MANUALE

DaamApisCollection

Manuale d’uso della collezione di API multipurpose

V. 1.3.2023.3

.Net Framework 4.8

SOMMARIO

1. DaamApisCollection

DaamApisCollection è una collezione di librerie le cui classi (utilizzate su vari software nell'arco del tempo) consentono l'interfaccia con il database, compreso le classi di creazione delle istruzioni MySql, il parsing e la scrittura di file XML, e la validazione dei campi di inserimento all'interno di un form.

In questo manuale cono elencati i namespace contenuri nella collezione, le loro classi con i relativi metodi e attributi, oltre ad essere riportati esempi di utilizzo.

Sarà fondamentale in futuro consulatare tale guida prima di utilizzare la libreria all'interno di un software, onde evitare errori di utilizzo o il discostamento dallo standard stabilito per la realizzazione delle classi funzionali al software.

1.1. Lista di namespace contenuti nella collection

1. DaamChecking
2. DaamDBManager
3. DaamXMLParser

2. DaamChecking

Namespace contenente le classi per la validazione e il refactoring dei valori inseriti all'interno degli items di inserimento, quali TextBox, ListBox, ComboBox ecc.

2.1 Lista delle Classi

- Checking
- RefactoringValue

2.2 Checking

Descrizione: Classe per la verifica dei campi di inserimento. Può essere associata a un file di gestione degli errori attraverso il setter, in tal caso i metodi vanno utilizzati in un blocco Try Catch per la gestione delle eccezioni, genera Exception. Se non viene specificato il file, i metodi si comportano come puri booleani e non generano alcuna eccezione.

Using: System; DaamXMLParser

2.2.1 Attributi:

NOME	TIPO	VISIBILITA	DESCRIZIONE
pathXMlErrorFile	String	Private	Attributo contenente il percorso del file XML con la lista errori. Acquisisce il valore dal setter
reader	ReadingXML	Private	Oggetto di istanza della classe ReadingXML
PathXMlErrorFile	String	Public	Setter per l'acquisizione del percorso del file con la lista errori. Non obbligatorio

2.2.2 Metodi:

NOME	TIPO	VISIBILITA	ATTRIBUTI	RETURN	DESCRIZIONE
Checking	Costruttore	Public			Costruttore della classe
isEmpty	Bool	Public	String value	Restituisce true se vuota	Metodo per la verifica di stringa vuota.
isNumeric	Bool	Public	String value	Restituisce true se numerico	Metodo per la verifica di un valore numerico intero
isNumericDouble	Bool	Public	String value	Restituisce true se numerico	Metodo per la verifica di un valore numerico double
isNumericFloat	Bool	Public	String value	Restituisce true se numerico	Metodo per la verifica di un valore numerico float
inRange polimorfo	Bool	Public	String value; int rangeMin; int rangeMax	Restituisce true se in range	Metodo per la verifica di un valore stringa all'interno di un range numerico intero
inRange polimorfo	Bool	Public	int value; int rangeMin; int rangeMax	Restituisce true se in range	Metodo per la verifica di un valore intero all'interno di un range numerico intero

2.3 RefactoringValue

Descrizione: Classe per il refactoring di un valore stringa o double. Utile quando si ha la necessità di troncare un valore numerico double con troppi decimali oltre la virgola, oppure quando si vuole troncare una stringa ad un determinato carattere.

Using: System;

2.3.1 Attributi:

2.3.2 Metodi:

NOME	TIPO	VISIBILITA	ATTRIBUTI	RETURN	DESCRIZIONE
RefactoringValue	Costruttore	Public			Costruttore della classe
truncateDoubleValue	String	Public	String value; int maxDecimal = 0	Restituisce il valore troncato in string	Metodo per la troncatura di un valore double passato come stringa al decimale settato. Se non specificato tronca al secondo decimale dopo la virgola
truncateString	String	Public	String value; char symbolToTruncate; int nChars = 0	Restituisce la stringa troncata	Metodo per la troncatura di una stringa al numero di caratteri specificato dopo il simbolo passato come parametro. Se non specificato il numero di caratteri tronca dopo 2.

3 DaamDBManager

Namespace contenente le classi per lo scambio dati verso il Database e per la creazione delle query DML e DQL del linguaggio MySql. Contiene anche l'interfaccia da implementare per la costruzione standard delle classi Model.

3.1 Lista delle Classi

- Connecting
- DML
- DQL
- TemplateModelData<T>

3.2 Connecting

Descrizione: Classe per la gestione dell'interfaccia verso il Database. Utilizza il connettore MySql v. 8.0.30. Da utilizzare all'interno di un blocco Try Catch, genera Exception in caso di errore di connessione; non è collegato ad alcun file per la gestione degli errori.

Using: System; System.Data; MySql.Data.MySqlClient

3.2.1 Attributi:

NOME	TIPO	VISIBILITA	DESCRIZIONE
strConnection	String	Private	Attributo contenente la stringa di connessione per il DB

3.2.2 Metodi:

NOME	TIPO	VISIBILITA	ATTRIBUTI	RETURN	DESCRIZIONE
Connecting	Costruttore	Public	String strConnection		Costruttore della classe
getOpenConnection	MySqlConnection	Public		Restituisce una connesione aperta	Metodo per aprire una connessione verso un DB MySql
getCommand	MySqlCommand	Public	MySqlConnection connection; string query	Restituisce un command con la query associata	Metodo per ottenere un command con query associata da passare a un reader o un writer
getReader polimorfo	MySqlReader	Public	MySqlCommand command	Restituisce un reader	Metodo per ottenere un reader che ha eseguito la query associata al command
getReader polimorfo	MySqlReader	Public	MySqlCommand command; string query	Restituisce un reader	Metodo per ottenere un reader che ha eseguito la query passata e non associata precedentemente ad un command
bindingDataTable	Void	Public	MySqlCommand command; string query		Metodo per popolare un DataTable attraverso un MySqlDataAdapter che esegue la query associata al command
Writer	Int	Public	MySqlCommand command;	Restituisce il numero di righe interessate dalla query	Metodo per l'inserimento di un record all'interno del DB attraverso un command con la query di inserimento associata.

3.3 DML

Descrizione: Classe per la creazione della query relativa ai comandi Data Manipulation Language di MySql. Il comando viene rilasciato in formato stringa per poter essere passato ad un command della classe Connecting. Da utilizzare all'interno di un blocco Try Catch, genera ArgumentException, non utilizza il file XML con la lista errori. Classe ricorsiva ogni metodo ritorna un oggetto di tipo DML.

Using: System;

3.3.1 Attributi:

NOME	TIPO	VISIBILITA	DESCRIZIONE
cudOperarion	Static String	Private	Attributo contenete la query formata da restituire attraverso il getter
nColums	Int	Private	Numero di colonne riportate nel metodo Fields. Necessario per verificare che il numero di valori inseriti nel metodo Values sia uguale al numero di colonne create

3.3.2 Metodi:

NOME	TIPO	VISIBILITA	ATTRIBUTI	RETURN	DESCRIZIONE
DML	Costruttore	Public			Costruttore della classe
InsertInto	DML	Public	String table	Restituisce oggetto DML	Metodo per la Creazione del comando INSERT INTO "tabella"
Fields	DML	Public	Params String[] fields	Restituisce oggetto DML	Metodo per definire il nome delle colonne per l'inserimento del record
Values Verifica il numero di valori con il numero di colonne	DML	Public	Params String[] values	Restituisce oggetto DML	Metodo per attribuire i valori alle colonne del metodo Fields per inserire il record nel DB
Update	DML	Public	String table	Restituisce oggetto DML	Metodo per la creazione del comando "UPDATE"
Set Verifica il numero di valori con il numero di colonne	DML	Public	Params String[] fieldsEqualsValues	Restituisce oggetto DML	Metodo per attribuire i valori alle colonne in accoppiata "field" = "value"
DeleteFrom	DML	Public	String table	Restituisce oggetto DML	Metodo per la creazione del comando DELETE FROM
Where	DML	Public	String field	Restituisce oggetto DML	Metodo per inserire il campo Key da verificare per le clausule di integrità
Equals	DML	Public	Int value; double value; string value (un attributo per ogni metodo polimorfo)	Restituisce oggetto DML	Metodo per inserire l'operatore di confornto "=" e il valore da confrontare. Polimorfo per i possibili attributi passabili da confrontare.

Like	DML	Public	String value	Restituisce oggetto DML	Metodo per inserire il metodo di confronto LIKE. Sono già previsti i limiti %% per la corrispondenza completa.
And	DML	Public	String field	Restituisce oggetto DML	Metodo per inserire un ulteriore campo di confronto per le clausule di integrità
endLine	DML	Public		Restituisce oggetto DML	Metodo per l'inserimento del simbolo di fine riga
getCudOperation	String	Public		Restituisce la stringa con la query	Getter per la restituzione della query ottenuta dall'implementazione dei vari metodi in modo concatenato

3.4 DQL

Descrizione: Classe per la creazione della query relativa ai comandi Data Query Language di MySql. Il comando viene rilasciato in formato stringa per poter essere passato ad un command della classe Connecting. Da utilizzare all'interno di un blocco Try Catch, genera ArgumentException, non utilizza il file XML con la lista errori. Classe ricorsiva ogni metodo ritorna un oggetto di tipo DQL.

Using: System;

3.4.1 Attributi:

NOME	TIPO	VISIBILITA	DESCRIZIONE
query	Static String	Private	Attributo contenete la query formata da restituire attraverso il getter

3.4.2 Metodi:

NOME	TIPO	VISIBILITA	ATTRIBUTI	RETURN	DESCRIZIONE
DQL	Costruttore	Public			Costruttore della classe
Select	DQL	Public	String field = "*"; bool distinct=false	Restituisce oggetto DQL	Metodo per la Creazione del comando SELECT "campi"
From	DQL	Public	String table	Restituisce oggetto DQL	Metodo per definire il nome della tabella in cui fare il mapping
InnerJoin	DQL	Public	String table	Restituisce oggetto DQL	Metodo per la creazione del commando InnerJoin per il join tra tabelle
Where	DQL	Public	String field	Restituisce oggetto DQL	Metodo per inserire il campo Key da verificare per le clausule di integrità
Limit	DQL	Public	Int startIndex; int maxRows	Restituisce oggetto DQL	Metodo per inserire la clausula di limit per la paginazione di una tabella
ON	DQL	Public	String field	Restituisce oggetto DQL	Metodo per inserire il campo Key da verificare per le clausule di integrità da utilizzare con il metodo InnerJoin
Equals	DQL	Public	Int value; double value; string value (un attributo per ogni metodo polimorfo)	Restituisce oggetto DQL	Metodo per inserire l'operatore di confornto "=" e il valore da confrontare. Polimorfo per i possibili attributi passabili da confrontare.
Like	DQL	Public	String value	Restituisce oggetto DQL	Metodo per inserire il metodo di confronto LIKE. Sono già previsti i limiti %% per la corrispondenza completa.

And	DQL	Public	String field	Restituisce oggetto DQL	Metodo per inserire un ulteriore campo di confronto per le clausule di integrità
endLine	DQL	Public		Restituisce oggetto DQL	Metodo per l'inserimento del simbolo di fine riga
getQuery	String	Public		Restituisce la stringa con la query	Getter per la restituzione della query ottenuta dall'implementazione dei vari metodi in modo concatenato

3.5 TemplateModelData<T>

Descrizione: Classe astratta per la definizione di un pattern da utilizzare nella creazione delle classi Model, le quali avranno il compito di scambiare i dati verso il DB. Con questa classe, e attraverso l'uso della seguente libreria, si intende creare un modello standard, valido per tutti i software, di gestione dati verso il DB. Il parametro <T> serve a passare l'ADT del record di una tabella, il quale è definito attraverso una struct, ai vari metodi della classe. Per mantenere la flessibilità di utilizzo, tutti i metodi sono virtual e non sono compresi metodi abstract.

Using: System.Data; System.Collections.Generic

3.5.1 Attributi in override:

NOME	TIPO	VISIBILITA	DESCRIZIONE
list	Virtual List<T>	Private	Attributo privato per la resituzione di una List<T> attraverso il metodo getAll(bool condition)
CurrentPage	Virtual Int	public	Getter per ottenere la pagina corrente in caso di estrazione dei dati con tabulazione
Pages	Virtual Int	Public	Getter per ottenere il numero di pagine totali in caso di estrazione dei dati con tabulazione
GetTable	Virtual DataTable	Public	Getter per ottenere la tabella definita nei suoi campi dal costruttore della classe, e popolata dai metodi della classe

3.5.2 Metodi in override:

NOME	TIPO	VISIBILITA	ATTRIBUTI	RETURN	DESCRIZIONE
getAll polimorfo	Virtual Bool	Public	T record	Ritorna true se l'estrazione è avvenuta correttamente	Metodo per ottenere tutti i record di una tabella, popola il DataTable
getAll polimorfo	Virtual Bool	Public	T record; String param	Ritorna true se l'estrazione è avvenuta correttamente	Metodo per ottenere tutti i record di una tabella con parametro aggiuntivo per vincolo di integrità, popola il DataTable
getAll polimorfo	Virtual List<T>	Public	Bool condition	Restituisce List di tipo T passato nella dichiarazione d'interfaccia	Metodo per ottenere tutti i record di una tabella, rilasciati in una lista di tipo struct del record
getAll polimorfo	Virtual DataTable	Public		Restituisce un DataTable popolato dai record	Metodo per ottenere tutti i record di una tabella rilasciati in un DataTable popolato
selectData	Virtual Bool	Public	T record; int indexStart = 0	Restituisce true se il mapping è avvenuto correttamente	Metodo per ottenere uno specifico range di record a partire da uno specifico indice, popola il DataTable
find	Virtual Bool	Public	T record	Restituisce true se il mapping è avvenuto correttamente	Metodo per cercare uno o più record in base ai campi di ricerca passati dall'oggetto struct

find	Virtual Bool	Public	T record; string param	Restituisce true se il mapping è avvenuto correttamente	Metodo per cercare uno o più record in base ai campi di ricerca passati dall'oggetto struct e da un parametro aggiuntivo
innerJoin	Virtual T	Public	`int indexForeignKey`	Restituisce un oggetto di tipo T	Metodo per ottenere uno o più record tramite la verifica di un vincolo di integrità referenziale
forward	Virtual Bool	Public	T record	Restituisce true se il mapping è avvenuto correttamente	Metodo per ottenere uno specifico range di record aumentando di 1 la pagina rispetto a selectData, popola il DataTable
back	Virtual Bool	Public	T record	Restituisce true se il mapping è avvenuto correttamente	Metodo per ottenere uno specifico range di record diminuendo di 1 la pagina rispetto a selectData, popola il DataTable
insert	Virtual Bool	Public	T record; string param = ""	Restituisce true se inserimento avvenuto correttamente	Metodo per inserire un record in una tabella del DB, popola il DataTable
update	Virtual Bool	Public	T record; string param = ""	Restituisce true se aggiornamento avvenuto correttamente	Metodo per aggiornare un record in una tabella del DB, popola il DataTable
delete	Virtual Bool	Public	T record; string param = ""	Restituisce true se eliminazione avvenuta correttamente	Metodo per eliminare un record in una tabella del DB, popola il DataTable

4 DaamXMLParser

Namespace contenente le classi necessarie alla lettura e alla scritturi di file XML.

4.1 Lista delle Classi

- ReadingXML
- WriterXML

4.2 ReadingXML

Descrizione: Classe per il parsing di un file XML. Il percorso del file da leggere viene passato alla classe attraverso il costruttore. Da utilizzare all'interno di un blocco Try Catch genera Exception. Per la lettura del singolo nodo mette a disposizione 2 metodi. Il metodo readNode necessita del solo nome del nodo per poterlo leggere, utilizza l'API c# XmlReader. Mentre il metodo readNodeFromPath necessita del percorso gerarchico del nodo da leggere (es. "/father/son"), utilizza l'API c# XmlNode (metodo da usare preferibilmente).

Using: System; System.Collections.Generic; System.Xml

4.2.1 Attributi:

NOME	TIPO	VISIBILITA	DESCRIZIONE
pathFile	String	Private	Attributo contenente il percorso del file dal leggere, viene valorizzato con l'associazione nel costruttore

4.2.2 Metodi:

NOME	TIPO	VISIBILITA	ATTRIBUTI	RETURN	DESCRIZIONE
ReadingXML	Costruttore	Public	String pathFile		Costruttore della classe
readNode	String	Public	String node	Restituisce il valore contenuto nel nodo	Metodo per leggere un singolo nodo, necessita del solo nome del nodo da leggere
readNodesFromPath	List<string>	Public	String pathNodes	Restituisce una lista di stringhe	Metodo per la lettura di più nodi identici nel nome, necessita del percorso gerarchico dei nodi da leggere.
readNodeFromPath	String	Public	String pathNode	Restituisce il valore contenuto nel nodo	Metodo per leggere un singolo nodo, necessita del percorso gerarchico del nodo da legger

4.3 WriterXML

Descrizione: Classe per la sola modifica di un file XML. Consente di modificare il valore contenuto all'interno di uno specifico nodo. Non prevede metodi per la creazione da zero di un file XML, in quanto è possibile farlo attraverso le stringe e utilizzando i metodi di c# per la creazione dei file.

Using: System; System.Xml

4.3.1 Attributi:

NOME	TIPO	VISIBILITA	DESCRIZIONE
pathFile	String	Private	Attributo contenente il percorso del file dal modificare, viene valorizzato con l'associazione nel costruttore

4.3.2 Metodi:

NOME	TIPO	VISIBILITA	ATTRIBUTI	RETURN	DESCRIZIONE
WriterXML	Costruttore	Public	String pathFile		Costruttore della classe
reWriteNode	Bool	Public	String pathNode; string insertValue	Restituisce true se la modifica avviene correttamente	Metodo per la modifica del valore di un singolo nodo, necessita del percorso gerarchico del nodo da modificare

5 Esempi di utilizzo delle classi

Esempio pratico per la creazione di una classe tipo Model, utilizzando i namespace DaamDBManager e DaamXMLParser. Per la gestione dei file XML, in questo esempio si vede soltanto l'utilizzo delle classi per il parser, in un esempio successivo si mostra l'utilizzo delle classi per il writer.

5.1 Esempio classe Model:

```
namespace ModelData
{
   /// <summary>
   /// Struttura che rappresenta la tupla della tabella mesi
   /// </summary>
   public struct RecordMese
   {
      /// <summary>
      /// Primary Key della tabella mesi
      /// </summary>
      public int id_mese;
      /// <summary>
      /// Causale di spesa
      /// </summary>
      public string voce_spesa;
      /// <summary>
      /// Note aggiuntive
      /// </summary>
      public string note;
      /// <summary>
      /// Importo della spesa
      /// </summary>
      public double importo;
      /// <summary>
      /// Foreign key per il vincolo di integrità referenziale con la tabella anni
      /// </summary>
      public RecordAnno recordAnno;
   }

   /// <summary>
   /// Classe per la gestione delle tabelle dei mesi
   /// </summary>
   public class Mesi : GeneralModelData<RecordMese>
   {
      /// <summary>
      /// Attributo privato rappresenta la struttura della tabella da passare al DataGridView
      /// </summary>
      private DataTable table;

      /// <summary>
      /// Attributo privato per l'istanza alla classe Connecting
      /// </summary>
      protected Connecting connecting;

      /// <summary>
      /// Attributo privato per l'istanza alla classe ReadingXML
      /// </summary>
      protected ReadingXML readerXML;


      /// <summary>
      /// Costruttore per la classe Mesi per la gestione delle tabelle dei mesi
```

```
/// </summary>
public Mesi()
{
   //Definizione del DataTable per il DataGridView
   table = new DataTable();

   //La colonna ID deve essere un array per poter essere passata
   //come chiave primaria
   DataColumn[] id_mese = new DataColumn[1];
   id_mese[0] = new DataColumn();
   id_mese[0].ColumnName = "ID";
   id_mese[0].DataType = Type.GetType("System.Int32");
   id_mese[0].AutoIncrement = true;

   DataColumn voce_spesa = new DataColumn();
   voce_spesa.ColumnName = "CAUSALE DI SPESA";
   voce_spesa.DataType = Type.GetType("System.String");

   DataColumn note = new DataColumn();
   note.ColumnName = "NOTE";
   note.DataType = Type.GetType("System.String");
   note.AllowDBNull = true;

   DataColumn importo = new DataColumn();
   importo.ColumnName = "IMPORTO";
   importo.DataType = Type.GetType("System.Double");


   table.Columns.Add(id_mese[0]);
   table.Columns.Add(voce_spesa);
   table.Columns.Add(note);
   table.Columns.Add(importo);

   //Istanza alla classe Connecting
   connecting = new Connecting(StringConnection.get());
   //Istanza alla classe ReadingXML
   readerXML = new ReadingXML(Routes.XMLERRORS);

}

/// <summary>
/// Getter del DataTable Mesi
/// </summary>
/// <returns>Restituisce il DataTable definito e valorizzato per il DataGridView</returns>
public override DataTable GetTable
{
   get { return table; }
}

/// <summary>
/// Metodo per il mapping della tabella mese prescelto per il modulo spese mensili
/// </summary>
/// <param name="record">Foreign Key per l'integrità referenziale con la tabella anno</param>
/// <param name="mese">Mese di cui caricare i dati</param>
/// <returns>Ritorna true se i dati sono letti correttamente</returns>
public override bool getAll(RecordMese record, string mese)
{
   //Istanza alla classe DQL per la costruzione delle query
   DQL dql = new DQL();
   //Variabile per il response di lettura dati da DB
   bool response = false;

   try
   {
      //Ottiene una connessione aperta
      var connection = connecting.getOpenConnection();
      //Crea la query
```

```
            string query =
dql.Select().From($"{mese}").Where("anno").Equals(record.recordAnno.anno).endLine().getQuery();
            //Ottiene un comando da eseguire su DB
            var command = connecting.getCommand(connection, query);
            //Ottiene il reader della tabella dal command
            var reader = connecting.getReader(command);

            //Inizializza le righe della tabella a null per caricare i nuovi dati
            table.Rows.Clear();

            //Cicla i risultati ottenuti dalla tabella
            while (reader.Read())
            {
               DataRow row = table.NewRow();
               row["ID"] = reader.GetInt32($"id_{mese}");
               row["CAUSALE DI SPESA"] = reader.GetString("voce_spesa");
               row["NOTE"] = reader.GetValue(2).ToString();//GetValue per accettare i valori nulli da DB
               row["IMPORTO"] = reader.GetDouble("importo");
               table.Rows.Add(row);
               table.AcceptChanges();
            }

            //Set il response a true
            response = true;

            //Scarica le risorse
            connection.Close();
            command.Dispose();
            reader.Close();

         }
         catch (Exception ex)
         {
            MessageBox.Show(readerXML.readNodeFromPath("/ListError/Error9") + "/n" + ex.Message, "ERRORE",
               MessageBoxButtons.OK, MessageBoxIcon.Error);
         }

         //Restituisce il response di lettura dati da DB
         return response;

      }

      /// <summary>
      /// Metodo per la ricerca di una specifica causale di spesa
      /// </summary>
      /// <param name="record">Record da cercare nella tabella</param>
      /// <param name="mese">Tabella del mese in cui cercare</param>
      /// <returns>Ritorna true se la causale viene trovata nel DB</returns>
      public override bool find(RecordMese record, string mese)
      {
         //Istanza alla classe DQL per la costruzione delle query
         DQL dql = new DQL();
         //Variabile per il response di lettura dati da DB
         bool response = false;

         try
         {
            //Ottiene una connessione aperta
            var connection = connecting.getOpenConnection();
            //Crea la query
            string query = dql.Select().From($"{mese}").Where("anno").Equals(record.recordAnno.anno)
               .And("voce_spesa").Like(record.voce_spesa).endLine().getQuery();
            //Ottiene un comando da eseguire su DB
            var command = connecting.getCommand(connection, query);
            //Ottiene il reader della tabella dal command
            var reader = connecting.getReader(command);
```

```
            //Inizializza le righe della tabella a null per caricare i nuovi dati
            table.Rows.Clear();

            //Cicla i risultati ottenuti dalla tabella
            while (reader.Read())
            {
               DataRow row = table.NewRow();
               row["ID"] = reader.GetInt32($"id_{mese}");
               row["CAUSALE DI SPESA"] = reader.GetString("voce_spesa");
               row["NOTE"] = reader.GetValue(2).ToString();//GetValue per accettare i valori nulli da DB
               row["IMPORTO"] = reader.GetDouble("importo");
               table.Rows.Add(row);
               table.AcceptChanges();
            }

            //Set il response a true
            response = true;

            //Scarica le risorse
            connection.Close();
            command.Dispose();
            reader.Close();

         }
         catch (Exception ex)
         {
            MessageBox.Show(readerXML.readNodeFromPath("/ListError/Error9") + "/n" + ex.Message, "ERRORE",
               MessageBoxButtons.OK, MessageBoxIcon.Error);
         }

         //Restituisce il response di lettura dati da DB
         return response;

      }

      /// <summary>
      /// Metodo pe inserire un nuovo record in tabella
      /// </summary>
      /// <param name="recordMese">Record da inserire in tabella</param>
      /// <param name="mese">Mese in cui inserire la voce di spesa</param>
      /// <returns>Ritorna true se il record viene inserito correttamente</returns>
      public override bool insert(RecordMese recordMese, string mese)
      {
         //Variabile per il risultato dell'inserimento
         bool result = false;
         //Istanza alla classe DQL per la costruzione delle query
         DML dml = new DML();
         DQL dql = new DQL();

         try
         {
            //Ottiene una connessione aperta
            var connection = connecting.getOpenConnection();
            //Sostituisce la virgola con il punto per l'inserimento nel DB
            string import = recordMese.importo.ToString().Replace(',', '.');
            //Crea la query di inserimento nel DB
            string query = dml.InsertInto($"{mese}").Fields("voce_spesa", "note", "importo", "anno")
               .Values($"'{recordMese.voce_spesa}'", $"'{recordMese.note}'", $"{import}",
$"{recordMese.recordAnno.anno}")
               .endLine().getCudOperation();
            //Genera il comando con la query
            var command = connecting.getCommand(connection, query);
            //Ricava il numero di righe interessate dall'inserimento
            int insertRows = connecting.writer(command);
            //Scarica il command per la query di prelievo della primary key
            command.Dispose();
```

```
        //Se avvenuto l'inserimento nel DB, aggiunge la riga alla tabella
        if (insertRows > 0)
        {
            //Preleva la primary key dopo l'inserimento
            //Necessaria per il corretto allineamento della tabella
            //Alla generazione di un nuovo anno
            query = dql.Select($"MAX(id_{mese}) as pk").From($"{mese}").endLine().getQuery();
            command = connecting.getCommand(connection, query);
            var reader = connecting.getReader(command);
            if (reader.Read()) recordMese.id_mese = reader.GetInt32("pk");

            //Inserisce il record in tabella
            DataRow row = table.NewRow();
            row["ID"] = recordMese.id_mese;
            row["CAUSALE DI SPESA"] = recordMese.voce_spesa;
            row["NOTE"] = recordMese.note;
            row["IMPORTO"] = recordMese.importo;
            table.Rows.Add(row);
            table.AcceptChanges();
            result = true;
        }
        else
        {
            //Messaggio di errore
            MessageBox.Show(readerXML.readNodeFromPath("/ListError/Error11") +
                "/n Verificare i dati di inserimento", "ERRORE",
                MessageBoxButtons.OK, MessageBoxIcon.Error);
        }

        //Scarica le risorse
        connection.Close();
        command.Dispose();

    }
    catch (Exception ex)
    {
        //Messaggio di errore
        MessageBox.Show(readerXML.readNodeFromPath("/ListError/Error11") + "/n" + ex.Message, "ERRORE",
            MessageBoxButtons.OK, MessageBoxIcon.Error);
    }

    //Ritorna l'esito dell'inserimento
    return result;

}

/// <summary>
/// Metodo per aggiornare un record
/// </summary>
/// <param name="recordMese">Record contenente i dati da aggiornare su DB</param>
/// <param name="mese">Mese in cui aggiornare la voce di spesa</param>
/// <returns>Ritorna true se l'aggiornamento è avvenuto correttamente</returns>
public override bool update(RecordMese recordMese, string mese)
{
    //Variabile per il risultato dell'inserimento
    bool result = false;
    //Istanza alla classe DQL per la costruzione delle query
    DML dml = new DML();

    try
    {
        //Ottiene una connessione aperta
        var connection = connecting.getOpenConnection();
        //Sostituisce la virgola con il punto per l'inserimento nel DB
        string import = recordMese.importo.ToString().Replace(',', '.');
        //Crea la query di aggiornamento nel DB
```

```
            string query = dml.Update($"{mese}").Set($"voce_spesa='{recordMese.voce_spesa}'",
$"note='{recordMese.note}'",
                $"importo={import}").Where($"id_{mese}").Equals(recordMese.id_mese)
                .And("anno").Equals(recordMese.recordAnno.anno).endLine().getCudOperation();
            //Genera il comando con la query
            var command = connecting.getCommand(connection, query);
            //Ricava il numero di righe interessate dall'aggiornamento
            int insertRows = connecting.writer(command);

            //Se avvenuto l'aggiornamento nel DB, aggiunge la riga alla tabella
            if (insertRows > 0)
            {
                //Cerca la riga della tabella dalla primary key
                DataRow row = table.Rows.Find(recordMese.id_mese);
                row["CAUSALE DI SPESA"] = recordMese.voce_spesa;
                row["NOTE"] = recordMese.note;
                row["IMPORTO"] = recordMese.importo;
                table.AcceptChanges();
                result = true;
            }
            else
            {
                //Messaggio di errore
                MessageBox.Show(readerXML.readNodeFromPath("/ListError/Error15") +
                    "/n Verificare i dati di inserimento", "ERRORE",
                    MessageBoxButtons.OK, MessageBoxIcon.Error);
            }

            //Scarica le risorse
            connection.Close();
            command.Dispose();

        }
        catch (Exception ex)
        {
            //Messaggio di errore
            MessageBox.Show(readerXML.readNodeFromPath("/ListError/Error11") + "/n" + ex.Message, "ERRORE",
                MessageBoxButtons.OK, MessageBoxIcon.Error);
        }

        //Ritorna l'esito dell'inserimento
        return result;

    }

    /// <summary>
    /// Metodo per eliminare un record dalla tabella
    /// </summary>
    /// <param name="recordMese">Record da eliminare in tabella</param>
    /// <param name="mese">Mese in cui eliminare la voce di spesa</param>
    /// <returns>Ritorna true se l'eliminazione è avvenuta correttamente</returns>
    public override bool delete(RecordMese recordMese, string mese)
    {
        //Variabile per il risultato dell'inserimento
        bool result = false;
        //Istanza alla classe DQL per la costruzione delle query
        DML dml = new DML();

        try
        {
            //Ottiene una connessione aperta
            var connection = connecting.getOpenConnection();
            //Crea la query di delete del record nel DB
            string query = dml.DeleteFrom($"{mese}").Where($"id_{mese}").Equals(recordMese.id_mese)
                .And("anno").Equals(recordMese.recordAnno.anno).endLine().getCudOperation();
            //Genera il comando con la query
            var command = connecting.getCommand(connection, query);
```

```
            //Ricava il numero di righe interessate dall'eliminazione
            int insertRows = connecting.writer(command);

            //Se avvenuto l'eliminazione nel DB, rimuove la riga dalla tabella
            if (insertRows > 0)
            {
               //Cerca la riga della tabella dalla primary key
               DataRow row = table.Rows.Find(recordMese.id_mese);
               //Elimina la riga dalla tabella
               row.Delete();
               //Accetta il cambiamento
               table.AcceptChanges();
               //Setta il response a true
               result = true;
            }
            else
            {
               //Messaggio di errore
               MessageBox.Show(readerXML.readNodeFromPath("/ListError/Error16") +
                  "/n Verificare i dati di inserimento", "ERRORE",
                  MessageBoxButtons.OK, MessageBoxIcon.Error);
            }

            //Scarica le risorse
            connection.Close();
            command.Dispose();

         }
         catch (Exception ex)
         {
            //Messaggio di errore
            MessageBox.Show(readerXML.readNodeFromPath("/ListError/Error11") + "/n" + ex.Message, "ERRORE",
               MessageBoxButtons.OK, MessageBoxIcon.Error);
         }

         //Ritorna l'esito dell'inserimento
         return result;

      }

   }

}
```

5.2 Esempio XML Writer:

```
private void button6_Click(object sender, EventArgs e)
        {
            //Definisce il percorso del file XML
            string path = @"D:\Documenti\Programmi Visual Studio 2017\Linguaggio
C#\TestApiCollection\TestApiCollection\bin\Debug\XMLErrorList.xml";
            //Istanza alla classe per la lettura
            ReadingXML reading = new ReadingXML(path);
            //Mostra il valore del nodo in un MessageBox letto dal file XML
            MessageBox.Show(reading.readNodeFromPath("/ListError/Error5"));
            //Definisce un valore stringa per sovrescrivere un nodo
            string newVal= "Impossibile leggere il file!!!";
            //Istanza alla classe per la scrittura
            WriterXML wirte = new WriterXML(path);
            //Sovrascrive il nodo
            wirte.reWriteNode("/ListError/Error5", newVal);
            //Mostra il risultato della sovrascrittura in un MessageBox
            MessageBox.Show(reading.readNodeFromPath("/ListError/Error5"));
        }
```