**The X# SqlRDD**

The X# SQL RDD was created to help fill the void that is created because SAP no longer develops the Advantage Database Server (ADS).

ADS is/was used by many of our clients because it offers a reliable and fast way to access data in a networked environment. Data stored in traditional DBF files and customized ADS specific data files. This data can be accessed both in Index/Sequential (Record oriented) mode and with SQL syntax in set-oriented mode.

As with all our development we try to avoid reinventing the wheel, so we did not want to add a SQL engine on top of the existing RDDs that we created to access DBF based data. Instead, we have chosen to build record-oriented data access on top of existing SQL engines, such as SQL Server, Oracle, MySql, Postgress etc.

**Database independency**

X# is compiling applications for .Net. And .Net already has a great data access layer in the form of Ado.Net. Ado.Net already provides several facilities to work with a common syntax with various databases. There are common classes for connections, commands and resultsets and each database vendor can create its own database specific implementation of these objects.

We have chosen to make the SQL RDD database agnostic. It should be able to work with any database for which an Ado.Net Database provider exists. The RDD works against the common layer that is supported by all DBMS-es.

The implementation of the Ado.Net database providers works great for many things, but unfortunately it lacks support to handle the differences in the syntax for Data Definition Language (DDL) statements, such as “CREATE TABLE” and “DROP INDEX” and also the syntax to retrieve a certain number of rows from a result set is different between providers. Some providers have a “SELECT TOP <n>” syntax, where others use a LIMIT clause “LIMIT <n>” to limit the number of rows in the result set. Also, the syntax to retrieve the value of the last inserted “autonumber” column differs, as well as the available datatypes for the database.

To solve that problem, we have introduced an abstract SqlDbProvider class in the SQLRDD (in the XSharp.RDD.SqlRDD.Providers namespace) that needs to be implemented, to work with a specific version of a SQL database. This class contains information about the assemblyname and class name of the DbProviderFactory class implemented for that database. The DbProviderFactory class is used to create DBMS specific connection, command, parameter classes etc. The SqlDbProvider class inside the SQLRDD also has properties that declare the syntax for the various DDL statements and has a method that can translate a DBF based column definition into the proper syntax for the target database.

The SQLRDD comes with implementations of the SqlDbProvider class for SqlServer, MySql, ODBC, OleDB and Oracle (using the Microsoft Dataprovider for Oracle).

Custom implementations can be created by inheriting from one of the built-in providers, or by directly inheriting from SqlDbProvider.

Please see the class documentation for the SQLRDD for the properties and methods in the SqlDbProvider class.

**Record Oriented vs Set oriented.**

Since the SQLRDD uses SQL databases, it is not really complicated to support retrieving data with a SQL statement. However, emulating Record Oriented data access is a bit more complicated. There are several decisions to be made when for example you would open a customers table with the SQL RDD like this:

USE Customers

Some of the things we need to consider:

* Retrieve all the rows from the database, or limit the # of rows (do we really want to retrieve a million rows when opening a table, when you’re just interested in a particular customer)
* Retrieve all the columns or limit the # of columns to the ones you want to access.
* DBF files have the concept of a record number. SQL engines do not have that. We can emulate a record number, based on the ordinal position of a row in the result set, but that means that changing the order of the rows will result in different record numbers or a row after changing the order of the result set.
* In DBF files rows are not directly deleted but marked for deletion. You can also undo that by calling RECALL. In SQL environments deleted rows are gone immediately
* Column names in DBF files are limited to 10 characters only. SQL does not have that limitation
* Column names in DBF files may be reserved words in the SQL database. How to handle that?
* If you retrieve a character column from a DBF file, it will be always padded with spaces to its maximum length. Most SQL databases do not pad the column values but return “trimmed” values.
* Which indexes are available?
* Which index tags are available for each index and what are the Xbase expressions for these orders?

To allow you to emulate record-oriented data access as best as possible, we have created a mechanism (a so called MetadataProvider) that your application can use to communicate with the RDD and tell the RDD what to do for a particular table. Maybe you want to limit the # of rows or columns for one relatively small table, but not for a large table with transactions.

The SQLRDD contains 3 built-in mechanisms to provider this metadata (again in the XSharp.RDD.SqlRDD.Providers namespace)

* IniMetaDataProvider, which gets its information from an ini-file
* CallBackMetadataProvider, which gets its information from a call back function in code.
* DatabaseMetadataProvider, which gets its information from tables inside the SQL database.

Further in this document we will describe the 3 providers and how they work.  
And of course you can inherit from one of these 3 providers or create your own provider that implements the IMetadataProvider interface.

**Working with indexes**

For a DBF table you can create indexes. These can be stored as files with a single sort order (NTX) or as files with multiple sort orders (CDX). The individual orders inside a CDX are often referred to as TAGS. Indexes consist of an index expression, which can be an individual column name ( “CustomerNo”) but also as a combination of several fields in an Xbase expression (“CustomerNo+DTOS(OrderDate)”). Indexes can also contain a For condition that makes sure that only records where that condition evaluates to TRUE become part of the index.

These index expressions do not always translate directly into SQL. That is why the RDD parses the index expressions into individual elements and tries to translate them into SQL expressions.   
Also SQL databases normally do not