ("). The precise character to use is obtainable from the connection meta-data.

Listing 77: Quote a table or field name in a SQL statement.

```
REM The quote charcter changes based on the database that is accessed.
Function DBQuoteName(sName As String, oCon) As String
    Dim sQuote As String
    sQuote = oCon.getMetaData().getIdentifierQuoteString()
    DBQuoteName = sQuote & sName & sQuote
End Function
```

13.2. Convert between an UNO Date and a Basic Date

The OOo database engine returns date objects as an UNO Date structure (see *Table 52*). Basic, on the other hand, has a date type based on a floating point number so the types are not directly compatible.

Table 56. Properties of the com.sun.star.util.Date structure.

Type	Property	Description
unsigned short	Year	year.
unsigned short	Month	Month of year (1-12 or 0 for a void date).
unsigned short	Day	Day of month (1-31 or 0 for a void date).

Listing 78 demonstrates how to convert between the Date type used internally by Basic, and the UNO structure returned by database calls.

Listing 78. The date routines are contained in the document SC13.

```
Function UNODateToDate(ByVal x) As Date
    If IsNull(x) OR IsEmpty(x) Then
        UNODateToDate = cDate(0.0)
    ElseIf x.Day = 0 OR x.Month = 0 Then
        UNODateToDate = cDate(0.0)
    Else
        UNODateToDate = DateSerial(x.Year, x.Month, x.Day)
    End If
End Function

Function DateToUNODate(ByVal x As Date)
```

```

    Dim oDate As New com.sun.star.util.Date
    SetUnoDate(x, oDate)
    DateToUNODate = oDate
End Function

```

```

Sub SetUnoDate(ByVal x As Date, oDate)
    oDate.Day = Day(x)
    oDate.Month = Month(x)
    oDate.Year = Year(x)
End Sub

```

The OOO database engine returns time data as an UNO Time structure (see *Table 57*). The Time structure contains values accurate to hundredths of a second. Basic is able to store times accurate hundredths of a second, but does not provide a function to extract the value (see *Listing 79*).

Table 57. Properties of the com.sun.star.util.Time structure.

Type	Property	Description
unsigned short	Hours	Hours (0-23).
unsigned short	Minutes	Minutes (0-59).
unsigned short	Seconds	Seconds (0-59).
unsigned short	HundredthSeconds	Hundredth seconds (0-99).

Listing 79. Convert UNO Time to a Basic Date

```

Function UNOTimeToDate(ByVal x) As Date
    If IsNull(x) OR IsEmpty(x) Then
        UNOTimeToDate = cDate(0.0)
    Else
        UNOTimeToDate = TimeSerial(x.Hours, x.Minutes, x.Seconds) + _
            x.HundredthSeconds / 8640000
    End If
End Function

Function DateToUNOTime(ByVal x As Date)
    Dim oDate As New com.sun.star.util.Time
    SetUNOTime(x, oDate)
    DateToUNOTime = oDate
End Function

Sub SetUNOTime(ByVal x As Date, oDate)
    Dim d As Double
    Dim i As Integer
    oDate.Hours = Hour(x)
    oDate.Minutes = Minute(x)
    oDate.Seconds = Second(x)

    REM Remove any date portion. Int() rounds towards negative infinity,
    REM which works properly, even for dates prior to December 29,1899.
    d = x - Int(x)

```

```

REM Now, convert to seconds. There are 86,400 seconds in a day.
d = d * 86400
d = d - Int(d)

REM Now, convert to hundredths of seconds. Use CInt to round to the
REM nearest integer rather than round towards negative infinity.
i = CInt(d * 100)
If i > 99 Then i = 99
oDate.HundredthSeconds = i
End Sub

```

A database time stamp is returned as a `DateTime` structure, that contains the properties in *Table 56* and *Table 57*. The flexibility offered by Basic allows the use of the routines in *Listing 78* and *Listing 79*, which provides for very simple routines to convert between the Basic Date type and the UNO `DateTime` structure (see *Listing 80*).

Listing 80. Convert UNO DateTime to a Basic Date

```

Function UNODateTimeToDate(ByVal x) As Date
    UNODateTimeToDate = UNODateToDate(x) + UNOTimeToDate(x)
End Function

Function DateToUNODateTime(ByVal x As Date)
    Dim oDate As New com.sun.star.util.DateTime
    SetUNOTime(x, oDate)
    SetUNODate(x, oDate)
    DateToUNODateTime = oDate
End Function

```

13.3. Convert a result set to an array of data

A result set contains rows of data. Each row of data contains columns with data. A result set allows a large dataset to be returned, even if it will not all fit into memory at one time—the macro presented in this section assumes that the data returned by the result set is not too large to fit into memory. The purpose is to provide a simple method to obtain all of the data from a query, formatted for use by `AppendDataToCalcDoc()` in *Listing 67*; complete with column headers.

A result set provides meta-data, which describes the data that it contains. The result set meta-data indicates the number of columns in the result set along with a description of each column. The meta-data is used to determine the number of columns and the column titles. The macro in *Listing 81* generates a data array that encapsulates the data based on its type.

Listing 81. Dump the results from a ResultSet into an array.

```

Sub ResultSetToData(oResult, oDoc, sFunc$, oData())
    Dim oData2() 'Temporary row data array.
    Dim oMeta 'Each result set also has meta-data.
    Dim n As Long 'Number of columns returned by a result set.
    Dim i As Long 'General index variable.
    Dim nRowCount& 'Count of the number of rows.
    Dim x 'Generic work variable.

    If IsNull(oResult) OR IsEmpty(oResult) Then
        oData() = Array(Array(sFunc, "NULL result set returned"))
    End If
End Sub

```

```

Exit Sub
End If

oData() = Array()
Do While oResult.next()
  If n = 0 Then
    REM Add column titles.
    oMeta = oResult.getMetaData()
    n = oMeta.getColumnCount()
    oData2() = DimArray(n)
    oData2(0) = sFunc
    For i = 1 To n
      oData2(i) = oMeta.getColumnName(i)
    Next
    REM Advanced programming concept....
    REM Do not replace the next two lines with:
    REM      oData() = Array(oData2())
    REM because it would use a reference to oData2().
    REM AppendToArray() copies the array by value.
    AppendToArray(oData(), oData2())
  End If

  REM The first column is always the function name.
  oData2() = Array()
  oData2() = DimArray(n)

  REM Although the row number should be available using
  REM oResult.getRow(), this is not always the case.
  nRowCount = nRowCount + 1
  oData2(0) = sFunc & " " & nRowCount
  For i = 1 To n
    Select Case oMeta.getColumnType(i)
    Case com.sun.star.sdbc.DataType.DATE
      oData2(i) = CStr(UNODateToDate(oResult.getDate(i)))
    Case com.sun.star.sdbc.DataType.TIME
      x = oResult.getTime(i)
      oData2(i) = Format(x.Hours, "00") & ":" & _
        Format(x.Minutes, "00") & ":" & _
        Format(x.Seconds, "00") & "." & x.HundredthSeconds
    Case com.sun.star.sdbc.DataType.TIMESTAMP
      x = oResult.getTimestamp(i)
      oData2(i) = CStr(UNODateToDate(x)) & " " & _
        Format(x.Hours, "00") & ":" & Format(x.Minutes, "00") & _
        ":" & Format(x.Seconds, "00") & "." & x.HundredthSeconds
    Case Else
      oData2(i) = oResult.getString(i)
    End Select
  Next
  AppendToArray(oData(), oData2())
Loop

REM In case no rows are returned...

```

```

If n = 0 AND nRowCount = 0 Then
    oData() = Array(Array(sFunc, "No rows returned"))
End If
End Sub

```

The macro in *Listing 81* inspects the data type by using the `DataType` constants, which are based on the `java.sql.Types` constants (see *Table 58*).

Table 58. Column types defined by the `com.sun.star.sdbc.DataType` constants.

Value	Name	Comment
-7	BIT	One or more bits.
-6	TINYINT	Typically an 8 bit signed integer.
-5	BIGINT	Typically a 64 bit signed integer.
-4	LONGVARBINARY	Variable length binary data, such as an image.
-3	VARBINARY	Variable length binary data, such as an image.
-2	BINARY	Fixed length binary data.
-1	LONGVARCHAR	Variable length character data.
0	SQLNULL	NULL value.
1	CHAR	Fixed length character data.
2	NUMERIC	Long integer with a specified precision.
3	DECIMAL	Long integer with a specified precision.
4	INTEGER	Typically a 32 bit signed integer.
5	SMALLINT	Typically a 16 bit signed integer.
6	FLOAT	Typically a single or double precision floating point number.
7	REAL	Typically a single or double precision floating point number.
8	DOUBLE	Typically a single or double precision floating point number.
12	VARCHAR	Variable length character data.
91	DATE	Date.
92	TIME	Time.
93	TIMESTAMP	Date and time accurate to seconds or milliseconds depending on the database. HSQLDB is accurate to milliseconds..
111	OTHER	Database-specific type, mapped to an object that is accessed using the method <code>XRow::getObject()</code> .
2000	OBJECT	A type represented by an object which implements this type.
2001	DISTINCT	Describes a type based on a built-in type; a user-defined data type (UDT).
2002	STRUCT	Attributes that may be any type; a user-defined data type (UDT).
2003	ARRAY	SQL ARRAY.

Value	Name	Comment
2004	BLOB	SQL Binary Large Object.
2005	CLOB	SQL Character Large Object.
2006	REF	SQL referencing type.

The `AddResultSetToDoc()` method in *Listing 82* copies a result set to a data array and then appends the data to a Calc document using the macros in *Listing 67* and *Listing 81*. It is assumed that the database is not exceedingly large and that all of the data will fit in memory.

Listing 82. AddResultSetToDoc is found in the document SC13.

```
Sub AddResultSetToDoc(oResult, oDoc, sFunc$)
    Dim oData() 'Primary data array added to the Calc document.
    ResultSetToData(oResult, oDoc, sFunc$, oData())
    AppendDataToCalcDoc(oDoc, oData())
End Sub
```

13.4. Create and populate a dialog from a result set

The macro in *Listing 83* accepts a result set, a column number and a maximum number of rows. A dialog with a list box displays the data from the specified column (see *Listing 76*). The single selected item is returned as a string. The list box is a standard UNO control is not actually tied to the database. In other words, the data is extracted manually.

Listing 83. Populate and use a list box from a result set.

```
Function SelItemFromResult(oResult, nCol%, nMaxRows&) As String
    Dim nCount As Long
    Dim oData() As String
    Dim s$

    SelItemFromResult = ""

    REM First, check to see if there is data to read.
    If nMaxRows < 1 Then Exit Function
    If IsNull(oResult) OR IsEmpty(oResult) Then
        Exit Function
    End If

    REM Now read the data from the database.
    nCount = 0
    ReDim oData(0 To nMaxRows - 1) As String
    Do While oResult.next() AND nCount < nMaxRows
        oData(nCount) = oResult.getString(nCol)
        nCount = nCount + 1
    Loop
    ReDim Preserve oData(0 To nCount - 1) As String
    SelItemFromResult() = DialogSelectItem(oData(), "Select Item")
End Function
```


14. Tips and tricks

At least initially, tips and tricks really means “things that Andrew always forgets”.

14.1. Limit the number of returned records

Every database does this differently. In a native Base document, use the following syntax:

```
!select limit 0 100 <cols> from <tables>
select limit 0 4 * from TABLE1 'Rows 0, 1, 2, and 3
select limit 4 4 * from TABLE1 'Rows 4, 5, 6, and 7
```

Notice that the rows are numbered starting with zero. Be careful, or you will miss a row. Second, you can not use this syntax in the GUI because the parser will not recognize the SQL. This syntax is specific to the database embedded in an OOo Base document.

Use Select Limit The **Use Select Limit** button asks you to choose a database. The DATA table is created in this database and data is added. The LIMIT statement is then used in two queries to display the result.

Listing 84. Create a simple database and then use LIMIT.

```
Sub UseSelectLimit(Optional dbURL$ = "")
    Dim sTableName$      'The name of the table to creat.
    Dim oTable           'A table in the database.
    Dim oTables          'Tables in the document
    Dim oTableDescriptor 'Defines a table and how it looks.
    Dim oCols            'The columns for a table.
    Dim oCol             'A single column descriptor.
    Dim oCon             'Database connection.
    Dim oBaseContext     'Database context service.
    Dim oDB              'Database data source.
    Dim oStatement
    Dim sSQL$

    If dbURL = "" Then dbURL = ChooseAFile(OOoBaseFilters(), False)
    If dbURL = "" Then Exit Sub

    CreateBinaryDB(dbURL$, False)

    REM Use the DatabaseContext to get a reference to the database.
    oBaseContext = CreateUnoService("com.sun.star.sdb.DatabaseContext")
    oDB = oBaseContext.getByName(dbURL)
    oCon = oDB.getConnection("", "")
    oTables = oCon.getTables()

    sTableName$ = "DATA"
    If oTables.hasByName(sTableName$) Then
        oTables.dropByName(sTableName)
        oDB.DatabaseDocument.store()
    End If
```



```

REM For now, this should always be True
If NOT oTables.hasByName(sTableName$) Then
    oTableDescriptor = oTables.createDataDescriptor()
    oTableDescriptor.Name = sTableName$

    oCols = oTableDescriptor.getColumns()
    oCol = oCols.createDataDescriptor()
    oCol.Name = "ID"
    oCol.Type = com.sun.star.sdbc.DataType.INTEGER
    oCol.IsNullable = com.sun.star.sdbc.ColumnValue.NO_NULLS
    oCol.IsAutoIncrement = True
    oCol.Precision = 10
    oCol.Description = "Primary Key"
    oCols.appendByDescriptor(oCol)

    oCol.Name = "NAME"
    oCol.Type = com.sun.star.sdbc.DataType.VARCHAR
    oCol.Description = "Filename"
    oCol.Precision = 255
    oCol.IsAutoIncrement = False
    oCols.appendByDescriptor(oCol)

    oTables.appendByDescriptor(oTableDescriptor)
End If

sSQL = "insert into DATA (NAME) values (?)"
oStatement = oCon.PrepareStatement(sSQL)
Dim x()
Dim i%
x() = Array("zero", "one", "two", "three", "four", _
    "five", "six", "seven", "eight", "nine", "ten")
For i = LBound(x()) To UBound(x())
    oStatement.SetString( 1, x(i))
    oStatement.ExecuteUpdate()
Next

oStatement = oCon.CreateStatement()
Dim s$
Dim oResult
For i = 0 To 4 Step 4
    sSQL = "select LIMIT " & i & " 4 * from DATA"
    oResult = oStatement.executeQuery(sSQL)
    Do While oResult.next()
        s = s & "LIMIT " & i & " 4 Yields " & oResult.getString(1) & _
            " " & oResult.getString(2) & CHR$(10)
    Loop
Next
MsgBox s

REM Do not dispose the database context or you will NOT be able to
REM get it back without restarting OpenOffice.org.
REM Store the associated document to persist the changes to disk.

```

```
oDB.DatabaseDocument.store()  
oCon.close()  
End Sub
```

15. Connect to a Base document using JDBC

A Base document can connect to an external database or it can create an HSQLDB database that is stored inside of the Base document. OOo documents are stored as zip files and Base documents are no exception. John Ward created a solution in Java that extracts the HSQLDB files from the Base document (which means zip file) into a temporary directory and then connects to it using JDBC. All of John's comments and code are available in his Blog.

<http://digiassn.blogspot.com/2006/07/java-creating-jdbc-connection-to.html>

Listing 85. Access a Base document using JDBC.

```
import java.sql.*;
import java.util.zip.*;
import java.io.*;
import org.hsqldb.jdbcDriver;
import java.util.*;

public class Test {
    public static void main(String[] args) {
        //Instantiate the jdbcDriver from HSQL
        jdbcDriver j = new jdbcDriver();
        Connection con = null; //Database objects
        Statement com = null;
        ResultSet rec = null;
        ZipFile file = null; //For handling zip files
        ZipEntry ent = null;
        Enumeration en = null; //For the entries in the zip file
        BufferedOutputStream out = null; //For output from zip class
        InputStream in = null; //for reading buffers from the zip file
        File f = null; //Get temporary file name, not used for anything
        int len; //General length counter for loops
        //Stores list of unzipped file for deletion at end of program
        List v = new ArrayList();

        //Unzip zip file, via info from
        //http://www.devx.com/getHelpOn/10MinuteSolution/20447

        try
        {
            //Open the zip file that holds the OO.Org Base file
            file = new ZipFile("/home/digiassn/OODatabase/employeeDatabase.odt");

            //Create a generic temp file. I only need to get the filename from
            //the tempfile to prefix the extracted files for OO Base
            f = File.createTempFile("ooTempDatabase", "tmp");
            f.deleteOnExit();

            //Get file entries from the zipfile and loop through all of them
            en = file.entries();
            while (en.hasMoreElements())
            {
```

```

//Get the current element
ent = (ZipEntry)en.nextElement();

//If the file is in the database directory, extract it to our
//temp folder using the temp filename above as a prefix
if (ent.getName().startsWith("database/"))
{
    System.out.println("Extracting File: " + ent.getName());
    byte[] buffer = new byte[1024];

    //Create an input stream file the file entry
    in = file.getInputStream(ent);

    //Create a output stream to write out the entry to, using the
    //temp filename created above
    out = new BufferedOutputStream(new FileOutputStream("/tmp/" + f.getName()
+ "." + ent.getName().substring(9)));

    //Add newly created temp file to tempfile vector for deleting
    //later
    vt.add("/tmp/"+f.getName() + "." + ent.getName().substring(9));

    //Read the input file into the buffer, then write out to
    //the output file
    while((len = in.read(buffer)) >= 0)
    out.write(buffer, 0, len);

    //close both the input stream and the output stream
    out.close();
    in.close();
}
}
//Close the zip file since the temp files have been created
file.close();

//Create our JDBC connection based on the temp filename used above
con = DriverManager.getConnection("jdbc:hsqldb:file:/tmp/" + f.getName(),
"SA", "");

//Create a command object and execute,
// storing the results in the rec object
com = con.createStatement();
rec = com.executeQuery("select * from \"employees\"");

//GO through the resultset, and output the results
while (rec.next())
    System.out.println("Last Name: " + rec.getString("nm_emp_last") + " First
Name: " + rec.getString("nm_emp_first"));

//Close all the database objects
rec.close();
com.close();

```

```

        con.close();

        //Delete temporary files,
        // the file names are stored in the v vector
        for (len = 0; len <> v.size(); len++)
            (new File((String)v.get(len))).delete();
    }
    catch (Exception e)
    {
        e.printStackTrace();
    }
}
}

```

John's code does the following:

- Extract the HSQLDB files to a temp location (which you must change so that this will work for you)
- Create a JDBC connection to the temporary files.
- Extract values from the *nm_emp_first* and *nm_emp_last* columns.
- Delete the temporary files.

Appendix A. Stuff I Own

I decided to create a database of stuff that I own. The database has changed over time. I have added new fields, and there are some that I should probably remove. The end result is sort of a collection of stuff that works but is not polished. My intention is not to demonstrate a polished application, but rather, to show a few techniques that worked for me.

A.1. Tables

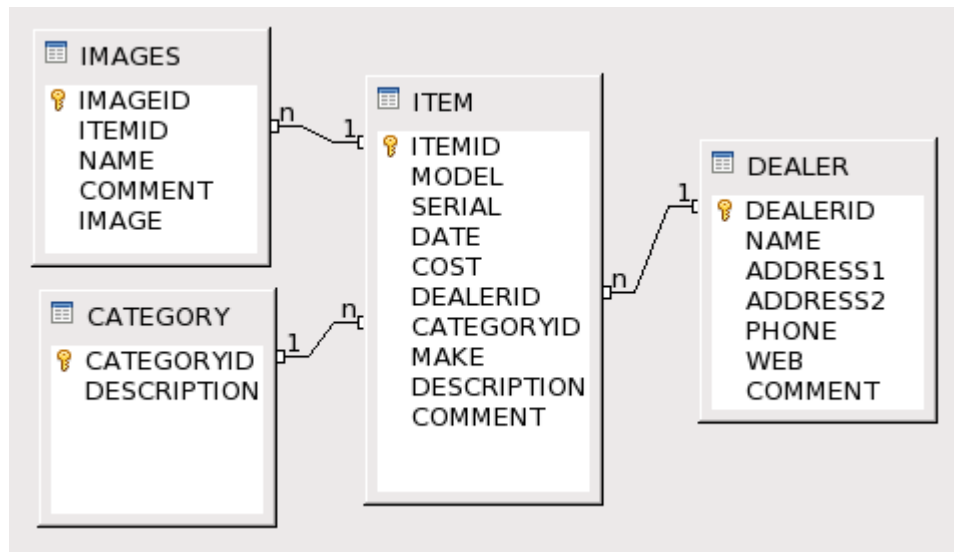


Figure 26. Vague schema of stuff I own.

Notice that table names and field names are in upper case; this is safer and avoids problems with certain types of database back-ends.

A.1.1. Category

I created arbitrary categorizations such as computers, electronics, and tools. It would be nice to have a hierarchy of categories – allow each category to have a parent. If I stated that wood working tools had a parent called tools, I would find hand wood working tools when I searched for tools. Unfortunately, this is notoriously difficult (and inefficient) to code in SQL. The usual substitute is to use a hierarchical naming convention. Sorry, no solutions on this today, I simply ignore the problem in my design.

A.1.2. Dealer

The dealer table tells me from whom I purchased an item. Very simple contact information and a comment field are used.

A.1.3. Images

A scanned image of a receipt, or a picture of the item. The Name field contains just the file name and does not include the path to the image. This is required because there is no other way to really understand the file type, other than looking at the file extension. As a binary field, it could contain just about any data.

The Comment field typically contains simple text such as “receipt”, so I know what the image is.

A.1.4. Item

What was I thinking when I created the Item table?

Field	Description
Make	Manufacturer such as Panasonic, BMW, or Wilson Combat.
Model	Model name and number.
Serial	Serial number of the unit.
Date	When did I buy it?
Cost	How much did I pay?
DealerID	From whom did I buy it?
Description	What is it?
Comment	Anything else such as “I like it”

It turns out that I have not followed my convention for make and model. It is usually just easier to enter the full make and model into a single field.

I use the description and comment fields for most anything – so perhaps I should have just one field. Usually, I mention the warranty, so I can just look in the database to see if the item is still under warranty.

A.2. Forms

The database evolved over time, so forms that I no longer use are still in the DB.

A.2.1. Item Two Tables

I think that I created the Item with two tables form using the wizard with the items table in the main form, and the images table in a the sub-form.

The form contains garbage characters to the right of the first table, this is a bug in either OOo, or in my video driver.

ITEMID	MODEL	SERIAL	DATE	COST	DEALERID	CATEGORYID	MAKE	DESCRIPTION
42	Seagate ST90G2414T	08/12/07	129.99		4	1	Seagate	5500GB Hard Drive
43	Panasonic (VN7KE0086	02/06/08	295		1	2	Panasonic	DVD/VCR Recorder
44	Canon VIXA 722622041	02/26/08	999		1	4	Canon	
45	Seagate Ba50D2HPW2	05/31/08	159.99		14	1		Firmware
46	UT44170 7 BML01801	06/01/08	39.97		5	7	Homelite	2 year warranty
47	Heath / ZenithScan	06/28/08	24.54		5	9	Heath / ZenithScan	270 degree
48	Bogen 1901		1000		1	4	Bogen	19" Tripod with
49	Buick LeSab	01/28/08	775.21		15	10		Tires and

IMAGEID	ITEMID	NAME	COMMENT
2	47	ZenithScan Receipt	

Figure 27. Items with two tables form.

The title-bar indicates the database, the table name, and the text (read-only). Read-only indicates that you can not edit the form in the current mode. You can, however, edit the data in the form.

Stuff_We_Own_20080531.odt : ITEM_TWO_TABLES (read-only) - OpenOffice.org Base: Form Design

The structure and relationship of the forms is very simple:

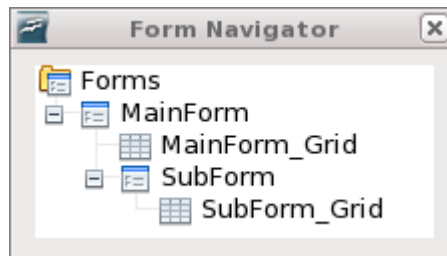


Figure 28. Form hierarchy.

The MainForm has content type Table, and the contents is ITEMS.

The SubForm has type Table, and the contents is IMAGES. Link Master Fields, and Link Slave Fields are both set to ITEMID, since that is the name in both tables. The stated links are clearly shown in Figure 26.

A few annoyances exist with this form.

1. Images are not shown, but you know that they are there.
2. You must set the category and dealer based on their numeric ID after manually finding an appropriate value.

The next form starts to deal with one issue.

A.2.2. Item One Table

The purpose for the item view with one table, was to figure out how to associate each record to the dealer and category. This form includes the item table in a grid similar to Figure 27. Below the grid are two list boxes.

The screenshot shows a software interface with a data grid at the top and two dropdown menus below it. The grid has columns labeled: ID, MODEL, DESCRIPTION, S/N, DATE, COST, DEALERID, COMMENT, CAT ID, and MAKE. Below the grid, there are two dropdown menus. The first is labeled 'Category :' and the second is labeled 'Purchased From :'. Both dropdown menus are currently empty.

Figure 29. Two list boxes.

This report has only one form. The category list box, therefore, is in the form, which is set to look at the item table. The control has the following general properties of interest:

- Set the Dropdown property to yes.
- Set Line count to 5 to limit the size.
- Set Multiselection to No.

The Data tab for the control properites is a bit more interesting.

- Set the Data field to CATEGORYID (from the item table).
- Set Type of list contents to Sql.
- Use the SQL in Listing 86 for the List content. This loads the list with the data from the Category table. Although the data is populated from the category table, the control is associated with a field from the items table.
- Set the bound field to 1.

Listing 86. SQL for the Category list box.

```
SELECT "DESCRIPTION", "CATEGORYID" FROM "CATEGORY" ORDER BY "DESCRIPTION"
```

As you select different rows in the grid, the values in the drop down list change to represent the data in the item table for the current record. Also, choosing a new value in the drop down, sets the associated category ID in the item table.

A.2.3. Item Fields

This form adds the image field, and buttons for adding, changing, and extracting images.

The screenshot displays a web-based form titled 'Item Fields'. The form is organized into several sections. At the top, there are fields for 'ITEMID' (47) and 'DATE' (06/28/08). Below these are fields for 'MODEL' (Heath / Zenith SL-5716-WH), 'SERIAL', 'COST' (\$24.54), 'DEALERID' (5), and 'CATEGORYID' (9). The 'DEALERID' and 'CATEGORYID' fields are accompanied by dropdown menus showing 'Home Depot' and 'Home and Garden' respectively. Further down, there are fields for 'MAKE' (Heath / Zenith SL-5716-WH), 'DESCRIPTION' (270 degree replacement motion sensor), 'COMMENT', 'Image ID' (2), and 'Item ID' (47). A 'File' field contains 'ZenithSensor.jpg'. At the bottom of the form, there is a 'Comment' field with the text 'Receipt'. To the left of the main form, there are four buttons: 'Add Image', 'Delete Image', 'Replace Image', and 'Extract Image'. Below the main form, there is a large image of a receipt from 'THE HOME DEPOT 0011'. The receipt includes the store address (5555 CLEVELAND AVENUE, COLUMBUS, OH 43229-6815), a date (06/28/08), a time (05:52 PM), a barcode, and a list of items including '016063871600 SEC LIGHTING -A-'. At the bottom of the receipt, there is a return policy statement and a website URL (www.homedepot.com/install). The entire form is set against an orange background.

Figure 30. Items with individual fields.

There are two internal forms, the main form contains the item information. The two drop-down list boxes are in the main item form, and they work and are configured in the same way as the Item One Table form. In other words, they perform a help and fill. The very bottom line contains a navigation bar for the first form.

The buttons to Add, Delete, Replace, and Extract an image are also in the main form.

The second form contains the image related fields. The image form also contains a navigation bar, which is used to navigate through the images.

A.3. Add an image macro

The Add Image button is associated to the AddImageButton macro. First, variables are declared, and required libraries are loaded. Notice that an event argument is passed.

Listing 87. Add an image to a form and database.

```
Sub AddImageButton(oEvent)
    Dim oImagesForm
    Dim oMainForm
    Dim nImage As Long
    Dim nImageName As Long
    Dim sFileName$
    Dim sNameOnly$

    REM Use some methods from the Tools library.
    If NOT GlobalScope.BasicLibraries.isLibraryLoaded("Tools") Then
        GlobalScope.BasicLibraries.LoadLibrary("Tools")
    End If
```

The button is contained in the main form. The form is, therefore, the button's parent. The following steps occur: (1) Get the main form, which is the button's parent. (2) Get the contained Images form, which is contained in the main form.

```
oMainForm = oEvent.Source.getModel().getParent()
oImagesForm = oMainForm.getByName("Images")
```

Ask the image form for the column number of the image field, and the name field.

```
nImage = oImagesForm.findColumn("IMAGE")
nImageName = oImagesForm.findColumn("NAME")
```

I want to insert an image into the database, but I can not do this unless the main form, which contains items, actually has a record. When you use the form to add a new item, the form moves to the insert row. I do not know how to tell if the cursor is on the insert row, but it looks like `getRow()` returns zero when this is the case. The following code checks the main fields in the form to see if any data has been entered. If no data exists, then the macro exits with a warning. If data is determined to be present (and we believe that we are on the “insert” row), then we call `insertRow()` to add the current data to the database.

```
If oMainForm.getRow() = 0 Then
    If IsEmpty(oMainForm.getByName("fmtITEMID").getCurrentValue()) AND _
        IsEmpty(oMainForm.getByName("datDATE").getCurrentValue()) AND _
        Len(oMainForm.getByName("txtCOMMENT").getCurrentValue()) = 0 AND _
        Len(oMainForm.getByName("txtDESCRIPTION").getCurrentValue()) = 0 AND _
        Len(oMainForm.getByName("txtSERIAL").getCurrentValue()) = 0 AND _
        Len(oMainForm.getByName("txtMODEL").getCurrentValue()) = 0 Then
        MsgBox ("The current row contains no data")
    Exit Sub
End If
oMainForm.insertRow()
```

```
End If
```

Although it may make sense to check for updated data and then update the row, I do not.

Next, use the ChooseAFile macro (see *Listing 58*), to select the file to input. The file without the pass is stored in sNameOnly, and it will be saved into the filename field.

```
sFileName = ChooseAFile$(GraphicFilters(), True)
if Len(sFileName) < 1 Then
    Exit Sub
End If
sNameOnly$ = FileNameoutofPath(sFileName, "/")
```

If the images form is on a current record, call updateRow() just in case a field has been updated in the form. In other words, save the data in case there is data to save. If I new how to check if the data was changed, I would.

```
If oImagesForm.isAfterLast() OR oImagesForm.isBeforeFirst() Then
    Rem No need to save!
Else
    Rem Save the current data
    oImagesForm.updateRow()
End If
```

Open the binary file.

```
Dim s$
Dim oSimpleFileAccess
Dim oStream
s$ = "com.sun.star.ucb.SimpleFileAccess"
oSimpleFileAccess = createUnoService(s$)
oStream = oSimpleFileAccess.openFileRead(sFileName)
```

Finally, use moveToInsertRow() to insert a new image record, update the image column using the stream to the image, update the file name without path for the image name column, and then tell the form to insert the row.

```
oImagesForm.moveToInsertRow()
oImagesForm.updateBinaryStream(nImage, oStream, oStream.getLength())
oImagesForm.updateString(nImageName, sNameOnly)
oImagesForm.insertRow()
End Sub
```

A.4. Delete an image macro

The Delete Image button is associated to the DeleteImageButton macro. The first portion is exactly like adding an image. Variables are declared, required libraries are loaded, and references to forms and columns are obtained.

Listing 88. Delete an image.

```
Sub DeleteImageButton(oEvent)
```

```

Dim oImagesForm
Dim oMainForm
Dim nImage As Long
Dim nImageName As Long
Dim sFileName$
Dim sNameOnly$

REM Use some methods from the Tools library.
If NOT GlobalScope.BasicLibraries.isLibraryLoaded("Tools") Then
    GlobalScope.BasicLibraries.LoadLibrary("Tools")
End If

REM Inspect oEvent.Source
oMainForm = oEvent.Source.getModel().getParent()
oImagesForm = oMainForm.getByName("Images")
nImage = oImagesForm.findColumn("IMAGE")
nImageName = oImagesForm.findColumn("NAME")

```

If there does not appear to be a record to delete, then exit. Otherwise, ask if the image should be deleted.

```

If oImagesForm.isAfterLast() OR oImagesForm.isBeforeFirst() Then
    Exit Sub
End If
Dim i%

i = MsgBox ("Delete current image?", 4, "Delete Image")
If i <> 6 Then
    Exit Sub
End If

```

Delete the current row, and then move to the next or previous record. Finally, reload the form.

```

oImagesForm.deleteRow()
If oImagesForm.isLast() Then
    oImagesForm.previous()
ElseIf NOT oImagesForm.next() Then
    oImagesForm.previous()
End If
oImagesForm.reload()
End Sub

```

A.5. Replace an image macro

The Replace Image button is associated to the ReplaceImageButton macro. The first portion is exactly like deleting an image. Variables are declared, required libraries are loaded, references to forms and columns are obtained, and if it looks like there is no image to delete, then exit the macro.

Listing 89. Replace an image.

```
Sub ReplaceImageButton(oEvent)
    Dim oImagesForm
    Dim oMainForm
    Dim nImage As Long
    Dim nImageName As Long
    Dim sFileName$
    Dim sNameOnly$

    REM Use some methods from the Tools library.
    If NOT GlobalScope.BasicLibraries.isLibraryLoaded("Tools") Then
        GlobalScope.BasicLibraries.LoadLibrary("Tools")
    End If

    REM Inspect oEvent.Source
    oMainForm = oEvent.Source.getModel().getParent()
    oImagesForm = oMainForm.getByName("Images")
    nImage = oImagesForm.findColumn("IMAGE")
    nImageName = oImagesForm.findColumn("NAME")

    If oImagesForm.isAfterLast() OR oImagesForm.isBeforeFirst() Then
        Exit Sub
    End If
```

Select a file that will replace the existing image. If no image is chosen, then exit the macro. If a file is selected, open a stream to the file, update the image, update the image name, and then update the row. The only difference between updating an image and adding a new image (for these steps), is that the row is updated rather than inserted.

```
sFileName = ChooseAFile$(GraphicFilters(), True)
if Len(sFileName) < 1 Then
    Exit Sub
End If
sNameOnly$ = FileNameoutofPath(sFileName, "/")

Dim s$
Dim oSimpleFileAccess
Dim oStream

s$ = "com.sun.star.ucb.SimpleFileAccess"
oSimpleFileAccess = createUnoService(s$)
oStream = oSimpleFileAccess.openFileRead(sFileName)

oImagesForm.updateBinaryStream(nImage, oStream, oStream.getLength())
oImagesForm.updateString(nImageName, sNameOnly)
oImagesForm.updateRow()
End Sub
```

A.6. Extract an image macro

The Extract Image button is associated to the ExtractImageButton macro. The first portion is exactly like deleting or updating an image. Variables are declared, required libraries are loaded, references to forms and columns are obtained, and if it looks like there is no image to delete, then exit the macro.

Listing 90. Replace an image.

```
Sub ExtractImageButton(oEvent)
    Dim oImagesForm
    Dim oMainForm
    Dim nImage As Long
    Dim nImageName As Long
    Dim sFileName$
    Dim sNameOnly$

    REM Use some methods from the Tools library.
    If NOT GlobalScope.BasicLibraries.isLibraryLoaded("Tools") Then
        GlobalScope.BasicLibraries.LoadLibrary("Tools")
    End If

    REM Inspect oEvent.Source
    oMainForm = oEvent.Source.getModel().getParent()
    oImagesForm = oMainForm.getByName("Images")
    nImage = oImagesForm.findColumn("IMAGE")
    nImageName = oImagesForm.findColumn("NAME")

    If oImagesForm.isAfterLast() OR oImagesForm.isBeforeFirst() Then
        Exit Sub
    End If
```

The image name is obtained from the database. The name was saved so that the file name and, perhaps more importantly, so that the file extension would be known. If there appears to be no path information in the file name, then use the working directory.

```
sFileName = oImagesForm.getString(nImageName)
If InStr(sFileName, "/") = 0 AND InStr(sFileName, "\") = 0 Then
    REM Does not contain any path components
    sFileName = GetWorkDir() & "/" & sFileName
End If
```

Select a target file name to save the file. The original name is used as the default value. If no file is selected, then exit the macro.

```
sFileName = ChooseAFile$(GraphicFilters(), False, sFileName)
if Len(sFileName) < 1 Then
    Exit Sub
End If
```

Obtain a stream from the database, and stream the file to disk.

```
Dim s$
Dim oSimpleFileAccess
Dim oStream

s$ = "com.sun.star.ucb.SimpleFileAccess"
oSimpleFileAccess = createUnoService(s$)
oStream = oImagesForm.getBinaryStream(nImage)
oSimpleFileAccess.writeFile(sFileName, oStream)
End Sub
```

A.7. Clean the database

I copied my original database and I wanted to place a sample on my web site. First, however, I wanted to make certain that deleted records were really removed.

Use **Tools > SQL** to open the SQL window. Issue the command “CHECKPOINT DEFRAG” and then save and close the database file.

A.8. Things to do

There are so many things to do, I just do not have the time.

Add the ability to update the dealer and category using a form. This should be callable from the main form. For now, I manually edit the table – yuck!