STSdb 4.0

GETTING STARTED

4.0.7 API

STSdb 4.0 is a NoSQL open-source embedded database. The database engine is based on WaterfallTree™ patented technology for maximum random/sequential keys indexing performance.

STSdb 4.0 is designed for ease of use. The database and tables are used like simple data structures. Each database (storage engine instance) can handle many tables. Each table is a sorted key/value store. In each table the user can work with its own types or can work with the internal database type.

Simple Example

1. Open the database and create a simple table:

```
    using (IStorageEngine engine = STSdb.FromFile("test.stsdb4"))
    {
        var table = engine.OpenXTable<int, Tick>("table");

        for (int i = 0; i < 1000000; i++)
        {
            table[i] = new Tick();
        }

        engine.Commit();
    }
```

2. Read from the table:

```
    using (IStorageEngine engine = STSdb.FromFile("test.stsdb4"))
    {
        var table = engine.OpenXTable<int, Tick>("table");

        foreach (var row in table) //table.Forward(), table.Backward()
        {
            Console.WriteLine("{0} {1}", row.Key, row.Value);
        }
    }
```

The type Tick has the following structure:

```
public class Tick
{
    public string Symbol { get; set; }
    public DateTime Timestamp { get; set; }
    public double Bid { get; set; }
    public double Ask { get; set; }
    public int BidSize { get; set; }
    public int AskSize { get; set; }
    public string Provider { get; set; }

    public Tick()
    {
    }
}
```

Supported types

Supported types for TKey are:

1. Primitive STSdb types – `Boolean`, `Char`, `SByte`, `Byte`, `Int16`, `UInt16`, `Int32`, `UInt32`, `Int64`, `UInt64`, `Single`, `Double`, `Decimal`, `DateTime`, `TimeSpan`, `String`, `byte[]`;
2. Enums;
3. Guid;
4. Classes (with public default constructor) and structures, containing public read/write properties or fields with types from [1-3];

Supported types for TRecord are:

1. Primitive STSdb types – `Boolean`, `Char`, `SByte`, `Byte`, `Int16`, `UInt16`, `Int32`, `UInt32`, `Int64`, `UInt64`, `Single`, `Double`, `Decimal`, `DateTime`, `TimeSpan`, `String`, `byte[]`;
2. Enums;
3. Guid;
4. T[], List<T>, KeyValuePair<K, V> , Dictionary<K, V> and Nullable<T>, where T, K and V are types from [1-4];
5. Classes (with public default constructor) and structures, containing public read/write properties or fields with types from [1-5].

For example, if we have the following two types:

```
public class Key
{
    public string Symbol { get; set; }
    public DateTime Timestamp { get; set; }
}

public class Tick
{
    public double Bid { get; set; }
    public double Ask { get; set; }
    public int BidSize { get; set; }
    public int AskSize { get; set; }
    public string Provider { get; set; }
}
```

We can easily open a table with a composite key in the following way:

```
    var table2 = engine.OpenXTable <Key, Tick>("table2");
```

Or even a more complicated example:

```
public class Key
{
    public string Symbol { get; set; }
    public DateTime Timestamp { get; set; }
}

public class Provider
{
    public string Name { get; set; }
    public string Website { get; set; }
}

public class Tick
{
    public double Bid { get; set; }
    public double Ask { get; set; }
    public int BidSize { get; set; }
    public int AskSize { get; set; }
    public Provider Provider { get; set; }
}
```

```
    var table3 = engine.OpenXTable<Key, Tick>("table3");
```

NOTE: For precise definition of the STSdb 4.0 supported types, see the Developer's guide document.

Scheme capabilities

The STSdb 4.0 storage engine provides scheme capabilities. This gives the users the possibility to obtain meta-information for the tables and the database itself. Every table has its own IDescriptor which stores meta-information about the table.

The usage of the scheme is shown in the following examples.

If we have a table:

```
    var table = engine.OpenXTable<int, Tick>("table");
```

Obtaining a table's IDescriptor can be done in the following way:

1. Via the *this* property of the storage engine instance:

```
    IDescriptor descriptor = engine["table"];
```

The engine also provides the following scheme methods and properties:

2. Delete a table

```
    engine.Delete("table");
```

3. Rename a table

```
    engine.Rename("table", "table_NewName");
```

4. Check if a table exists:

```
bool exists = engine.Exists("table");
```

5. Obtain the number of tables & virtual files the storage engine holds:

```
int tablesCount = engine.Count;
```

Thread-safe

STSdb 4.0 engine is thread-safe. XTable-s and XFile-s are also thread-safe.

Transactions

STSdb 4.0 supports atomic commit at storage engine level – engine.Commit() commits all changes in all opened tables.

NOTE: For further reading and more detailed examples check out the STSdb 4.0 Developer's Guide document.