# Executing stored procedures in ADO.NET

Before all else, make sure to add the namespaces you'll need:

```
using System.Data;
using System.Data.SQL;
```

When executing anything in a database, you'll need to create a connection object. This goes for simple **stored procedures which return recordsets**. Remember the connection strings in ADO.NET are different than in ADO.

```
// create a connection object
SqlDbConnection con = new SqlDbConnection("connection string");

// actually connect to the database
con.Open();

// create command object, passing reference to connection
SqlDbCommand cmd = new SqlDbCommand
    ("prc_MyStoredProc", con);

// required for stored procs, otherwise you get a funky error
cmd.Type = CommandType.StoredProcedure;

// add parameters to command object
cmd.Parameters.Add(new SqlDbParameter("@Param1", "value1"));
cmd.Parameters.Add(new SqlDbParameter("@Param2", "value2"));

// run the stored procedure and grab a reader (somewhat
// comparable to ADO's "recordset" object)
SqlDbReader reader = cmd.ExecuteQuery();

// an initial Read() must be done to start getting data;
// this function returns false if there are no more records
if (!reader.Read())
    throw new SystemException("No records returned");

do
{
    // access columns through 'reader[ColumnName]' which returns
    // an object-type you can try to cast or parse

    string s = reader["VarCharColumn"].ToString();

    // When casting you need to realize it could throw an
    // exception if the database type is not the same as what
    // you are trying to cast to
    DateTime dt = (DateTime)reader["DateColumn"];

    // I like to use int.Parse() on numeric columns, because
    // casting will throw an exception if the native type
    // (int in this case) is not exactly the same as the database
    // type.
    int i = int.Parse(reader["IntColumn"].ToString());

} while (!reader.Read());
```

```
// close reader BEFORE running any more stored procs
reader.Close();

// close database connection when finished running ALL procs
con.Close();
```

To execute stored procedures without any output whatsoever (no parameters, no records) the code is much simpler since the method used returns no reader that needs to be closed.

```
cmd.ExecuteNonQuery();
```

For **stored procedures with output parameters** you will have to define the types of each output parameter along with their name.

```
// define the output parameter(s)
cmd.Parameters.Add(new SqlDbParameter("@OutParam1", ParamType.VarChar,
ParamDirection.Output));

// a proc which has only output parameters is a non-query
cmd.ExecuteNonQuery();

// the parameter will be null if it hasn't been filled
if (null == cmd.Parameters["@OutParam1"])
    throw new SystemException("@OutParam1 was not returned");

// grab the value similar to how you would with a reader
string s = cmd.Parameters["@OutParam1"].ToString();
```

If **the stored procedure returns records and output parameters**, you must read *all* records *before* accessing the output parameters.  Otherwise it will appear as though no output parameters were set.

```
// a proc which returns records is considered a query
SqlDbReader reader = cmd.ExecuteQuery();

// read all records first
while (!reader.Read())
{
    // process record
}

// now you can access output parameters
string s = cmd.Parameters["@OutParam"].ToString();
```

If you're not accessing a Microsoft SQL Database, or if you simply want your code to be portable between database vendors, you can use the OLE-based classes instead of the SQL-specific ones.  These are for generic database interaction and work with any database supporting ODBC.

```
using System.Data;
using System.Data.Ole;
```

Going back to the first example, here's how you'd execute a simple stored procedure in ADO.NET, one which takes parameters and returns a recordset.

```
OleDbConnection con = new OleDbConnection("connection string");
con.Open();
```

```
OleDbCommand cmd = new OleDbCommand
    ("prc_MyStoredProc", con);
cmd.Type = CommandType.StoredProcedure;

cmd.Parameters.Add(new OleDbParameter("@Param1", "value1"));
cmd.Parameters.Add(new OleDbParameter("@Param2", "value2"));

OleDbReader reader = cmd.ExecuteQuery();

if (!reader.Read())
    throw new SystemException("No records returned");

do
{
    string s = reader["VarCharColumn"].ToString();
    DateTime dt = (DateTime)reader["DateColumn"];
    int i = int.Parse(reader["IntColumn"].ToString());
} while (!reader.Read());

reader.Close();

con.Close();
```

Common pitfalls:

- Forgetting to set the Type of a command object to StoredProcedure.
- Not closing a reader before executing another procedure with the same connection object.

Good practices:

- Use try/finally to insure readers are always closed.

Answered questions:

- How do I run a stored procedure with C#?
- How do I use T-SQL stored procedures with ADO.NET?

- Neil C. Obremski (June 1st, 2006)