Data Migration Assistant Help

Sample for download available at no charge here: http://www.informedcorp.com/Access-Development-developers-page.html

The intent of this tool is to allow developers to have an easy way of field upgrading existing customer database installations without having to retrieve their databases and do it for them. The assumption is that a split architecture system is used, meaning that code and other objects (forms, reports, queries) all reside in one database and that a back end database consisting of tables only is used via linkage of the tables to the front end.

In my experience, the most common "upgrade" has been to distribute a new front end but that as the pace of development increases, more often than not, major upgrades (as opposed to bug fixes) require changing the table structure of the back end (adding new tables, fields and relationships). Until now, the process of upgrading our customers required us to ask the customer to:

1. Return their existing back end to us.

2. Wait until we could "migrate (move)" their data into the newer version of the application by moving their data from the older back end into the newer one.

3. Install and license a new front-end/back-end pair.

Otherwise, several issues could be encountered, depending on the changes made:

1. Users would attempt to open an older back-end with a newer front-end and the automated linkage system we use would fail when it could not find a required table.

2. Users would attempt to open a newer back-end with an older front-end and not attach to all required tables.

3. Time was consumed communicating our needs (please zip up and email your back-end, etc.). The time cost on my end is/was the biggest driver to develop this system over the years.

Our sample database, Data Migration Assistant, demonstrates a solution to this problem that reduces the time and complexity of the standard upgrade process considerably. It is provided free of charge as a courtesy to developers and our way of thanking the large number of contributor's to Access help forums who help others with no repayment. As I look back, I realize that without the answers and help I obtained over the years, there is no way I could have hoped to do the things I did.

This database demonstrates our newest approach to upgrades and that approach reduces the process to a simple, 2-step process:

1. Rename/Move: User (or IT personnel) installs and licenses a new front-end/back-end pair after moving or renaming their current pair. We do not change the names of these files from version to version since that requires downloading and running very large installer (setup.exe).

2. Migrate Data: User activates a function in our software that allows browsing to the older back-end file and selecting it. Once this is done, the system moves all the data from the older back-end to the new-back-end. In practice, we can document this and/or walk them through it in a webinar.

System Overview (How it Works)

The MigrateData.accdb file simulates a front-end and should be opened first. It will automatically determine if the back-end is linked and, if the location has changed, prompt you to select a back-end database. For this example, select MigrateDataNewBackEnd.accdb to get started.

Form frmMigrateData should open automatically. It shows a list of linked tables the database is attached to and this list is taken from table TableOfTables. When you click the Migrate Data button, you're prompted to select the "old" back-end file, for this example, select MigrateDataOldBackEnd.accdb and the following happens:

1. Code loops through the tables recorded in table TableOfTables in the reverse of the sequence determined by the sequence number in the table and in this loop, two things are done:

    A. The corresponding table in the old back-end is linked (with the name of the table prefaced as "Imported" so there are two sets of tables linked, one from the new back-end with its original name, such as "Employees" and the corresponding table in the old back-end, named "ImportedEmployees".

    B. All data is deleted in the new back-end tables. This removes extraneous data such as sample data or data left in extraneously in the course of development.

2. In the same function, code loops through the tables again, this time in the sequence order listed (not the reverse order as in the first loop and two things occur:

    A. Data is "moved" from the old back-end tables into the new back-end tables. The code that "moves" the data handles attachments in attachment fields (and that was the toughest part of this project for me) since Microsoft kindly failed to provide an easy and natural way to do it via queries as with ALL OTHER DATA TYPES. Thanks Microsoft we developers really appreciate easy-to-use "features" such as this, especially without clear warnings, and why the hell didn't you provide code like this to do the job?

    B. The old back-end database table linkages are removed.

Details for Setting up a Similar System for Your Own Use

Note that all this work is provided at no cost and with no warranty of fitness for use in any particular situation. As a developer, I can provide only a small amount time to help at no charge. My standard

rates are $105/per hour with a minimum charge of 15 minutes. That said, if you need a small amount of help at no charge, my email address is claude.berman@informedcorp.com and I'll let you know if additional charges will be incurred in advance. Initially, I suspect some problems due to weaknesses in this documentation and will improve the documentation at no additional charge, but that will be at my sole discretion. I may also update the application from time to time.

Realize this is a fairly complete solution pieced together over the years with paid and unpaid help. I'm particularly grateful for the paid support over the years provided by Sagekey (www.sagekey.com) and I've found them to be extremely proficient and reputable. Their support time cost is reasonable given the value of their work. They can be trusted to develop this solution further and, in fact, a good bit of the linkage code was developed by them on a paid basis so they'll recognize it.

For the first pass, I recommend following this simple procedure:

0. Ensure you have a clean back up copy of your application both the front-ends and the back-ends. (Tip: I use the services of Mozy.com for automatic, online and local backups and recommend them.) Import the included forms and modules into your own application front end. I do apologize that there are many extraneous bits of code unrelated to the issue at hand; however I currently don't have the time to clean this up. Obviously, for testing, you'll need something to simulate an "old" back-end, and personally, I'd use an exact copy of the current "new" back-end.

1. Delete the sample data in TableOfTables

2. Populate TableOfTables with a list of your currently linked tables via the button on form frmMigrateData

3. In the table TableOfTables directly, or in the subform on frmMigrateData set the sequence of the operation. Unrelated tables can be in any position in the sequence. One-to-many related tables should be in many first and one side later in the sequence. The only other fast rules are that the sequence should have no duplicate numbers and no gaps (although I haven't tested the latter).

4. Ensure the Boolean "Include" is checked for any linked tables you need to process.

5. List the exact name of any attachment fields (only one per table can be handled properly) or the default field name "No Attachment".

6. Run the code using the **Migrate Data** button and see what happens. You should be prompted to select the old back-end file and the rest of the process should proceed automatically. Since an exact duplicate of the data will be present if successful, you should change some data in the "old" back end to ensure it works as intended. The debug window will tell you if something failed.

Other functionality:

1. You can play around with the **Check Employee Table Populated** button initially. It is a quick way to check if data moved from the back-end to the front end (particularly if you change something in the back-end table "Employees" prior to migration. It also demonstrates the functionality of the function

MoveData in the module basRecordHandling. This function is called in the loop mentioned above during the data migration process.

2. The **Help** button shows a simple way to offer users help on a per-Form basis. The on-click event of the button is set to =callhelp() and this function searches the table HelpTbl for the name of the form and if a match is found for the activeform it opens frmHelp with the correct data displayed. I open the form after setting up the basics and then copy and paste the help I've written in MS Word. Formatting is mostly preserved although I turn off auto-numbering since it doesn't "translate" well into Rich Text format used by the memo field in the underlying table.

3. The scrolling message pane was developed as part of an Excel automation project – it works pretty well here, I think. I clear the memo field when the form is closed, to avoid issues hitting the 64k character limit of memo fields. In the other project, I wanted to preserve as much history as possible, so I had to take steps to warn the user that space was running out and then automatically clear the field at when a certain set point of characters were reached. If you use something like this actively, it's surprising how quickly space is consumed.

4. Do you have trouble remembering the syntax for dCount and similar functions, especially with criteria? I do, my brain age and over-fill with useless data makes it really hard to remember details. Well a gentleman named Allen Browne came to the rescue with a form that does the heavy lifting for you and it's saved me a lot of time. His work is embodied in form GeneralFunctionTest and it makes this job easy by translating query syntax into code-compatible syntax (and why Microsoft had to have two different syntax forms in the same application is beyond me). To use the form, create a query with the tables and criterion needed. Then, switch the query to SQL view and paste that into the upper portion of the form. Then, click the Build SQL button on the form and the needed conversion is available in the lower pane. Thank you Allen, thank you.

Limitations:

1. I never had occasion to add more than one attachment field and the system is limited to one per table. I suppose one could add additional attachment field name columns and add an additional variable to the function to handle that.

2. Testing: I haven't worked with this extensively, so there may be bugs. This is a beta and may remain a beta forever. Beta stands for "**B**ugs **E**xist **T**o **A**nnoy", and here in the New Hampshire woods we have plenty of bugs. I'll be replacing about 90 append queries and 90 delete queries with this code over the next few days and will test against large data sets once that is done.

Contents of Package

Forms

frmHelp
frmDataMover
frmDataMoverSub1

frmDataMoverSub2
frmMigrateData (main working form)
frmMigrateDataSub1
frmMigrateDataSub2
GeneralFunctionTest

Tables (essential)

TableOfTables: (needed for functions to work)
Table_Field_Definitions: this will be created automatically in the course of operation of function MoveData, replacing the existing copy. You can safely delete it in code if you don't like temp tables hanging around in the front end.
HelpTbl:
tblMsg: Recordsource of form frmMigrateData and only used for the message pane memo field
Additional Tables: Various sample tables, linked and local

Queries (essential)

qryRecordTableNames: Used in the auto-population of TableOfTables. Note the use of tempvars here and in some of the other queries as dynamic criteria and append sources.
qryDeleteFromTarget: Only used for the example with form frmDataMover
qryFieldName
qryMigrateData00: Used in first loop iteration as recordset
qryMigrationData01: Used in second loop iteration
qryRecordSource1: Subform recordset for frmDataMoverSub
qryRecordSource2: Subform recordset for frmDataMoverSub2

Reports

Help

Modules *too many to list, most are only there for one or two functions

basRecordHandling: Has Function MoveData(FromTableName As String, ToTableName As String, Optional AttachmentFieldName As String = "No Attachment", Optional IgnoreAttachment As Boolean = True) ' this does most of the heavy lifting.

Import: Has Function MigrateData(OldDataBase As String) 'takes the path and name of the old database from which we want data

These two are the most important and relevant to this example and discussion. Additional functions and subs are called as needed.

Comments and Notes:

1. Via “versioning” (internally identifying) both the front end and the back end, the upgrade process could be reduced in complexity further since the front end could detect that the user is attempting to attach to an old back-end and automatically prompt the user for a newer one.

2. There are other approaches and I last tried the ambitious process of automatically upgrading an older back end via code. But, given the pace of development and extensive changes we’re doing, it seemed the developer (me) would have to spend too much time recording the changes made and coding to automate the upgrade. I’d consider purchasing a commercial system if someone offered it but, given this current system’s capabilities, I have a hard time justifying the cost. I assume the cost would have to be substantial since this would require quite a bit of work. I know, as I tried to develop something similar in the past.

3. Just a tip: Access developers use the terms front-end and back-end frequently, but I’ve found that IT personnel generally don’t know the meaning of these terms. They do understand the concept, but the terminology isn’t always clear to them. It is not an Access-only phenomenon as developers using other systems (SQL Server, MySQL, etc.) do use the terms, but I’ve found I can’t count on the IT person to immediately understand the terms. End-users often have trouble as well. Consequently, I’ve settled on calling my front-end programfile.accde and the back-end datafile.accdb. This allows use of the readily understood command, Open Data File, and customers seem to more easily learn to use it when told to look for something called “datafile”.

4. In moving attachments, the initial SaveToDisk and LoadFromFile methods in the inner (attachment) loop in function MoveData were very slow. Albert Kallal worked out a direct transfer method and that was incorporated into Rev 3, released on June 25, 2013.

5. Bonus Tip (or at least good for a smirk): For years, in the code window I used copy and paste and never noticed the work “definition” there on the right-click menu. So, whenever I wanted to find a procedure or function I encountered, I used the Edit > Find function, bopping through every place in code the word was used until I hit the origin, and often over-shooting it. Then, almost by accident, I discovered that by simply placing the cursor in the function or procedure word, the “definition” would instantly transport one to it. Duh, after decades, I finally notice this… And, here’s another one, at least in Win 7, the old three-finger salute is the hard way since one has to click through a sub-menu. Instead, just right-click on the start menu bar and pick “Start Task Manager”.