Automating DSN-Less connection to SQL Server

PJ Bryant

Corylus Business Systems

pbryant@corylus-business.co.uk

My background

- Began with a 50 step programmable calculator in the mid 70's
- Early 80's studied CompSci working on mainframe/mini development (punched cards and green screens) – Pascal, Assembler, COBOL, Algol68R, Fortran
- Started work early 80's mainframe COBOL development
- Mid 80's - the PC arrived in business
- Used Windows since v2.x, Access since the 1st beta (this does not, necessarily, make me an expert!)
- 25+ years of IT Management dabbling in code for internal needs
- Self-employed for nearly 20 years, 3 primary areas:
  - Turn data into information - long term (pseudo-)agile/prototyping projects
  - Build new systems to replace rubbish or outmoded system
  - Take over maintenance
- That's my excuse list finished ☺

Agenda

- Assumptions (versions)
- Infrastructure
- Establishing first set of connections (use a DSN)
- Backend tables required
- Features on the form
- Goals
  - Never relink a database on a client site again
  - Always send them a compiled front end that they “just” run
  - Avoid endless DSN creation and management.

Assumptions & Infrastructure

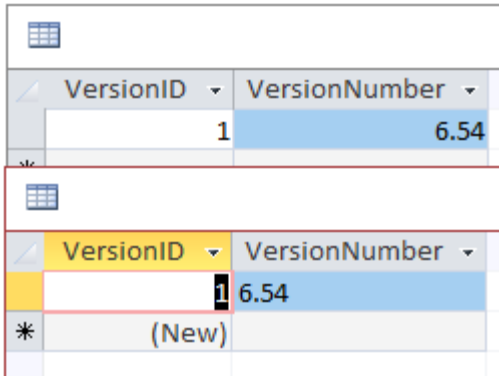
- Access 365 – demo code will be 32bit, but 64bit is fine
- You know how to setup a DSN connection to SQL
- SQL Server 2008R2 and above *seem* to work
  - I generally use SQL Server 2019 Express and above these days
  - SSMS 18.x works for me
- I've been doing this since Access 2007/SQL2008, so far no version change (Access or SQL) has stymied me
- N.B. This relies on trusted connections to the SQL Server (usually the norm for my sites), so you do not embed passwords into the code. If you use SQL Roles, or SQL Authentication, you'll need to handle that in the reconnection code (and you will see where).
- I must emphasize that for the SME environment trust is not a "everything must be totally secure etc" world. There are compromises everywhere. You will need to apply this in context to your client/product/project.

Establish connections

- One-time connect to your backend tables using a DSN
- You only need one DSN for each SQL database, on one network (normally yours, and it is usually for just 1 database)
- It might be possible to avoid this, but I couldn't be ***** to find out. Anyway, creating and using the DSN checks you can see the data and get through the security of the database connection.

Tables you need (1/4)

- Version ID – local and backend



VersionID	VersionNumber
1	6.54

VersionID	VersionNumber
1	6.54
(New)	

- This is used to control schema changes. If you change the schema (or the front end dramatically) then increment the version number and the F/E will no longer connect.

Tables you need (2/4)

- lclNetworks – this is a list of valid domains on which the application can work, the last column is SQL Server Instance name (notice the Express edition requires the instance).

lclNetworks

NetworkID	NetBIOS_Domain	SQLServer
1	CORYLUS	BUSINESS1\SQLEXPRESS
2	Domain2	ServerName2
3	Domain3	ServerName3
* (New)		

- There is an element of embedded security this gives you – the compiled app can't be loaded and run on a domain that isn't yours or your client's ☺
- Until they hack the local table. I've considered internal data for this, but this means more work… ☺
- The table is the demo is a compromise structure (I'll explain)

Tables you need (3/4)

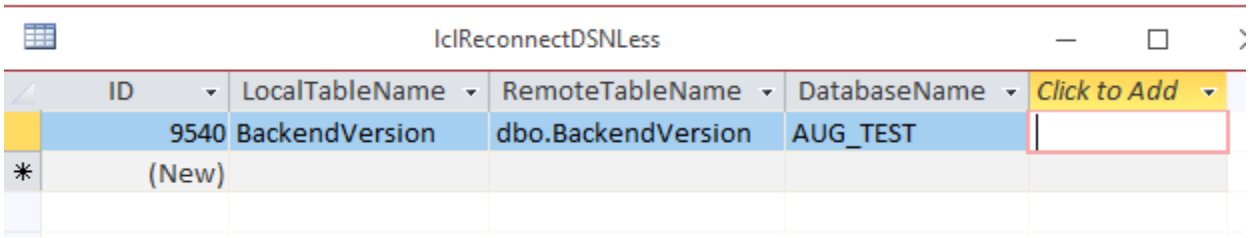
- Some users may reconnect the f/e to a different database. The lclDatabaseNames table stores them and a sort order for presenting them to the user
- In this instance I have CD and Vinyl in the demo app, but normally this would be standard set of names for any app - <appname>_TEST, _UAT and _LIVE.
- This allows the user (subject to version number) to switch between back ends
- I further secure this by only allowing administrative users to switch databases

lclDatabaseNames

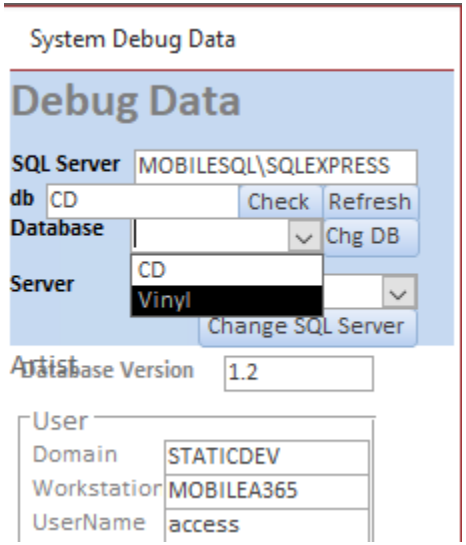
ID	DSNName	SortOrder	Click to Add
1	CD	1	
2	Vinyl	2	
(New)		0	

Tables you need (4/4)

- Reconnection information used to manage the DSN-less connection.
- Regard this as a temporary table, that you do not need to consider



Features on the debug form



- A list of valid database names to which you can switch.
- Three buttons – check my connection, refresh my connection, and change database
- Some static data on username, computer, network, default printer, and a couple of control functions
- A TEMPORARY dropdown and button to manually change servers. This is for DEMO only, this in normally done in code.

Demo Time

- Firstly a quick explanation of the SQL Server setup
  - This is a closed network of Hyper-V VM's, on a domain called STATICDEV, with a single DC and RRAS servers to connect to the outside world for updates, OneDrive, DropBox.
  - There are 2 SQL Servers (SQL2014 and SQL2019\SQLEXPRESS), on the network. One contains Marillion music data, the other King Crimson
  - Each SQL Server has 2 databases "CD" and "Vinyl" (although the latter is really "Non-CD"
  - All 4 databases have the same schema – 2 tables, BackendVersion and Records (as in music, not db)
- Access. The system start process is:
  - Use a default form that is hidden to load the system, and do "stuff"
  - Then optionally load a debug form for the user – this is controlled by user privilege on the system (and the domain on which it is running). Here I have hard coded it to loads every time for this domain.
  - Then load the main form
- OK…

Post demo notes

- The following slides summarise the points that I raised (or should have raised) about the forms in the system and the code used

Code Notes – 1st form load

- Check the domain to which I am connected
- Check the SQL Server to which I am connected, and to which I should be connected
- Reconnect if necessary
- Show the diagnostics form or not (controlled by domain and user); this gives them the opportunity (or not) to change databases (not that for my domain STATICDEV, the form is shown automatically)

Code… – 2nd form open/load

- Load
  - Some stuff
- Open
  - Get environment details. (Programmatically – not just DOS-like environment variables!).
  - Check the database table connections until I find one that is connected to a backend database
  - Get that database details and put on the form
- If you show the debug form all the time, then you can consolidate these 2 forms into one. But as my systems are optional, I need to keep the form activities separate (or at least did when I wrote it!)

Code… – 2nd form chg db/SQL

- Mistake handling (blanks, and selecting the current db/server)
- Close the main menu of the app (just in case)
- Get confirmation from user
- Connect to the different database/server
- Update form display
- Check version

Code… – 2nd form chk/refresh

- Check
  - Calls the routine
  - Updates the user
- Refresh
  - Seeks confirmation
  - Does it

Summary & thoughts

- Deployment – is great.
  - You compress the database, and compile it
  - You compress it again, just in case ☺
  - You create the ACCDE for the client and send it to them
  - They run it, and it just reconnects to the right database
  - (optionally)
    - They actually test the upgrade
    - They preserve the reconnected ACCDE for distribution to all
- If they change their SQL Server without saying it breaks
- If they change their network name without telling you it breaks
- If a member of staff steals the SQL database and the front end, it won't work – but see comment much earlier.

Things I *will* change

- Add into the change database code that if the version numbers don't match it reverts back to the previous database/SQL Server.
  - This means recording the previous working database name and SQL Server option; not difficult – just not needed yet.
  - Run the code again
- After it was needed elsewhere, I have left the framework for more than 1 SQL Server/Database combination in use, duplicating connection variables. You will need to fill in the framework for this according to need.
- In that case it was a large NHS Trust with a common admin database (users, locations…), and then application specific databases.

Things that could go wrong

- The “[“ bug in the re-indexing (now fixed)
  - To allow for spaces in the names of an index – rare occurrence, but I hit it somewhere (recall – I often work on pre-existing servers/apps)
- If a new table is introduced to the system and a user switches the client to an older db without it, the relink will crash. This happens.
  - However, if I insert a “is the new database the same version as the front end” fix ***before*** I switch (see last slide)…
  - *AND*
  - Make sure I keep numbering up to date
  - All shall be well

Things you could change

- This assumes DSN-Less SQL connections with passthrough security.
- With extra data in the local db you could store and use named userid and password authentication (but thereby introduce different security issues).
- With a bit more effort, you could do something similar for other db connections (Access, Oracle, etc etc). You just capture the entire connection string, tweak and relink.
- With even more effort; you could probably link tables from different sources, and just store the link details, and have a function for each type of relink.
- Tempting, but I refer you back to the “I can’t be *****” comment (and also, “no one has needed it yet”).

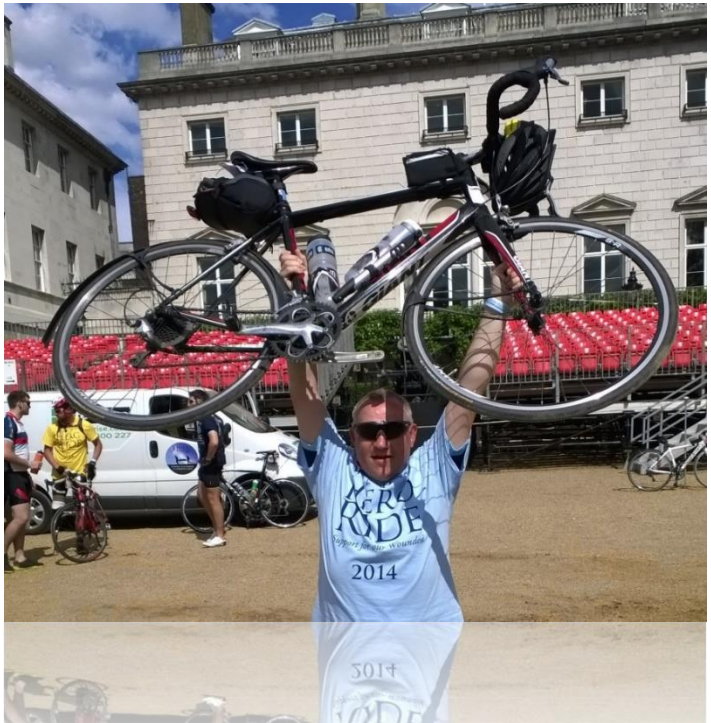
Post event documents

- Slides available.
- ACCDB available
- The 4 SQL databases (possibly with reduced content) as .BAK backups
- If feedback suggests it would be useful, T-SQL Scripts to create the databases instead of backups.

- This will be released “as is”; no support (although you can ping me if you find a bug ☺)
- Please retain all credit comments in the code
- Credit due to other parties is acknowledged in the code. In particular – Andy Couch.

Help for Heroes

- After a 4 year hiatus (thanks to c***d and two spinal operations) – I'm cycling again with Help For Heroes June 2024. Just 350 miles through the battlefields of France.
- If you would like to donate – https://www.justgiving.com/fundraising/pjbryant2024



Automating DSN-Less connection to SQL Server

Peter Bryant

Corylus Business Systems

pbryant@corylus-business.co.uk