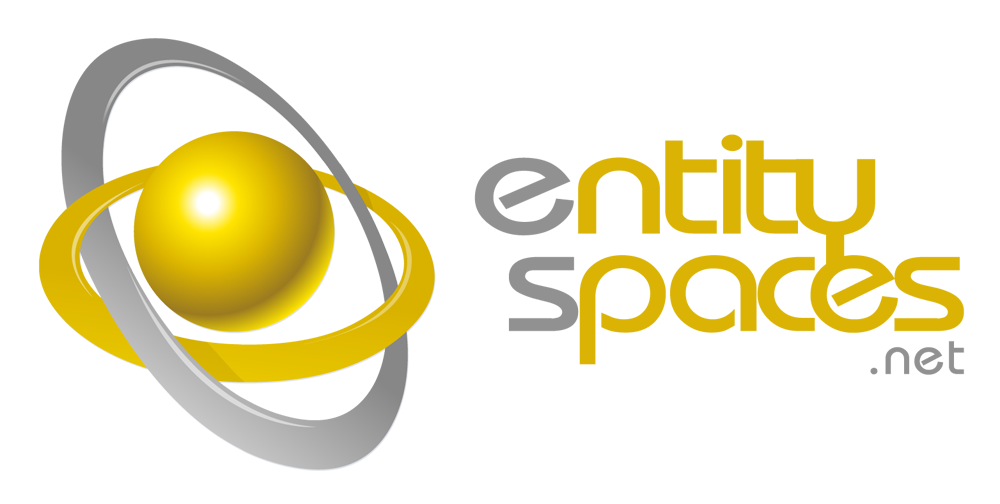
Metadata

Using EntitySpaces MEtadata

February 18, 2012  
EntitySpaces, LLC  
The EntitySpaces Team

**[](http://www.entityspaces.net/)**

Persistence Layer and Business Objects for Microsoft .NET

Contents

[Metadata 1](#_Toc317357563)

[Column Names and Property Names 1](#_Toc317357564)

[Column Metadata 2](#_Toc317357565)

[Iterating over Column Metadata 2](#_Toc317357566)

[Iterating Over Primary Keys 2](#_Toc317357567)

[Searching Column Metadata 3](#_Toc317357568)

[Provider Specific Metadata 3](#_Toc317357569)

# Metadata

EntitySpaces provides on-board metadata that can be used in your application if needed. EntitySpaces also uses it, for instance, when it is generating dynamic SQL for an update statement. The metadata class is the last class in your “generated” class. If your class is named is “Employees” then your metadata class will be named “EmployeesMetadata”. The metadata class is available as a singleton, that is, there is only one copy in memory and you never create it yourself.

The following sample code will demonstrate some of the features of the metadata class.

## Column Names and Property Names

During the code generation process EntitySpaces Studio will turn your table or view column names “ColumnNames” into nice readable property names “PropertyNames”. We strongly recommend that you to never use hard coded strings to access properties or columns. There are only rare times that you would even need to do this and in fact, you might not ever need to, but if you do, use the names as shown below. The advantage of using the Metadata class for this information is that if you were to drop a column in your table or view and regenerate your code you will get a compiler error versus a runtime crash if you were to just use “MyColumnName” as a string in your code. Instead, use the names below.

public class **ColumnNames**

{

public const string EmployeeID = "EmployeeID";

public const string LastName = "LastName";

public const string FirstName = "FirstName";

// more

}

public class **PropertyNames**

{

public const string EmployeeID = "EmployeeID";

public const string LastName = "LastName";

public const string FirstName = "FirstName";

// more ...

}

This is the way you would access these.

string colName = EmployeesMetadata.ColumnNames.FirstName;

string propName = EmployeesMetadata.PropertyNames.FirstName;

## Column Metadata

Each entity type has an esColumnMetadataCollection which contains one esColumnMetadata objects for each column in your table or view. There is a protected constructor in the metadata class that is called for you. An example is shown below.

protected EmployeesMetadata()

{

m\_columns = new esColumnMetadataCollection();

esColumnMetadata c;

c = new esColumnMetadata(EmployeesMetadata.ColumnNames.EmployeeID, 0,   
 typeof(System.Int32), esSystemType.Int32);

c.PropertyName = EmployeesMetadata.PropertyNames.EmployeeID;

c.IsInPrimaryKey = true;

c.IsAutoIncrement = true;

c.NumericPrecision = 10;

m\_columns.Add(c);

c = new esColumnMetadata(EmployeesMetadata.ColumnNames.LastName, 1,   
 typeof(System.String), esSystemType.String);

c.PropertyName = EmployeesMetadata.PropertyNames.LastName;

c.CharacterMaxLength = 20;

m\_columns.Add(c);

// more columns ...

}

Notice there is information describing the properties of the column. You can access this data in your code if you need to like this.

### Iterating over Column Metadata

Employees employee = new Employees();

foreach(esColumnMetadata column in employee.es.Meta.Columns)

{

bool isNullable = column.IsNullable;

}

### Iterating Over Primary Keys

The primary keys are included in the “Iterating over Column Metadata” above and can be identified by looking at the IsInPrimaryKey property. However, if you are only interested in the primary keys you can iterate this way.

foreach(esColumnMetadata column in employee.es.Meta.Columns.**PrimaryKeys**)

{

Console.WriteLine(column.Name);

}

### Searching Column Metadata

You can also search for column metadata like this. If the column or property are not found, the null is returned (an error is not thrown). You would rarely need to use this code but it is here if you need it.

Employees employee = new Employees();

// By Column Name (the name in the database)

esColumnMetadata col = employee.es.Meta.Columns.**FindByColumnName**("SomeColumn");

// By Property Name

esColumnMetadata col = employee.es.Meta.Columns.**FindByPropertyName**("SomeProperty");

## Provider Specific Metadata

Because EntitySpaces operates in a database independent fashion there is certain data that is provider specific (SqlClient vs OracleClient for instance). For example, you might have your database schema in Oracle and Microsoft SQL Server. You application can then run against both databases with no more than a database connection string change required. This is one of the most powerful features of EntitySpaces.

There are certain properties that might be different between those schemas such as the stored procedures names. For instance, Oracle has a very small name length for stored procedures whereas SQL Server does not. Also, the catalog and schema names may differ, and the native database column types might be different such as Oracle’s NVARCHAR2 and Microsoft SQL’s NVARCHAR but still map to a string property. EntitySpaces deals with all this just fine.

Most developers would never need access down to this level of Metadata but it is available as shown below:

Employees employee = new Employees();

esProviderSpecificMetadata meta = employee.es.Meta.GetProviderMetadata("default");

string insertProc = meta.spInsert;

string deleteProc = meta.spUpdate;

string updateProc = meta.spUpdate;

string catalog = meta.Catalog;

string schema = meta.Schema;

esTypeMap map = meta.GetTypeMap(EmployeesMetadata.ColumnNames.FirstName);

string nativeDatabaseType = map.NativeType;

string systemType = map.SystemType;