MediaPortal Refactor – How it Works

MediaPortal Refactor – How it Works

# Introduction

This document describes the issues involved in making MediaPortal work with a single multi-user database; i.e. for all clients. There are, in fact several databases, the main ones being:

* Fanart database (optional);
* Folder database;
* Music database;
* Picture database.
* Video database;

Each set of databases must be present for each client individually, because MP locks the client databases for the duration of a session.

## Background

MediaPortal (MP) is a well-used medium for accessing your videos, music and pictures. It was developed some time ago and has grown in complexity over the years. There are two versions of MediaPortal: MP1 (the original version that uses SQL Server as its main programs database) and MP2 (a more recent version that essentially uses a small-footprint database, SQLite, instead of SQL Server and includes a skin very much like Windows Media Centre).

The author first used Windows Media Center (WMC) to access digital media and changed to MP when Microsoft stopped supporting WMC. WMC required the use of Xbox devices while MP requires a PC for each client.

MP provided many improvements over WMC, not the least of which was that MP supported as many tuners as you could install on a machine while WMC restricted the number tuners to 2 tuners, and later 4 tuners. Before Windows 8 (I think, but it may have been Windows 7) came out, there was a hack that removed the 2 tuner restriction. However, for some unfathomable reason, Microsoft seemed intent on imposing a restriction and they also tightened up the security on the WMC database preventing the hack from working. Although I was forced into moving away from WMC, it turned out that MP was, in my opinion, a better product anyway.

After using MP1 for several years, it has become evident that we are not quite at a place where MP1 is fully multi-user capable when it probably could be. This document will discuss how that goal was achieved.

NOTE: MP has an option whether to only process media when it is selected or to process all media in a selected folder in one go. This document will assume that the latter option has been selected since this is the most testing condition.

## Document Content

This document will cover the following topics:

* MediaPortal - a Brief Description;
* How to Implement Multi-User Capability;
* The Solution;
* Data Management.

## Who should read this document

This document is intended for advanced MP users and developers that are interested in making MP multi-user capable.

## Defining terms

This document uses the following terms:

* SQL
* SQL Server;
* SQLite.
* Movie/Video

### SQL

SQL (Structured Query Language) was introduced to democratize database accessing. Before the introduction of SQL, databases were proprietary to the manufacturer’s machines and were only available to specialized coders. Semi-relational databases were introduced by some producers, like IBM with the System36 and System38 on which developers could define indexes over tables in the certainty that their code would use those indexes. In fact, they could declare which index they wanted to use to access a table.

Fully relational databases do not allow developers to specify which index to use when accessing a table. The database engines of each manufacturer’s database determine the rules for whether an index will be used or not. In other words, just because an index is created, it does not mean that it will be used. For example, if an index is created over a bool column, which obviously has a very low level of uniqueness (2%), it is extremely unlikely to be used by the database engine; because the system would have to read the entire index anyway. The more unique the data in the key column, the more efficient the index will be. For example, a timestamp has a level of uniqueness of 100%. The general rule for the use of an index is to have a level of uniqueness of around 33% or greater – the actual figure depending upon the database manufacturer.

The self-management aspect of fully relational databases indicates why the design of the database, and the relationships between the tables, is critical to their performance, because there is little a developer can do to improve performance once the design is fixed.

### SQL Server

SQL Server is an enterprise level database that was originally developed by Sybase and then taken over and completely re-developed by Microsoft. It is a fully multi-user database and comes with the costs and management overheads commensurate with that feature. It achieves its multi-user functionality by locking pages within the database around the data being changed (as opposed to the entire database).

### SQLite

SQLite is a small footprint database that is a file sitting on your hard drive. It has far fewer management needs and is surprisingly resilient despite having one drawback; it is essentially a single user database. Although that may sound like a major problem, it actually only locks the database when data is written to it (however it is the whole database). You can have multiple concurrent reads from a SQLite database but only one write at a time and reads are prevented when the database is locked.

This does not mean that you cannot have a multi-user application using SQLite, but it does mean that you will have to take great care to ensure that all of your database writes happen synchronously.

### Movie/Video

The terms “movie” and “video” will be used interchangeably in this document meaning any kind of video file.

# MediaPortal – a Brief Description

MediaPortal is comprised of two components:

* A server side repository for TV programmes;
* One or more clients.

The data stored by MP falls into the following categories:

* Data processed by the TV Service (on the MP server component);
* Data processed by each MP client.

## The Server Component

The server side component of MP1 uses tuner cards to access broadcast media and to store information about that media in a database. For MP1, that media is some form of SQL database; e.g. SQL Server. In MP2, all databases are SQLite databases.

## The Client Component

The client component accesses the data stored by the server component and allows users to view and record broadcast media. The client databases collect information about the media stored by the users and keeps a record of where each user left off viewing/listening to said media. There is a third-party plugin that synchronizes client databases but it is not a real-time solution. As we will see below, synchronizing relational databases is not as simple a task as might be assumed because relational databases, quite rightly, usually employ numeric primary keys to join tables together and the next index number generator is completely dependent on the order in which videos are processed. This means that two client databases may have different IDs for the same videos because they operate independently.

## How MP Works

The key feature of MP is that it can either access private media (stored locally on a PC) or shared media (stored elsewhere and given general access rights). In either case, information about media is stored in local SQLite databases for the following:

* Fanart data;
* Folder information;
* Music data;
* Picture data;
* Video data.



MP1 (out of the box) does not provide data synchronization; as is stated by the website (above).

MP has moved to smaller footprint databases over the years. This made sense because it reduced the cost of using MP for everyone, as well as the run-time overheads, but there were some consequences of this move: namely, synchronization. To date, both versions of MP have adopted a closed system approach to data client data storage. Each client kept a separate copy of each of its repositories for videos, music and pictures. This meant, in reality, that the client data was tied to a physical PC. Attempts were made to provide tools to synchronize client data, but this was a very difficult task to get right. Furthermore, such tools required the household members to understand how not to tread on each other’s toes (as it were).

The weakness in this approach is that households don’t usually work that way. Different members of the household will use different machines, as they move around the house. Unsurprisingly, each member will think that, when they have finished watching a program half-way through on one client, they will resume watching that program from where they left off when they view it from another client. MP1 does not cater for this requirement.

There are reasons why MP1 adopted this strategy and they are valid (particularly for the time of development). Time has moved on and the cost of hardware has fallen dramatically since then; making it much more viable for users to have machines running 24/7. Also, when media is shared, as it is when it resides on a server or NAS storage, then it will be likely that the place of storage will be running 24/7.

MP1, but not MP2, allowed users to configure a centrally controlled system. This resolved the previously very irritating feature that users had to repeat setup procedures for each client machine individually. This is okay when you are only doing this once. But when you are trying out different settings, it is maddening to have to go around every machine every time you make a configuration change to one machine. This centralization feature included the database. So, it is possible to easily setup MP1 to use one database for all clients.

Although it is possible to have a single database for all clients, that configuration will not work because MP locks the client databases for the entirety of each client session: effectively preventing more than one client from using MP at the same time.

## Data Points in MP

The main point of access in MP is when folder contents are displayed. This is the point at which the system works out whether a movie, or picture, or music track exists in the system’s database. This is also the point at which the system can detect whether sources held in the database no longer exist. This is the largest bottleneck point in the system and is where threading is essential to allow the user interface to progress smoothly while data processing continues in the background.

Another point of access is when users scan for information about their media; involving internet and database processing.

The rest of the time, users predominantly access the media they have; involving far less database processing.

## How Processing Determines which Database Technology to Use

MP1 uses SQL Server, or equivalent technology, to handle TV programming; i.e. the capture of programs. In MP2, the backend technology changes to SQLite. This is possible because of the nature of the tasks performed by TVService; the windows service that captures and processes TV programs, is, in practice, limited to single user access. All of the tasks are performed by a single process on a single machine – in effect the MP server.

SQL Server, as well as other enterprise level databases like Oracle, have engines that handle multi-user access as part of its data management; which is why they have a significant overhead built into their use. These enterprise level databases take care of the processing of data processing from multiple sources without the client software having to do anything about the multi-user aspect of data processing. While this is generally correct, the database developer can do things to ensure the integrity of the data by employing techniques like optimistic locking to make data processing very quick while handling more than one client trying to access the same data at the same time. Optimistic locking involves using a timestamp field in each row of each table, which is altered every time a record is changed, so that the application can know when data collected by that application has been changed by someone else before any actions have been performed on that data by the current application. This sort of processing is necessary when there is a very high hit-rate on the database. An example of such an application is when there is a significant amount of data entry; e.g. where people are entering their meter readings for gas and electricity. The provider will have no alternative to using an enterprise level database.

MP is the opposite kind of application because it has a very low database hit-rate. Most of the time users are watching videos, listening to music or looking at pictures. The main points of data processing are when folders and folder contents are displayed or when the user stops watching a video and/or deletes it. This behaviour makes MP a suitable candidate for a small footprint database like SQLite and makes the processing capable of multi-user access of a single, central database.

### How MP Clients Work

There is an aspect to MP client behaviour that is not standard/expected behaviour in most other applications; the user may want to have a closed system during a session. What I mean by this is that the user may start to watch a movie and may stop watching for some reason and continue watching the video in the same session later on. Suppose that another person in the household decided to watch the same video, the first user would not want the second user’s activity to interfere with their progress through the video.

At the same time, we would want each client to be aware of real-world changes, like a video being deleted or new videos being added to the hard drive’s shared folders.

We can achieve both aims by using cached data for the duration of a session while responding to new videos being added to the database and existing videos being removed. The database would be updated in real time for items like the resume stop time but the session would process the stop time held in memory. We have a choice; we can populate the cache at the start of the program and only change the cache with the client’s changes during that session, or we can repopulate the cache periodically to ensure that the session is kept aligned with changes made by other clients. The author favours the first option but it would be easy to implement the second option if required (albeit with some performance overhead).

### Data Integrity in a Multi-User Environment

It is proposed that, no matter whether the methods making calls to the database think they are performing inserts, deletes or updates, the back end, the Data Queue, will always deal with the data it has. In other words, the Data Queue will always check, against the database itself (as opposed to a cache), whether a record already exists and acts according to whether the instruction is to delete or add/update.

This raises a very important question about what we do when different clients from the same household process the same videos, but in a different order. We are going to be sacrificing real-time database activity in order to facilitate multi-threaded processing and using a small footprint database. Therefore, it is highly likely that conflicts will occur. We will cater for this by adopting a strategy of using the description field, for an actor (as an example), to ensure that we link to the correct actor when performing database updates. Another example is using a path name when processing a movie. When we create a movie record, we get the Id of the path associated with the movie and join the movie record to a path record. To ensure data integrity, we will pass the description of the path, alongside the movie object to the data queue, so that the Data Queue will always use the correct path Id to join to the movie record. In effect, the Data Queue will always use the actual path ID, held in the database for a given path, as opposed to the path ID a client was using.

## Issues in detail

We are now going to look, in detail, at the reasons why MP1 locks the database and possible ways around it. The author has already done some development on resolving these issues and there has come a point at which major decisions need to be made because these steps are irreversible and will interfere with the upgrade path.

### Keeping Databases Concurrent

MP1 ensures that you are using the right version of the database for the software version and it performs this task every time a new connection is made to the database. I guess this is okay when you are only going to make one connection, but we are wanting to allow MP to be multi-user. Therefore, the first thing we must do is stop having one connection that lasts the whole session and that also means that we will have to stop rechecking the database on every new connection.

It could be argued that we should use MP Configuration to setup the database as well as the install procedure to maintain the correct version of the database. This would seem to be a more performant way to handle database versioning. However, for now, both applications will check the database version when they start up.

### The Microsoft Mantra

Microsoft have had a mantra, to my knowledge since the Microsoft conference in Nice in 1998:

“Use resource as late as possible and release them as soon as possible.”

If there is one thing that MP1 doesn’t do, it is this. The author thinks that developers often believe that it is expensive to make repeated connects and disconnects from databases – but this is not true. Only the first connect is expensive. From then on all connections are made in memory. The author was lucky enough to meet the head of SQL Server development for Microsoft at the Nice conference and this, together with a few other performance related issues (like the number of columns in a table effecting performance exponentially), were raised in conversation then.

It is strongly recommended that all SQL statements be encompassed by a using statement. This approach guarantees that a connection is disconnected as soon as the using scope is left.

It can be argued that we shouldn’t employ this technique when performing iterative tasks. If we were performing all of our tasks using the database that might be true, but in this case that scenario will not happen because all databases actions - writes, updates and deletes - will be queued. Under other scenarios, the using statement can be wrapped around the loop to improve performance by ensuring that there is only one connect/disconnect for the duration of the loop.

### Threading

The author mentioned earlier that it is safe to use SQLite in a multi-user environment so long as the writes are synchronous. MP1 employs multi-threading, in order to allow the GUI interface to proceed uninterrupted, but it performs writes to the database in more than one concurrent thread. Consequently, the database class has to be locked to prevent write failures owing to the database engine being in use.

#### Why this matters

When we look at the current system, and we accept the use of a tool to synchronize client databases, then we need to understand how problems can arise. Consider the following:

* We have two client machine;
* Client A starts an loads its database with the contents of a folder – writing data to the movie and path tables;
* Client B knows nothing about these contents unless it happens to access the same folder;
* If client B accesses some other folders before accessing the same folder as client A, then the IDs of the client B equivalent records will differ from those held by client A;
* Trying to copy the client B database to client A’s machine will cause client A losing any resume data it had created in the previous session. A copy, say on a watcher being triggered, will not work. The synchronization process has to go through every record in every table, comparing the two databases, to ensure that both databases get the changed data from the other client database (and vice-versa) without relying on the IDs being the same. Not a simple matter and by no means quick.

Although the author has managed to get this process down to around two to three seconds, it would be better not to need to do any synchronization at all.

Therefore, the best solution would be to remove the locks – or better put – to reduce the lock time to an absolute minimum and ensure that any writes are only performed after that process has checked that the database is not locked beforehand.

### The Technology Used to Access Data

MP1 uses the time-honoured SQL language to gather and write data. There is however a fundamental weakness with SQL; it has to access the database directly. In an environment where database access can be locked out we need to have other options.

Time has introduced a whole new set of tools to access and write data, including Entity Framework (EF), Linq and Lambda, which work for all database mediums (like xml and SQL) in the same way. They can also be used to access databases directly, or they can perform exactly the same procedures on cached data. These features provide a better way to ensure that database changes are committed to databases, and locks freed, that were previously unavailable or complex to adopt.

The use of EF and LINQ are object based. This approach ensures a better level of data integrity and typing than is available when using SQL. For the most part, EF and LINQ are self-validating – or, at least, validate to a much higher level than SQL.

There are some cases in MP1 where we need to keep some current code because it uses dynamic queries (generating SQL on the fly) and the new methodologies, being object driven, and don’t lend themselves to dynamic SQL. Developers should be acutely aware that whenever they use SQL in this refactored solution, they will be accessing the database directly – which the rest of the solution goes to great lengths to avoid.

Therefore it is proposed that the new solution employs both techniques, the new techniques when possible, but only Entity Framework/Linq/Lambda will write to the database using SaveChanges() to commit changes to the database.

NOTE: it is advisable that all SQL code be removed as soon as is practicable because they will be accessing the database directly and, inevitably, will generate run-time errors – albeit infrequently, and randomly – when timings cause database clashes.

# How to Implement Multi-user Capability

The best way to ensure data integrity will require more than one change to MP1. We should consider the following:

* Allowing database writes from only one thread or implementing a queue;
* Removing the data layer from the front-end.

This is not a trivial matter and will require considerable work to pull off. However, the aim is to provide a system that can access a single database and it is felt that the ends justify the effort.

## A Thread for Database Writes

MP1 employs multi-threading to handle many tasks that improve the user experience: for example, applying thumbnails to videos and blacklisting videos. There are some database accesses that go alongside that processing. If we could change the database accessing to set data in memory, then we can perform the actual database writes in the single thread proposed.

## Database Layer

One way to allow the front end and the database layer to communicate might be through typed lists where the lists reflect changes and then be used to trigger database accesses. This is the approach that the author adopted for his synchronization application and it provided a workable solution.

## Strategies to Make a Central Database Work

SQL databases work by employing joins between tables, the main table having a Primary Key and the table joining to that table having a Foreign Key that relates to the primary table. The most efficient way to declare these relationships (joins) is by using unique identifying columns that store each relationship as a number. The usual way to generate this number is by making the Primary Key auto-generate its value. We cannot obtain the generated number until the record has been committed to the database.

The reason why numeric indexes are much better than any other way to join tables is because the indexes are much smaller than those for character indexes. Furthermore, concatenated indexes are much larger than single column indexes. At least one primary key index must exist for each table. In fact, the system will automatically implicitly create one. Taking these three points together, the performance difference between using numeric and character indexes is exponential. Finally, the algorithms that process indexes work far faster for numeric keys than they do for character keys.

The path entity is the central entity in the data model for MediaPortal from which everything else is joined. Therefore, we must be able to get the primary key for each path before we can progress to using its child entities. The disadvantage of using queues to process data is that we cannot predict exactly when each operation will happen, and in the case of paths, we need to know exactly when a path record will be written so we can obtain each path’s primary key.

# The Solution

It is proposed to use the following techniques to ensure that the data processing is managed as efficiently as possible:

* Use object based data processing with Entity Framework, Linq and Lambda;
* Store client information in lists for access by the front end application without having to keep hitting the database;
* Employ ConcurrentQueue to queue database writes in its own thread which will be fired in the BuildDatabase method of the VideoDatabase class (and all other MP databases).

## List storage (caching)

The lists will be populated when the application starts up – in the Init method of GUIVideoFiles; GUIMusicFiles and GUIPictures for the music and picture data. From then on, the lists will be maintained by the queueing process to ensure that the database and lists are kept in-step with each other.

## The Queue

The queue will receive calls to EnQueue data using a class that holds the following data:

* ForeignKeyName – description names of any foreign keys(separated by ‘|’);
* Database – the database to which the action refers;
* Entity – the entity affected by the action;
* Transaction – indicates whether the action is a Add, Delete or Update;
* Object – a row of data stored as an object.

The application’s DeQueue process will sit there waiting for data coming in to be added to the queue and will dequeue the item and then process the item in the database.

## To Queue or Not To Queue

There are some entities whose IDs will be required by the application to be available before further processing may be performed. They are:

* Actor;
* Bookmark;
* Collection;
* Country;
* Genre;
* Language;
* Movie;
* Path;
* Role;
* Studio;
* User Group;
* VideoThumbBList;
* Writer.

All of these entities have auto-generated primary keys. Some of these tables do not have dependent entities but contain data that the application will need immediately after the record has been generated.

All of the other entities are dependent on one or more of these entities to already exist. This means that some entities will have to be written to the database differently from others. It is difficult to think of a better way to write records for these entities except by adding them directly to the database at the time of coming across these entities as processing goes forward. This will mean that there may be a performance cost when records are first written for an entity. This is only an issue the first time an entity is added to the database; from then onwards MP will obtain data from the cache.

Dependent entities will be added to the database using a data queueing system in the background. These entities are:

* ActorInfo;
* ActorRole;
* MovieActorRole;
* MovieBookmark;
* MovieCollection;
* MovieCountry;
* MovieGenre;
* MovieInfo;
* MovieLanguage;
* MoviePart;
* MovieResume;
* MovieStudio;
* MovieURL;
* MoveiUserGroup;
* MovieWriter.

All of these entities contain data for which the application can wait for data.

## Multi-user Households

There are three approaches that could be adopted to process data in a multi-user set-up:

1. Use the auto-increment facility for primary keys;
2. Employ a current IDs table (used by Oracle);
3. Generate IDs manually.

### Auto-increment

We have already seen that we have to write a record to the database before we can obtain the record’s ID column value. We also know that some of our entities depend upon other entities having been created, and their IDs generated, before we can create their child entities – the classic chicken and egg scenario. This effectively rules out, in a queueing system, the adoption of auto-increment for all entities that have dependencies.

### Oracle’s IDs Table

Oracle provided an alternative solution to ID generation – do not use auto-increment facility but store the MAX id for each table in a separate table, containing an ID column for each table containing a primary key, and pre-allocate the next write value when wanting to insert a new record. If the commit is successful then the ID is used – otherwise it is forgotten. If we adopt this approach, then we can queue all database writes, only updating the IDs table in real time, and pre-allocate the next IDs before queueing the write allowing the next record to be cached first and then written to the database when it is allocated by the queue. This is an old idea but could work well for our scenario. In our case there is a major drawback with this approach; the database would have to be accessed every time a new record is added to table. In a multi-thread environment, this would mean that there would be a high likelihood that there would be a conflict for resources causing the lookup to fail; meaning that the next ID could fail to be retrieved causing a duplicate ID to be created.

### Manually Generate IDs

We need a way to generate IDs without having to access the database each time. We could manually increment the ID column of each table. This approach would not need to access the database to get the next ID because we would use the cache to maintain records; meaning that we would generate images for each record without any database thrashing and each new record would be queued as originally intended.

It has been proven important not to be too dogmatic about the correct approach. One example of having to adopt an alternative approach is VideoThumbBList processing. It turns out that auto-increment processing works better for VideoThumbBList processing than manual ID incrementing because of the multi-threaded approach to VideoThumbBList processing. This is only workable because VideoThumbBList has no dependencies.

Finally, there will need to be a mechanism in place to cope with more than one attempt to access the database at the same time. This will be handled by a wait process that will be called immediately before a queue process starts. The application will be forced to wait until the database is free before continuing. This can lead to the application appearing to hang, for example when the network is extremely busy, but the application will always return and this will only happen very occasionally. It is a cost of having a multi-user system using a small footprint database.

Database processing may also be blocked when reading from the database when database writes are being performed by another client process. This solution has built the wait process (in the previous paragraph) into the data loading process which happens when MP starts. The cache is used to obtain data for the rest of the time, locally, so the wait process is not required.

## Conclusion

After trying out all of the options above, it was found that the following is the best way to proceed:

* Store all of the current data in lists when the application starts up;
* Manually increment Ids for all tables except VideoThumbBList, keeping the lists up-to-date as processing occurs, queueing database updates;
* Detecting when the database is locked and waiting until the database is free before updating it.

If it is desired that the cache be kept up-to-date as opposed to lasting for as long as the current session, then the cache could be refreshed in the Init method of each Windows Plugin for Music, Picture and Video (Video also taking care of Folder).

# Data Management

Database data in the MP databases will keep growing, even when physical data has been removed, unless the applications do something to deal with redundant data.

The refactored version of MP uses two ways to ensure that out-of-date data is removed from the database:

* On start-up;
* On each media section opening.

## On start-up

When MP starts up, the system checks whether the existing client databases are the correct database versions for the current release. The refactored version performs this task in the Init method of each of the plugins for Folders, Music, Pictures and Videos.

A new task has been added to each of these methods to implement a check to remove data stored in the database for disk files that no longer exist as well as for orphaned data; i.e. data in child tables for which there is no parent record. In a fully managed database engine, the latter should not be able to happen. The refactored version handles this improbable, but possible, occurrence.

When this process has finished, Optimize is run against the database – so all databases are optimized when the system starts up.

NOTE: On start-up, Video media is only checked for the removal of non-stacked files (stacked videos are handled below).

## On each media section opening

When you select any of the media options, like Video, you will be presented with a folder structure. MP detects when new media has been added to the current folder since the last time the folder was used. In the case of Video, the refactored version also checks for whether any stacked files have been removed from disk since the last time the folder was accessed. It needs to do this to make sure that the parts reflect what is on disk and that the screen shows the correct length of the existing parts (disk size, percentage watched and duration).