

Archived:Using Symbian SQL API with data streams

 Archived: This article is **archived** because it is not considered relevant for third-party developers creating commercial solutions today. If you think this article is still relevant, let us know by adding the template `{{ReviewForRemovalFromArchive|user=~~~~|write your reason here}}`.

Overview

This snippet shows how to use two classes that support the streaming functionality with the SQL API. Data streams can be used to write to a database by declaring an `RSqlParamWriteStream` object and call one of its `Bind()` methods, for example, `BindText()`. Reading data from a database to a data stream is possible by declaring an `RSqlColumnReadStream` object and call one of its `Column()` methods, such as `columnBinary()`.

This snippet can be self-signed.

MMP file

The following libraries are required:

```
LIBRARY euser.lib
LIBRARY sqldb.lib
LIBRARY estor.lib //RSqlColumnReadStream, RSqlParamWriteStream
```

Source file

```
#include <e32base.h>
#include <SqlDb.h>

void ReadAndWriteStreamsL()
{
    _LIT(KStreamsDbName, "\\streams.db");

    RSqlDatabase database;
    TInt error = database.Create(KStreamsDbName);

    // == Writing from a data stream ==

    if(error == KErrNone)
    {
        CleanupClosePushL(database);

        // Create a table
        _LIT(KSqlCreateTableStreams, "CREATE TABLE STREAMS(ID INTEGER, STREAM1 TEXT,
STREAM2 BLOB);");
        User::LeaveIfError(database.Exec(KSqlCreateTableStreams));

        _LIT(KSqlInsertIntoTableStreams, "INSERT INTO STREAMS(ID, STREAM1, STREAM2)
VALUES(:Val1, :Val2, :Val3)");
```

```
RSqlStatement sqlInsertIntoStreamsStatement;
sqlInsertIntoStreamsStatement.Prepare(database, KSqlInsertIntoTableStreams);
CleanupClosePushL(sqlInsertIntoStreamsStatement);

TInt paramIndex1 = sqlInsertIntoStreamsStatement.ParameterIndex(_L(":Val1"));
TInt paramIndex2 = sqlInsertIntoStreamsStatement.ParameterIndex(_L(":Val2"));
TInt paramIndex3 = sqlInsertIntoStreamsStatement.ParameterIndex(_L(":Val3"));

for(TInt id=1; id<10; id++)
{
    //Set value into column ID
    User::LeaveIfError(sqlInsertIntoStreamsStatement.BindInt(paramIndex1, id));

    RSqlParamWriteStream stream1;
    RSqlParamWriteStream stream2;
    CleanupClosePushL(stream1);
    CleanupClosePushL(stream2);

    User::LeaveIfError(stream1.BindText(sqlInsertIntoStreamsStatement,
paramIndex2));

    //Generate some text data into column STREAM2
    TChar ch = '0' + id;
    TBuf<500> idBuf;
    idBuf.Fill(ch,500);
    stream1.WriteL(idBuf);

    User::LeaveIfError(stream2.BindBinary(sqlInsertIntoStreamsStatement,
paramIndex3));

    //Generate some binary data into column STREAM2
    TUint8 startValue = '0' + id;
    TUint8 endValue = startValue + 100;

    for(; startValue<endValue; startValue++)
    {
        stream2 << static_cast <TUint8> (startValue);
    }
    stream2.CommitL();

    User::LeaveIfError(sqlInsertIntoStreamsStatement.Exec());
    User::LeaveIfError(sqlInsertIntoStreamsStatement.Reset());

    CleanupStack::PopAndDestroy(2); //stream2, stream1
}

CleanupStack::PopAndDestroy(2); //sqlInsertIntoStreamsStatement, database
}
else
{
    //open database failed
}

// == Reading to a data stream ==
error = database.Open(KStreamsDbName);

if (error == KErrNone)
```

```

{
CleanupClosePushL(database);

RSqlStatement sqlSelectFromStreamsTableStatement;

_LIT(KSqlSelectFromStreamsTable, "SELECT * FROM STREAMS");
TInt ret = sqlSelectFromStreamsTableStatement.Prepare(database,
KSqlSelectFromStreamsTable);

if(ret == KErrNone)
{
CleanupClosePushL(sqlSelectFromStreamsTableStatement);

TInt columnIndex1 = sqlSelectFromStreamsTableStatement.ColumnIndex(_L("STREAM1"));
TInt columnIndex2 = sqlSelectFromStreamsTableStatement.ColumnIndex(_L("STREAM2"));

TInt err = KErrNone;

while((err = sqlSelectFromStreamsTableStatement.Next()) == KSqlAtRow)
{
RSqlColumnReadStream stream1;
RSqlColumnReadStream stream2;
CleanupClosePushL(stream1);
CleanupClosePushL(stream2);

//Get data from text column STREAM1
User::LeaveIfError(stream1.ColumnText(sqlSelectFromStreamsTableStatement,
columnIndex1));
TInt size1 = sqlSelectFromStreamsTableStatement.ColumnSize(columnIndex1);
RBuf buf;
buf.CreateL(size1);
CleanupClosePushL(buf);

stream1.ReadL(buf, size1);
//do something with the data...

//Get data from binary column STREAM2
User::LeaveIfError(stream2.ColumnBinary(sqlSelectFromStreamsTableStatement,
columnIndex2));
TInt size2 = sqlSelectFromStreamsTableStatement.ColumnSize(columnIndex2);
HBufC8* buf2 = HBufC8::NewLC(size2);
TPtr8 buf2Ptr = buf2->Des();

stream2.ReadL(buf2Ptr, size2);
//do something with the data...
CleanupStack::PopAndDestroy(4); //buf2, buf, stream2, stream1
}
if(err == KSqlAtEnd)
{
// OK - no more records
}
else
{
// process the error
}
}

```

```
CleanupStack::PopAndDestroy(2); //sqlSelectFromStreamsTableStatement, database
    }
else
    {
    //open database failed
    }
}
```

Postconditions

The example database streams.db has been created and the test rows have been inserted into tables using RSQLParamWriteStream streams. After that RSQLColumnReadStream streams are used to fetch the test data.

See also

- [Archived:Using SQL API for creating non-secure and secure databases on Symbian](#)
- [Archived:Using Symbian SQL API for attaching and detaching databases](#)
- [Archived:Using Symbian SQL API with SQL statements which do not return data](#)
- [Archived:Using Symbian SQL API with SQL statements which return data](#)
- [Archived:Using Symbian SQL API with scalar queries](#)