



The **CMyDBClass** class illustrates how to use the **RDbNamedDatabase** API with your own application. This example code is made for [S60](#) 3rd Edition but if you want to use it on the older platforms just remove the line that utilizes **RFs**'s *PrivatePath()* method and replace it with a constant path definition.

As shown in the construct method, the database is opened if it exists and created if it doesn't. To change it to accommodate your own data you need to change the table definition in the *CreateTableL()* method. Also remember to change other functions to use the same table definition.

Library required:

```
LIBRARY    efsrv.lib //RFs
LIBRARY    edbms.lib //RDbNamedDatabase, RDbView
```

NamedDataBase.cpp

```
#include "NamedDataBase.h"

CMyDBClass::~CMyDBClass()
{
    iItemsDatabase;
    iFsSession;
}

void CMyDBClass::ConstructL()
{
    ::LeaveIfError(iFsSession.Connect());

    TFileName DBFileName

    //create private path
    ::LeaveIfError(iFsSession.CreatePrivatePath(RFs::GetSystemDrive()));
    // private path with no drive on it
    iFsSession.PrivatePath(DBFileName);

    TFindFile PrivFsSession);
    // find out the drive
    if(KErrNone == PrivFolder.FindByDir(DBFileName, KNullDesc))
    {
        Copy(DBFileName.File());
        Append(RFs::GetSystemDriveName());
    }

    if(BaflUtils::FileExists(iFsSession, DBFileName))
    {
        ::LeaveIfError(iItemsDatabase.Open(iFsSession,
                                           DBFileName));

        (iItemArray) ReadDbItemsL
    }
    else
    {
        // no database exists so we make one
        ::LeaveIfError(iItemsDatabase.Create(iFsSession,
                                           DBFileName));

        // and will create the onlt table needed for it
        (iItemsDatabase) CreateTableL
```

Database_Example

```
}
}
}

void CMyDBClass::CreateTableL(RDbDatabase& aDatabase)
{
// Create a table definition
    CDbColSms=CDbColSet::NewLC();

// Add Columns
    TDbCol0; EDbColInt32);
// automatic indexing for items, it is our key field.
    iAttributes=id.EAutoIncrement;
    ->AddIn($d);

    ->AddIn($DbCol(NCol1, EDbColText,100));
    ->AddIn($DbCol(NCol2, EDbColText,100));

// Create a table
    ::LeaveIfError(aDatabase.CreateTable(KtxtItemList, *columns));

// cleanup the column set
    CleanupSpaceAndDestroy(columns);
}

void CMyDBClass::ReadDbItemsL(RArray<TExampleItem>& aItemArray)
{
    aItemArray; // first reset the array

    TFileName QueryBuffer
// just get all columns & rows
    QueryBuffer("SELECT * FROM ");
    QueryAppend(KtxtItemList);

    RDbView Myview
    Prepare(iItemsDatabase, TDbQuery(QueryBuffer));
    CleanupSpaceAndDestroy(Myview);
    Myview.All();
    FMyview();

// Just delete one instance of the message
while(Myview.AtRow())
{
    GetL(); Myview.

        TExampleItem NewItem
        iIndex =Myview.ColInt(1);
        NewItem.iName.Copy(Myview.ColDes(2));
        NewItem.iValue.Copy(Myview.ColDes(3));

        Append(NewItem);
        NextL(); Myview.
}

    CleanupSpaceAndDestroy(1); // Myview
}

void CMyDBClass::DeleteFromDatabaseL(TInt& aIndex)
{
    TFileName QueryBuffer
    QueryBuffer("SELECT * FROM ");
    QueryAppend(KtxtItemList);
```

Database_Example

```
QueryAppendL(" WHERE ");
QueryAppend(NCol0);
QueryAppendL(" = ");
QueryAppendNum(aIndex);

iItemsDatabase;

RDbView Myview
// query buffer with index finds only the selected item row.
Myview(iItemsDatabase, TDbQuery(QueryBuffer));
CleanupClassPushL;

Myview.All();
Myview();
// we have autoincrement in index so it should be unique
// but just to make sure, we use 'while', instead of 'if'
while(Myview.AtRow())
{
    GetL(); Myview.
    DeleteL()Myview.
    NextL(); Myview.
}

CleanupSpanAndDestroy(1); // Myview
iItemsDatabase;
// compacts the database, by physicaly removig deleted data.
iItemsDatabase();
}

void CMyDBClass::UpdateDatabaseL(const TDesC& aName,
                                const TDesC& aValue,
                                TInt& aIndex)
{
    TFileName QueryBuffer
    QueryBuffer("SELECT * FROM ");
    QueryAppend(KtxtItemList);
    QueryAppendL(" WHERE ");
    QueryAppend(NCol0);
    QueryAppendL(" = ");
    QueryAppendNum(aIndex);

    iItemsDatabase;

    RDbView Myview
    Myview(iItemsDatabase, TDbQuery(QueryBuffer));
    CleanupClassPushL;

    Myview.All();
    Myview();

    if(Myview.AtRow())
    {
        UpdateL()Myview.
        SetColl(2MyName);
        SetColl(3MyValue);
        PutL(); Myview.
    }

    CleanupSpanAndDestroy(1); // Myview
    iItemsDatabase;
}
```

Database_Example

```
void CMyDBClass::SaveToDatabaseL(const TDesC& aName,
                                const TDesC& aValue,
                                TInt& aIndex)
{
    TFileName QueryBuffer
    QueryBuffer = ("SELECT * FROM ");
    QueryBuffer.Append(KtxtItemList);

    iItemsDatabase;

    RDbView Myview
    Myview(iItemsDatabase, TDbQuery(QueryBuffer));
    CleanupClosePushed(Myview);

    Myview();

    SetCol(2, aName);
    SetCol(3, aValue);

    Myview.

    =aIndex.ColInt(1); // autoincrement gives us unique index.

    CleanupStackAndDestroy(1); // Myview
    iItemsDatabase;
}
```

NamedDataBase.h

```
//Headers needed
#include <f32file.h> //RFs
#include <d32dbms.h> //RDbNamedDatabase, RDbView

//database name
_LIT(KtxDatabaseName, "Items.db");

// database column names
_LIT(NCol0, "index");
_LIT(NCol1, "name");
_LIT(NCol2, "value");

// database table name
_LIT(KtxtItemList, "itemlist");

class TExampleItem
{
public:
    TExampleItem(): iIndex(-1){};
public:
    TInt          ;    iIndex
    TBuf<100>     iName, iValue
};

class CMyDBClass : public CBase
{
public :
    void ConstructL();
    ~CMyDBClass();
public:
```

Database_Example

```
void DeleteFromDatabaseL(TInt& aIndex);
void UpdateDatabaseL(const TDesC& aName, const TDesC& aValue, TInt& aIndex);
void SaveToDatabaseL(const TDesC& aName, const TDesC& aValue, TInt& aIndex);
void ReadDbItemsL(RArray<TExampleItem>& aItemArray
private:
    void CreateTableL(RDbDatabase& aDatabase);
private:
    RArray<TExampleItem> iItemArray;
    RDbNamedDatabase iItemsDatabase;
    RFs iFsSession;
};
```