

**METHODOLOGY FOR CONVERTING  
LEGACY ISAM BASED APPLICATIONS AND DATABASES  
TO MODERN ENVIRONMENTS AND CLIENT-SERVER  
DATABASES**

Developed by:



6215 Split Creek Lane  
Alexandria, VA 22312

June 2007

## TABLE OF CONTENTS

1	Overview.....	2
2	Scope.....	4
3	Process Flow.....	4
3.1	Step 1: Establish the Design of the Replacement Application.....	4
3.2	Step 2: Address Table and File Integrity and Associate Related Files Together.....	6
3.3	Step 3: Document Table Structures and Index Expressions.....	6
3.4	Step 4: Map Table Structures to Client-server Equivalents.....	7
3.5	Step 5: Create Indices for Each Table and Define Relationships.....	8
3.6	Step 6: Migrate Raw Data to Client-server Environment.....	8
3.7	Step 7: Test and Validate the Data and New Application.....	8
3.8	Step 8: Documentation.....	9

## 1 OVERVIEW

This document describes Najia System's LLC (NSL's) Methodology for converting legacy ISAM based applications and databases to modern environments and client-server databases. This section provides an overview of the current challenges. ISAM databases possess several properties that distinguish them from modern environments that made them suitable for small scale applications prior to the ubiquitous spread of the Internet and extensive networks. These properties were beneficial for the task at hand then, but now do not lend themselves to modern distributed application environments, especially where the user community extends to the hundreds, or even thousands. This is particularly true of graphical environments that are more taxing on hardware than their character based predecessors. Some of these properties are:

- *File server dependency*: the data is stored in files or groups of files (dBase III/IV, Clipper, and FoxPro as well as all their derivatives distinguish a table as a series of files that must operate together in concert.
- *Limited scalability*: this takes on several elements:
  - Beyond a certain user load, the performance of the database begins to degrade steeply. Depending on the technology, multiple hundreds or thousands of concurrent users will result in serious slowdowns in delivery of data to a user's workstation.
  - The same applies to database and table size – beyond a given threshold, depending on the technology used, the very size of the database makes maintenance difficult, time consuming and prone to data corruption.
  - Server capacity becomes an issue also because every table opened represents a 'file handle'. The number of concurrently open files was dependent on the operating system. The advent of graphical user interfaces (GUIs) in operating systems (e.g., Windows), ensured additional limitations since GUIs have more concurrently open files than their character based equivalents (such as the ever present Disk Operating System, or DOS).
  - Records passed to and from the client workstation were limited by network bandwidth in that the ISAM model sends *entire records* starting at the top of a given table and going to its end. The server does not return only the desired records, instead the workstation has to traverse and entire table to retrieve a specific record or series of records.
- *Limited security*: data security was heavily dependent on the application layer. That is, security controls had to either be encoded in the application, defined by permissions on the server or both. Once access to the database tables in their file form was gained, the data was fully viewable. Some technologies had limited encryption capabilities, but were confined to certain field types and sizes, since any column or row level encryption would lead serious performance degradation

- the burden of performance was on the client workstation, and at the time, there had a fraction of computing ability we see today.
- *Limited row size, column size and native field validation*: due to memory constraints row size was limited, data types were less flexible and a lot of validation was offloaded to the application. This had benefits because the server did little work in terms of controlling the data being passed around the network.
- *Garbage collection*: memory leaks and defragmentation are still problems today, but today's operating systems are much better at resolving them.
- *Pessimistic locking*: today's SQL based client-server databases rely on 'optimistic' locking, where a record is not locked while a user is editing it. A user who edits a record and then walks away from his/her workstation would prevent other users' from accessing the record (at times this can be disruptive, especially during maintenance) until the lock is released. This is especially disruptive where automated access of records is required (such as during batch process execution).

Migrating to client-server databases is more than a simple move of data from one to another. The simple transfer of data can be accomplished via a number of embedded ETL (Extract, Transform, Load) tools (both Oracle and SQL Server have built in functions for this). However, not all data will carry over cleanly. Indices must be transcribed, table relationships established, validations defined and fields with characters that do not migrate must be addressed.

The code from the application cannot be re-factored for the following reasons:

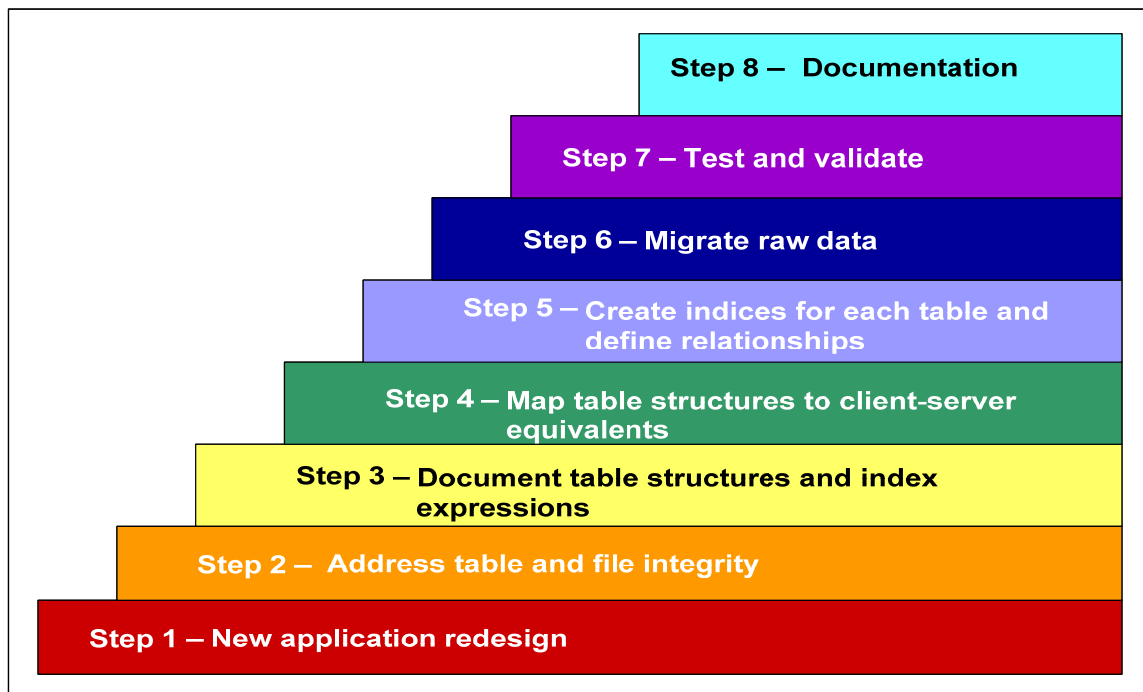
- Beyond Visual FoxPro, xBase clones are not supported today. Microsoft continues to support Visual FoxPro, but it has transformed significantly since its inclusion in Visual Studio and now has more analogues with Visual Basic.
- Third party library add-ons will not migrate unless the vendor continues to support it. In any case, much of the functionality has been subsumed into the successors' environments. The third party library/add-on might be supported in the replacement environment, but that isn't always the case.
- The user interface is typically not recyclable as it is based on the DOS user interface concept. Today's environments provide for non-Modal windows and multi-threaded applications that the early xBase world did not support.
- The validation logic of some of the older xBase dialects did not have on-the-fly field validation – rather the entire form had to be 'submitted' before any validation could take place. This made any conditional content of fields difficult or impossible to implement – something that is easily done today.
- All xBase dialects are not object oriented (Visual FoxPro is *object based* only). Today's programming environments are object based at least, and largely object oriented.

## 2 SCOPE

This document focuses its attention on databases and tables from the xBase era. These include dBase III/IV, FoxPro, FoxPro for Windows, Visual dBase, Clipper and all their derivatives. The steps we take in our methodology are outlined in the sections below.

## 3 PROCESS FLOW

The migration out of the ISAM environment to the modern technologies involves several discrete steps outlined in the figure below:



The first step, Application Redesign is the most crucial. A clean design, even if independent of the overall data migration and the legacy application, will dictate the effects of the subsequent steps. The entire sequence is discussed in the sections that follow.

### 3.1 STEP 1: ESTABLISH THE DESIGN OF THE REPLACEMENT APPLICATION

It is not possible to migrate any of the data to another structure that does not perpetuate the limitations of the ISAM structure unless the new system that is planned for the data has been thoroughly analyzed, requirements established, design completed; and vetted with stakeholders. Once the functionality of the new system is clear, a database design may be developed that includes as much of the legacy database's data content as possible. At this stage, the target database structure will have been optimized for performance (speed or capacity). The target database will also have had any embedded validation defined – for example, if a field is meant to have something from a fixed list of values, this may be embedded in the database and offload validation code to the database.

For several years, xBase clone vendors have sought to keep the xBase dialect alive with a variety of products. The most notable of these are Microsoft (FoxPro/Windows) and dBase, Inc. (dBase Plus). While Microsoft continues to support FoxPro, and dBase Plus is a GUI, this document deals with **conversion** and in the DOS to Windows world, this frequently means a re-write of the code, irrespective of the target environment. A quick ‘interim’ move using dBase Plus might be viable to go from DOS to Windows (dBase Plus actually has a migration tool), but there are no guarantees that this will work seamlessly.

Converting the application code must begin with a review of any available documentation (user, system and technical). If requirements documentation is available, it should be confirmed against the currently deployed application. Once the requirements have been updated to include current as well as *desired* functionality (which also may include interfaces with upstream and downstream systems), a new technical specification may be developed. This specification must take into account the migration of the legacy data as well as business rules, artifacts, interfaces and business processes into the replacement system. That is, it is simple enough to define a migration from dBase IV to J2EE if no new functionality is desired, but more than likely, new functionality will be necessary just because the two technologies are so dissimilar. For example, dBase IV is DOS based, but Visual dBase was based on the early Windows 9x platform. Today, neither of these platforms is supported and the Windows user interface is moving towards the Microsoft Windows Vista model as Windows XP (the 32 bit version) approaches the end of its service life. Components of the operating system (assuming the thick client model) are also not compatible as the XP and Vista portions have different “under-the-hood” functionality. Consequently, a full revisit of the user interface design is warranted.

Extracting system documentation may be possible directly from the code if the application code was well documented internally. Since this is rarely the case, a documenting tool such as DiamondEdge may be. This product reads through the code and builds a “call-tree” to identify functions that call each other. A more straightforward approach may also be to use Visio to read the legacy database schema and define the database relationship ONLY, but this only applies more to smaller databases. Once the code is documented, the basic logic flow and code rules may be also documented, and be used as inputs to the overall replacement application design. The xBase code itself cannot be directly re-used in a reliable and consistent fashion when departing from the xBase model (e.g. converting to VB.NET or J2EE) – the effort to conduct a direct line by line conversion is equally costly and time consuming as a full redesign and redevelopment but would lack the more modern user interface artifacts. This was evident when FoxPro/DOS applications were first migrated to FoxPro/Windows and when Clipper and dBase III/IV code was first ported to Visual dBase and the like.

Therefore, the migration path of the application code has to be approached post review of available documentation, current and future requirements, and platform standards inherent to the organization where the application is resident. xBase’s open-source

dialects have made their way to the Linux world, so if a command line/character based interface is desired, the code may be updatable under that platform.

Two additional factors to consider in xBase applications are reporting and multi-user issues. The first factor is that the original xBase systems were very limited when it came to multiple concurrent users and relied heavily on workstations, and ultimately the server's ability to provide access had a definitive threshold beyond which performance would seriously degrade. Consequently, any migration must address multi-user factors, which could bring about major architectural changes (again, requiring a full rewrite). Integration with other systems external to the application will also need investigation since many products nowadays attempt to lock entire tables whenever they are accessed, even in read only mode, which may effectively disrupt the operation of the legacy application.

The second factor, reporting, may be a mishmash of report tools (e.g. early versions of Seagate's Crystal Reports) and native reporting functions. At the time, older xBase applications relied on an 80 character by 25 line page (in portrait on an 8.5" x 11" sheet of paper), because all reports were based on a dot matrix printer or a mono-spaced font in early laser printers. Today we have a variety of fonts (Arial, Times Roman, etc.) and text artifacts (bold, italic, font pitch) that provide richer report content, not to mention the addition of charts and figures, as well as integration with office suite (e.g., Microsoft Office) products. Perpetuating what amounts to serial (line by line) printing would only provide an interim solution that would have to be addressed again. Converting reports from one format (e.g., older Crystal Reports, to Cognos) may not be straightforward and require manual intervention, again speaking to a complete rewrite.

### **3.2 STEP 2: ADDRESS TABLE AND FILE INTEGRITY AND ASSOCIATE RELATED FILES TOGETHER**

ISAM databases are comprised of several table files and attendant index and 'memo' field files. Index files may be 'compound index' files (e.g. dBase IV's MDX, and FoxPro's CDX files). The initial step is to ensure that all files sets are in sync. This means that every DBF/DBT (dBase IV/Clipper), or DBF/FPT (FoxPro) file pair should be free of file pointer problems that result from file open/close issues.

The next step would be to identify all related file sets. This means associating the DBF/DBT or DBF/FPT file pair with their associated index files. For a given file pair, both compound and individual index files may be defined (e.g. both NDX and MDX files). While the compound index file *usually* carries the same file name as the source DBF file name, the NTX file names need not, and frequently do not, match. Definition of the relationships between the files permits the analysis of the prevailing indices which may then be defined in the client-server environment.

### **3.3 STEP 3: DOCUMENT TABLE STRUCTURES AND INDEX EXPRESSIONS**

There are several ways to generate a table structure and the index expressions for a series of tables. The easiest is to write a small script that scans all tables and generates a text

file containing each table's structure, along with the expressions of each index. The relationships between tables will have to be inferred either from the legacy application code or by inspection. This is not as complex as it seems. Another simple mechanism is to import the database structure into another product (this is a menu driven operation) like Visio or Microsoft Access and have that product attempt to establish relationships as a baseline.

### **3.4 STEP 4: MAP TABLE STRUCTURES TO CLIENT-SERVER EQUIVALENTS**

Investigating the content of the database schema now allows for the mapping of old structure to new. At this stage, it is attractive to attempt to optimize the database structure for speed or volume. For the simplest databases, with flatter structures, optimization for speed and volume is straight forward and may entail little to no change in the basic schema. However, where larger, more complex databases are desired and expected, the database schema must be optimized for either speed and volume using a variety of techniques, particularly where the user community is expected to be orders of magnitude larger than that of the legacy system.

If a new structure is to be defined, it must map old to new. For example, for performance reasons (usually speed related), an ISAM database may have tables not normalized to 3<sup>rd</sup> normal form. At this stage, the mapping could implement conversion to full normalization. In any event, this table structure is the final structure of the database and must be preceded by a thorough analysis of the database content, field structure, size, expected capacities (users, network load) and security requirements. If the database is to 'feed' a new generation of the application so that legacy information can be carried forward, there must be enough flexibility to handle any inconsistencies in the database (data entry errors or orphaned records, for example). The target database structure will likely have been established as part of the replacement system's design, and the mapping of old to new would define a clear path from legacy to replacement.

At this stage, decisions need to be made about how to clean the content of legacy database. The expectation is for the final database to be in a pristine form so that there are no integrity issues that might crash the subsequent application. The new application would, of course, have tighter controls to prevent orphaned records and to better protect against data entry errors. It is therefore appropriate to purge bad records out of the database since it is likely the migration will take place more than once: for testing purposes (especially during a "dry run"), to generate one or more test databases or scenarios, and finally when going into production.

Several fields take on different, but same functionality going from ISAM to client-server. For example, Boolean fields containing .T. or .F. (for True and False) map to *binary* or *tinyint* fields containing a 1 or a 0. These carry over automatically as part of the translation. However, the xBase dialects make use of only (typically) 5 types of fields:

- Numeric (integer or otherwise)
- Date

- Memo
- Text
- Boolean

In the client-server environment, Memo fields could be *varchar* types with as much as 8000 (for SQL Server, 4000 for Oracle) characters, but also as *text* which allows for much more space (in some cases as much as 64k worth of text). This then means that the actual migration has to take place using an ETL or SQL script post the creation of the target database structure.

Finally, if any data driven security system is built in to the application, this would be the time to have defined a new security model, and map this new security model to the old database. It is entirely possible that an automated migration of the security data (user IDs, passwords, roles) may not be possible and may have to be hand entered back into the new system – for example, authentication may be offloaded to Active Directory, or database level security (i.e., authentication by the database server).

### **3.5 STEP 5: CREATE INDICES FOR EACH TABLE AND DEFINE RELATIONSHIPS**

The process of defining the database above, along with its mapping should lead to a script to create an instance of the database with its associated indices and table relationships. This also establishes the rules for referential integrity and ensures that any automated script that actually does the migration maintains the integrity of the data (the script will generate errors if not).

### **3.6 STEP 6: MIGRATE RAW DATA TO CLIENT-SERVER ENVIRONMENT**

With the before and after database structures established, documented and well understood, the initial database migration may begin. At this point, all users and external (upstream and downstream) systems must be prevented from access to the legacy application tables until the migration has taken place. It is typical that a “dry run” migration will take place prior to final migration into production. This means that an established procedure to move to production must be in place, which includes ensuring data integrity prior to the movement of the data into the client-server medium.

The actual migration may take place by a combination of one or more scripts, some native (such as the definition of an ETL package in Microsoft’s SQL Server) and some developed (e.g. a custom SQL script to dynamically move and format data as it is moved). At this stage, indices and relationships will have already been defined as part of the target database creation. Referential integrity will therefore be confirmed (as defined by the database and table definitions and their associated rules) automatically. However, post migration, the database should be inspected to confirm that all records did migrate over, and all data did migrate in the form and format it should.

### **3.7 STEP 7: TEST AND VALIDATE THE DATA AND NEW APPLICATION**

The converted legacy database can have dual purposes:

- As a source of test data
- As the ‘seed’ and initial set of production data (which is the original intent)

Aside from the new application itself, a series of “integrity” routines may be developed to confirm all the required data that should have migrated did indeed migrate. The nature of the client-server database model will ensure no orphan records exist so long as table relationships have been defined. The act of moving data via the ETL process will prevent any orphaned records from existing. However, this does not test the relationships themselves and a short routine to confirm that child records have been carried forward with associated parent records may be quickly assembled. Other integrity routines to develop include:

- Record counters, confirming that the expected number of records exist.
- Field content checks (e.g., no truncated text fields, no invalid characters in text fields, correct precision of numbers – rounding of large numbers would create an issue).
- Correct operation and generation of ID fields (auto generated or otherwise).
- No duplicate records where duplicate records are not allowed – at times duplicate records may exist because of an auto-generated key for otherwise identical records. The ETL process would not defend against this condition. In other words, if a record can be defined as unique via multiple routes (e.g., both ID and a composite of other fields), than a duplicate check is required.

Invalid records should be removed from the source database and if possible the migration script re-executed. If this is not possible, then the invalid records will have to be manually addressed in both legacy and new database. This is to ensure parity between the old and new systems should parallel testing be required.

The application itself may then undergo testing to confirm it meets all requirements in terms of functionality and performance. Parallel testing, which checks against the legacy application will help confirm any calculations. Reports should also match in content if not appearance.

### **3.8 STEP 8: DOCUMENTATION**

Prior to full deployment, user, administrative, and system documentation should be completed. From the beginning of the new system’s development, documentation will have been available against which requirements may be validated. This in turn can drive the user and administrative manuals. Internally, all code should be documented to clearly indicate the purpose of every function, routine, module and component. If possible the development environment itself should be documented should special settings be required (e.g., development environment parameters, server settings, potential conflicts with other software and the like). The testing process should also put the documentation through a degree of rigor that confirms its currency with the deployed application, ease of use and

completeness. For example, during testing under multiple environments, several factors may appear with slightly different screens (e.g., the File Dialog under Windows 2000 changes under XP and changes again under Vista, the Print Dialog is one 'box' in Windows 2000 but is split up into multiple dialogs in XP and Vista).