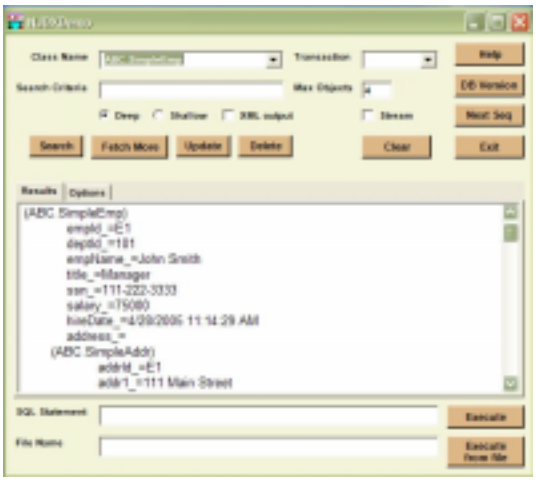
Beyond ADO.NET

ADO.NET is a standard application-programming interface to relational databases defined by Microsoft. Although a useful technology, the use of ADO.NET forces application developers to write complex SQL statements explicitly. This requires a lot of hand coding of SQL statements and processing the results. It is a very mundane, error-prone, and time-consuming job.

NJDX Object-Relational Mapper provides an efficient, intuitive, and object-oriented interface to relational data. NJDX spares the application developers from having to deal with the complexity of working with two different programming paradigms - object-oriented CLR languages (C#, VB.NET, J#) and relational SQL. As the code examples on the next page demonstrate, NJDX boosts developer productivity and reduces maintenance hassles by eliminating endless lines of tedious ADO.NET/OleDB/SQL code.

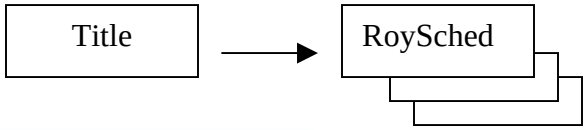
Imagine the productivity gains resulting from such an intuitive programming interface, avoiding impedance mismatch between two programming models, freedom to work with natural domain object models, greatly reduced program size, better performance, ease of maintenance, having tools to generate relational schema from domain class definitions and vice-versa, and seamless integration with the development environment of Visual Studio.NET!



Some of the issues to consider while using raw ADO.NET:

- Generation of SQL statements(SELECT, INSERT, UPDATE and DELETE) for each class
  Do you have to write these statements manually? What if you have hundreds of classes for your application?
- Hard coding of database column names
  Aaargh….!
- What if a new attribute is added to a class or an attribute name changes?
  All corresponding statements have to be updated.
- What if an attribute type changes?
  The getter call has to be changed appropriately. May involve database changes also.
- What if a class hierarchy is involved (e.g., a class PoliticalTitle may inherit from Title and its objects may come from a different table)?
  We have to potentially collect objects from multiple tables. Lots of changes at many levels.
- What if the class structure is more complex (more references, more levels)?
  The code becomes exponentially complex!
- What if we want to do directed queries for a complex object (i.e., follow some references and ignore a few of them etc.)?
  How to specify such a query?
  Do we repeat the hard-coded SQL statements in different parts of the code?
- How easily can this code be maintained / enhanced?
- **Wouldn't you rather be devoting more time to business logic?**

ADO.NET vs. NJDX Example: Assume 2 classes – Title and RoySched. Each Title object has an array of RoySched objects. Primitive attributes of Title objects come from titles table and that of RoySched objects come from roysched table. Using C# language, we are trying to retrieve Title object(s) corresponding to a title_id stored in a String variable 'tid'.



ADO.NET Code

```
// Assuming a Connection object 'conn'
// to the database has been obtained.
// Retrieve a Title object.

// Create a command object
String query="SELECT title_id, type,
   price, title, ytd_sales, pub_id,
   pubdate, royalty, advance, notes FROM
   titles " + "WHERE title_id = '" + tid
   + "'";

SqlCommand cmd = new SqlCommand(query,
   conn);

// Declare the SqlDataReader
SqlDataReader rdr=null;
rdr=cmd.ExecuteReader();

Title title = new Title();
while (rdr.Read()) {
   // Get the results of each column
   title.title_id=(string)rdr["title_id"
      ];
   title.type=(string)rdr["type"];
   title.price=(decimal)rdr["price"];
   title.title=(string)rdr["title"];
   title.ytd_sales=int)rdr["ytd_sales"];
   title.pub_id=(string)rdr["pub_id"];
   title.pubdate=System.DateTime)rdr["pu
      bdate"];
   title.royalty=(int)rdr["royalty"];
   title.advance=(decimal)rdr["advance"]
      ;
   title.notes=(string)rdr["notes"];
   break; // Read only first record
}
rdr.Close();

ArrayList royScheds=new ArrayList();
RoySched rSch;

query="SELECT title_id, lorange, hirange,
   royalty FROM roysched " +
   " WHERE title_id = '" +
   title.title_id + "'" + "order by
   royalty";

// Open another Sqldatareader for
// the roysched table SqlDataReader

rdr1=null;
SqlCommand cmd1=new SqlCommand(query,
   conn);

rdr1 = cmd1.ExecuteReader();

while(rdr1.Read()) {
   rSch=new RoySched();
   rSch.title_id=(string)rdr1["title_id"];
   rSch.lorange=(int)rdr1["lorange"];
   rSch.hirange=(int)rdr1["hirange"];
   rSch.royalty=(int)rdr1["royalty"];
   royScheds.Add(rSch);
}
rdr1.Close();

title.royscheds=new
RoySched[royScheds.Count];

royScheds.CopyTo(title.royscheds);
```

NJDX Code

```
// Assuming a handle 'njdx1' to the
// NJDX service object has been
// obtained.
// Retrieve the Title object(s).
// In general, many qualifying objects
// may be returned.

System.Collections.ArrayList
     queryResults=njdx1.query("Title",
     "title_id = '" + tid + "'", -1,
     0, null);
```

By using NJDX, similar efficiencies are also gained for inserting, updating, or deleting objects.

www.softwaretree.com