DB Manager Actor API

This document will detail the implemented DB API via Actor Messaging.

All queries are derived classes of the Query Class, which is an abstract class containing the Query type as String.

All replies are derived classes of the Reply Class, which is an abstract class containing the Reply type as String.

All error or faillure replies are derived classes of the DBOperationFaillure Class, which is itself a derived class of the Reply Class. This DBOperationFaillure contains the faillure cause as a String if the DB manager was able to pinpoint any.

For any query, the DB manager could respond with the UnsupportedOperation reply (derived class of the DBOperationFaillure) in case the received Query Object is not recognized.

Every operation is atomic on the DB, i.e. when there is an operation ongoing no other operation can be run on the DB via the manager (they are queued). The atomicity isn't guaranteed on separated operations, this means that if a client queries the DB multiple times, the result can change based upon what other operations have been asked to the DB Manager (state of the DB isn't mantained between queries, but only inter-query).
To overcome this, the DB Manager implements a Lock for the DB. A client that wants to be assured of the atomicity of a block of queries must first acquire the Lock via the AcquireDBLock operation.
Only the Client who owns the lock is able to run operations on the DB, while other client can only check if whether the DB is locked and by who.
After it has done its work, the client must release the lock to let the DB be queriable from other actors.
The DB Manager has a timeout for unused lock, which means that if the DB is locked but the last received Query was received too far in the past, the DB will unlock (this mechanism is present to overcame client failures that would leave the DB locked forever).

In the following table the queries/reply mapping of the DB API is presented. For all FaillureReplies, the cause field is omitted since it was explained above.

Note: The term "Collection" is intended both for table, collection or in general an aggregation of data which has been put in a category (this depends on the DB convention and structure).
For safety, clients cannot operate on collections directly, but can only access the elements therin. The only avaiable operations at collection level for clients are to check if a single collection is present and to retreive all collections present in the DB.

Client Query	**DB Manager Replies**
AquireDBLock Tries to acquire the DB Lock.	DBLockAcquired *(isa Reply)* Lock Acquired successfully DBIsLockedByYou *(isa DBLockAcquired)* The DB is already locked by yourself. DBFailedToBeLocked *(isa DBOperationFaillure)* Failed to acquire lock. DBIsAlreadyLocked *(isa DBIsAlreadyLocked)* The DB is already locked • Owner: String – Name of the Lock Owner
GetCollectionList Asks for the list of collections present in the DB.	CollectionListSuccess *(isa Reply)* Object containing an ArrayList<String> of all the found collection names. This list can also be empty in the case no collection has been found. CollectionListFaillure *(isa DBOperationFaillure)* Indicates a faillure in obtaining the collectionList.
GetServizioByID Ask the DB to retreive the Servizio associated to the gived ID. • ID: String – ID of the wanted Servizio • collectionName: String – Name of the collection where to look for the Servizio.	ReplyServizioByID *(isa Reply)* Returns the wanted servizio • ID: String ID of the retreived servizio. • servizio: String JSON of the servizio • collectionName: String – Name of the collection where the Servizio was found. ReplyServizioEnrichedByID *(isa Reply)* Returns the wanted servizio as enriched object since the DB has support for it • ID: String - ID of the retreived servizio. • servizio: TommyEnrichedJSON – Object containing an enriched representation of the servizio (the raw servizio is in the json field). • collectionName: String – Name of the collection where the Servizio was found. ServizioByIDNotFound *(isa DBOperationFaillure)* The servizio ID wasn’t present in the given collection • ID: String - ID of the retreived servizio.

	• collectionName: String – Name of the collection where the Servizio was found. CollectionNotFound *(isa DBOperationFaillure)* The given collection is not present in the DB • collectionName: String – Name of the collection where the Servizio was found. GetServizioByIdFaillure *(isa DBOperationFaillure)* Faillure in reading the servizio from the DB • ID: String - ID of the retreived servizio. • collectionName: String – Name of the collection where the Servizio was found.
GetAllServiziInCollection Ask the DB to retreive the all the Servizi in the given Collection. • collectionName: String – Name of the collection where to look for the Servizio.	ReplyServiziInCollection *(isa Reply)* Returns the wanted servizi as Hashmap of ids and jsons • serviziMap: HashMap<String, String> – Map containing all the serviziID as keys and the json of the servizi. • collectionName: String – Name of the collection where tto read the servizi. ReplyServiziInCollectionEnriched *(isa Reply)* Returns the wanted servizi as Hashmap of ids and enriched object since the DB has support for it • serviziMap: HashMap<String, TommyEnrichedJSON> – Map containing all the serviziID as keys and the enriched representation of the servizi as values (the raw servizio is in the json field). • collectionName: String – Name of the collection where tto read the servizi. CollectionNotFound *(isa DBOperationFaillure)* The given collection is not present in the DB • collectionName: String – Name of the collection where the Servizio was found.
IsCollectionByNamePresent Asks if the given collection is present in the DB • collectionName: String – Name of the collection where to look for the Servizio.	CollectionFound *(isa Reply)* The given collection is present in the DB • collectionName: String – Name of the collection. CollectionNotFound *(isa DBOperationFaillure)*

	The given collection is not present in the DB • collectionName: String – Name of the collection.
IsDBAlive Asks the DB Manager if the underlining DB is active and responding	DBIsAlive *(isa Reply)* The DB is alive and well. DBNotAlive *(isa DBOperationFaillure)* Faillure, the DB manager wasn't able to connect to the DB.
IsDBLocked Asks if the DB is locked	DBIsAlreadyLocked *(isa DBOperationFaillure)* The DB is already locked • Owner: String – Name of the Lock Owner DBIsNotLocked *(isa Reply)* DB is not locked (this could change between this and the acquireLock call). DBIsLockedByYou *(isa Reply)* The DB is locked by yourself.
IsDBManagerActive Asks if for the DB Manager Status	Returns a DBManagerStatus, which contains a status String. It is actually divided in the following possibilities: DBManagerActive *(isa DBManagerStatus)* The DB Manager Actor is alive and ready DBManagerInitializing *((isa DBManagerStatus)* The DB Manager Actor is alive but is not ready yet. DBManagerPause *(isa DBManagerStatus)* The DB Manager Actor is alive but was put in a pause state where it won't run any operation (eventual locks will be released in this state) DBManagerErrorState *(isa DBManagerStatus)* The DB Manager Actor is alive but it in erroneus state
MoveServizioByID Asks for the servizio to be moved from the old collection to a new one.	MoveServizioByIDSuccess *(isa Reply)* The move operation was successful

• ID: String ID of the servizio. • oldCollectionName: String – Name of the collection where the Servizio is to be found initially. • newCollectionName: String – Name of the collection where the Servizio is to be moved.	• ID: String ID of the retreived servizio. • oldCollectionName: String – Name of the collection where the Servizio was found initially. • newCollectionName: String – Name of the collection where the Servizio was moved. MoveServizioByIDFaillure *(isa DBOperationFaillure)* The move operation was not successful. The DB Manager will create a backup of the service tol try to restore the original state of the DB in case of faillure. • ID: String ID of the retreived servizio. • oldCollectionName: String – Name of the collection where the Servizio was found initially. • newCollectionName: String – Name of the collection where the Servizio was moved ServizioByIDNotFound *(isa DBOperationFaillure)* The servizio ID wasn't present in the old collection • ID: String - ID of the retreived servizio. • oldCollectionName: String – Name of the collection where the Servizio was found initially. • newCollectionName: String – Name of the collection where the Servizio was to be moved. ServizioByIDAlreadyPresentInCollection *(isa DBOperationFaillure)* The servizio ID was already present in the new collection • ID: String - ID of the retreived servizio. • collectionName: String – Name of the collection where the Servizio was to be moved. CollectionNotFound *(isa DBOperationFaillure)* The one of the given collections is not present in the DB • collectionName: String – Name of the collection not found.
ReleaseDBLock Ask the DB manager to release the caller as owner of the DB Lock	DBLockReleased *(isa Reply)* The DB Lock was released.

	DBIsNotLocked *(isa Reply)* The DB wasn't locked to begin with so this call wasn't effective. DBLockReleaseFaillure *(isa DBOperationFaillure)* Faillure in releasing the lock (maybe timeout will succed). DBLockReleaseUnowning *(isa DBLockReleaseFaillure)* The calling client didn't own the Lock so cannot release it.
RemoveServizioByID Asks for the servizio to be removed from the DB. • ID: String ID of the servizio. • collectionName: String – Name of the collection where the Servizio is to be found.	RemoveServizioByIDSuccess *(isa Reply)* The servizio was removed from the DB • ID: String ID of the removed servizio. • servizio: String JSON of the removed servizio • collectionName: String – Name of the collection where the Servizio was removed from. ServizioByIDNotFound *(isa DBOperationFaillure)* The servizio ID wasn't present in the given collection • ID: String - ID of the servizio to be removed. • collectionName: String – Name of the collection where the Servizio was to be found. CollectionNotFound *(isa DBOperationFaillure)* The given collection is not present in the DB • collectionName: String – Name of the collection where the Servizio was to be found. RemoveServizioByIdFaillure *(isa DBOperationFaillure)* Faillure in removing the servizio from the DB • ID: String - ID of the servizio to be removed. • collectionName: String – Name of the collection where the Servizio was to be removed.
UpdateServizioByID	UpdateServizioByIDSuccess

Asks for the servizio to be updated in the DB with the data of a new one. • ID: String ID of the servizio. • servizio: String JSON of the new servizio • collectionName: String – Name of the collection where the Servizio is to be found.	*(isa Reply)* The servizio was updated correctly • ID: String ID of the servizio. • oldServizio: String JSON of the old servizio • updatedServizio: String JSON of the updated servizio • collectionName: String – Name of the collection where the Servizio was found. <u>UpdateServizioByIDEnrichedSuccess</u> *(isa Reply)* The servizio was updated and written as enriched object since the DB has support for it • ID: String - ID of the retreived servizio. • oldServizio: TommyEnrichedJSON – Object containing an enriched representation of the old servizio (the raw old servizio is in the json field). • updatedServizio: TommyEnrichedJSON – Object containing an enriched representation of the updated servizio (the raw servizio is in the json field). • collectionName: String – Name of the collection where the Servizio was found. <u>ServizioNotValid</u> *(isa DBOperationFaillure)* The given servizio JSON was invalid (the parsing operation to Servizio Object failed) • ID: String ID of the servizio. • newServizio: String JSON of the new servizio. • collectionName: String – Name of the collection where the Servizio was to be found. <u>ServizioByIDAlreadyPresentInCollection</u> *(isa DBOperationFaillure)* The servizio ID wasn’t present in the given collection. Update operation only works if there is already a servizio in the DB. (To write a new servizio, there is the WriteNewServizioByID Operation) • ID: String ID of the servizio. • newServizio: String JSON of the new servizio. • collectionName: String – Name of the collection where the Servizio was to be

	found. CollectionNotFound *(isa DBOperationFaillure)* The given collection is not present in the DB • collectionName: String – Name of the collection where the Servizio was to be found. UpdateServizioByIdFaillure *(isa DBOperationFaillure)* Faillure in updating the servizio • ID: String - ID of the retreived servizio. • newServizio: String JSON of the new servizio. • collectionName: String – Name of the collection where the Servizio was found.
WriteNewServizioByID Asks for the servizio to be written in the DB. • ID: String ID of the servizio. • servizio: String JSON of the servizio • collectionName: String – Name of the collection where the Servizio is to be written.	WriteNewServizioByIDSuccess *(isa Reply)* The servizio was written correctly • ID: String ID of the servizio. • servizio: String JSON of the written servizio • collectionName: String – Name of the collection where the Servizio was found. WriteNewServizioByIDEnrichedSuccess *(isa Reply)* The servizio was written correctly as enriched object since the DB has support for it • ID: String - ID of the retreived servizio. • oldServizio: TommyEnrichedJSON – Object containing an enriched representation of the old servizio (the raw old servizio is in the json field). • updatedServizio: TommyEnrichedJSON – Object containing an enriched representation of the updated servizio (the raw servizio is in the json field). • collectionName: String – Name of the collection where the Servizio was found. ServizioNotValid *(isa DBOperationFaillure)* The given servizio JSON was invalid (the parsing operation to Servizio Object failed) • ID: String ID of the servizio. • newServizio: String JSON of the new servizio. • collectionName: String – Name of the

	collection where the Servizio was to be found. ServizioByIDNotFound *(isa DBOperationFaillure)* The servizio ID wasn't present in the given collection. Update operation only works if there is already a servizio in the DB. (To write a new servizio, there is the WriteNewServizioByID Operation) • ID: String ID of the servizio. • newServizio: String JSON of the new servizio. • collectionName: String – Name of the collection where the Servizio was to be found. CollectionNotFound *(isa DBOperationFaillure)* The given collection is not present in the DB • collectionName: String – Name of the collection where the Servizio was to be found. WriteNewServizioByIDFaillure *(isa DBOperationFaillure)* Faillure in updating the servizio • ID: String - ID of the retreived servizio. • newServizio: String JSON of the new servizio. • collectionName: String – Name of the collection where the Servizio was found.