Технически Университет – Варна

Проект
по “Програмиране за бази от данни”

Студент: Иван Радославов Димов, Ф.№:20621603,
Група: 4а, 3к спец. „СИТ“

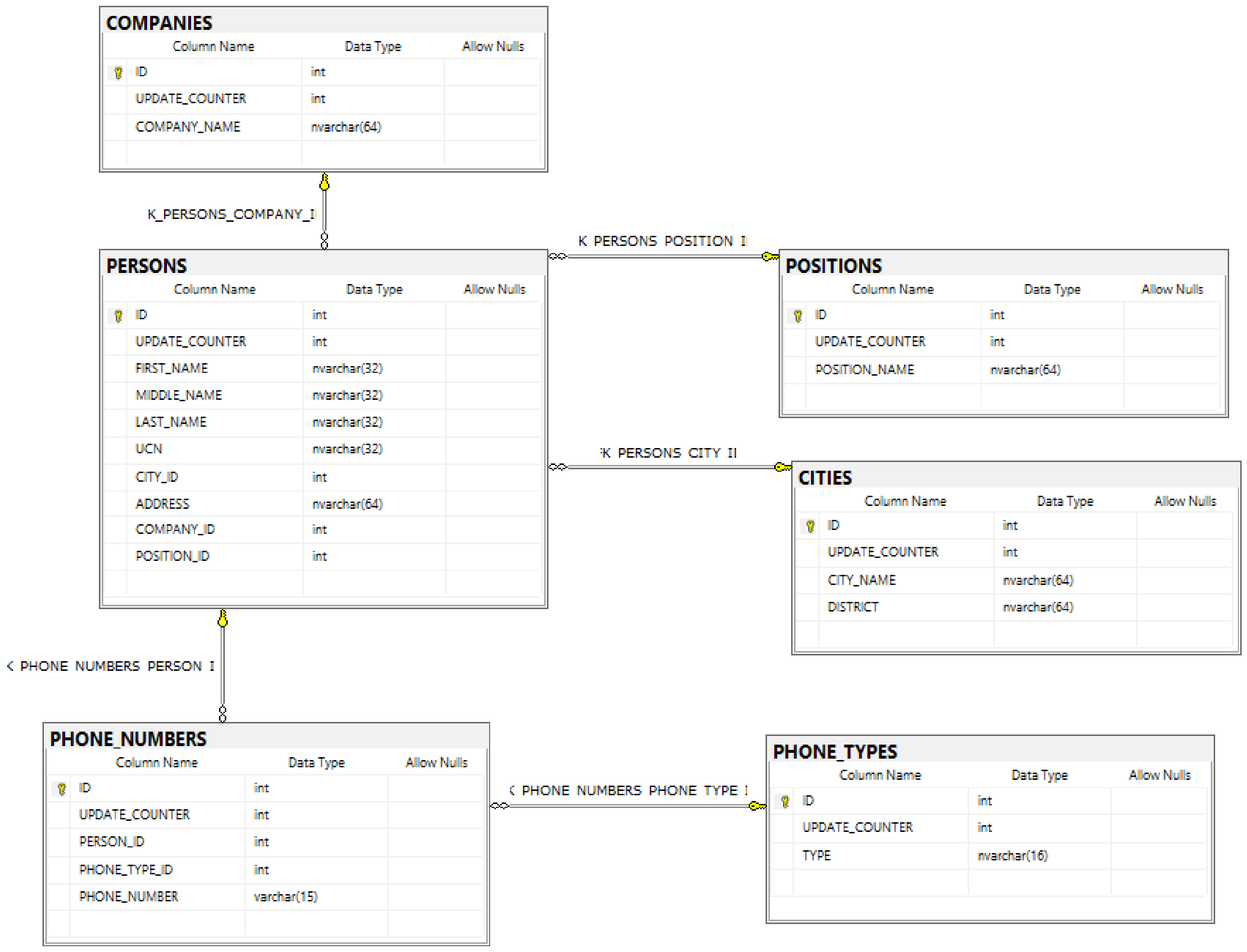
Съдържание

1. Задание

Темата на проекта е система за телефонен указател. Системата трябва системата трябва да предоставя графичен потребителски интерфейс и да реализира възможности за:
- Добавяне/редактиране/ изтриване на информация към таблиците
- Възможности за търсения и справки (мин. 5) по различни критерии
- Печат на справките (Reports)

2. Модел

Използван е следния релационен модел за базата данни:



Състои се от 6 таблици Companies (компании), Persons (хора), Positions (позициите на хората в компаниите), Cities (градове), Phone_numbers (телефонни номера) и Phone_types (видове телефонни номера). Базата данни съдържа и допълнителна таблица за потребителите на системата.

Всяка една от таблиците съдържа полетата ID и Update_counter. ID служи за уникална идентификация на всеки запис (primary key). Полето Update_counter служи като брояч на версията на всеки запис. Той се увеличава всеки път, когато записът се обновява, и когато потребител опитва да обнови записа, системата проверява дали Update_counter в записа съвпада с Update_counter , който потребителят е прочел, когато е първоначално достъпил записа. Ако броячите за обновяване съвпадат, потребителят може да продължи с обновяването, но ако не съвпадат, това означава, че друг потребител е променил записа междувременно, и обновлението на потребителя се отхвърля.

Таблицата Companies съхранява имена на компании с максимална дължина 64 символа. Името на компанията е уникален индекс, което не разрешава дублиране на имената. Реферира се от таблицата Persons.

Таблицата Positions съхранява имената на длъжностите заети от съответния човек. Името е уникален индекс. Максималната дължина е 64 символа.

Таблицата Cities съхранява градовете в които живеят хората. Съдържа поле за името на града(City_name) и областта в която се намира (District). Реферира се от таблицата Persons.

Таблицата Persons съхранява информация за хората, въведени в телефонния указател. Съдържа 3 полета за собствено, бащино и фамилно име, поле ЕГН, което е уникален индекс, адреса и компанията в която работи, както и позицията му в нея (осъществено чрез foreign key).

Таблицата Phone_numbers съхранява телефонните номера и техния тип. Самите телефонни номера са уникален индекс, а типа е foreign key, който реферира таблицата за типа.

Таблицата за типа на телефонния номер съдържа поле за името на типа, което е уникален индекс. В програмата са зададени 3 възможни типа – домашен, мобилен и служебен.

3. Средства за реализация

Приложението за телефонен указател е написано на C++. То използва множество софтуерни технологии като Microsoft MFC (Microsoft Foundation Classes), което е C++ библиотека, която предоставя рамка за създаване на приложения, базирани на Windows. MFC предоставя класове за управление на потребителски интерфейси, контроли, диалози и обработка на съобщения.

Приложението също използва Microsoft SQL Server, система за управление на релационни бази от данни.

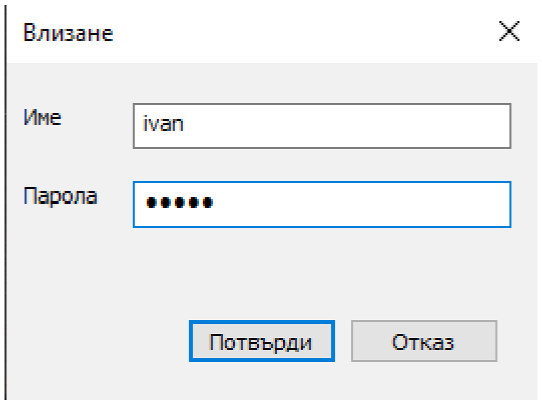
Използваната архитектура е архитектурата Document-View. Тази архитектура разделя логиката за представяне (View) от основните данни и бизнес логика (Document). Това улеснява модифицирането на потребителския интерфейс, без да се засягат основните данни или бизнес логиката, осъществявайки енкапсулация. Архитектурата също използва шаблона "Observer", който позволява на Views да бъдат уведомявани за промените в данните. Когато Document бъде променен, той изпраща известие до всички свои Views, които след това обновяват своите представяния на данните.

В приложението за телефонен указател има система за вход на потребители, която хешира паролите с помощта на алгоритъма SHA256. Това гарантира, че паролите на потребителите не се съхраняват в чист текст, като се защитават данните на потребителите от потенциални нарушения на сигурността. Самият алгоритъм е използван наготово от https://github.com/System-Glitch/SHA256.

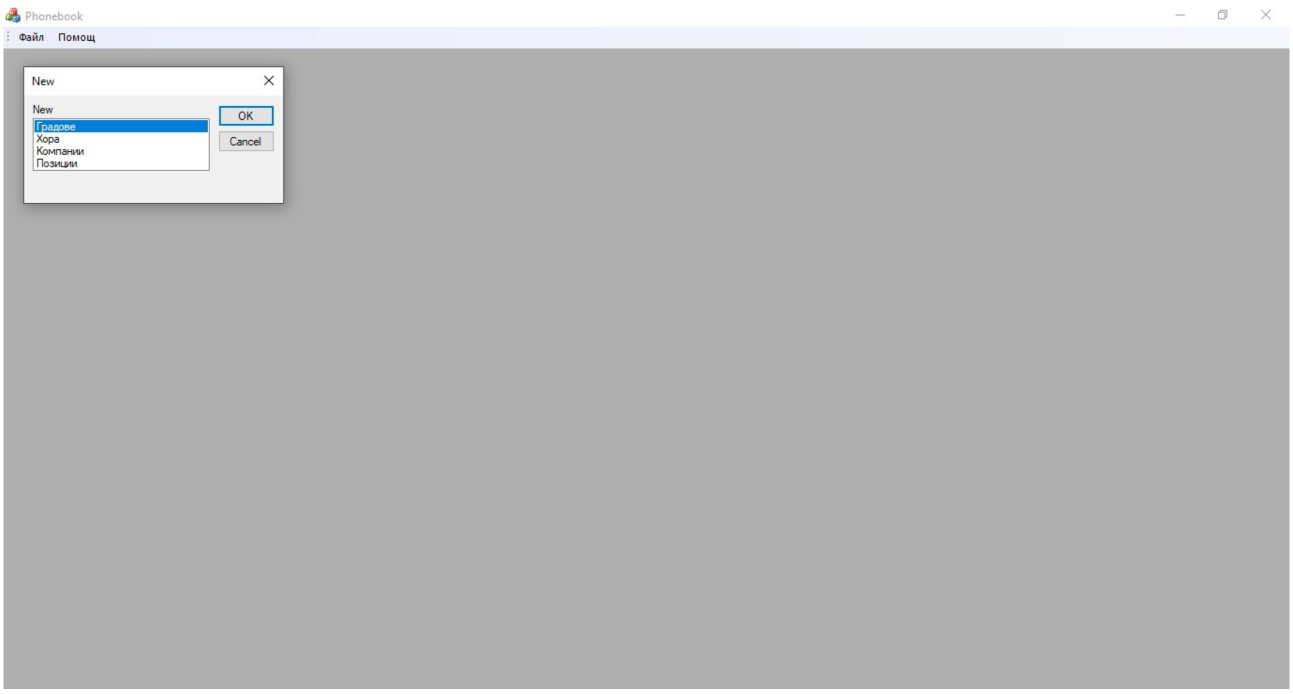
За да се гарантира съгласуваността на данните, приложението използва оптимистичния метод, техника за управление на едновременния достъп до данни в база от данни. Този метод позволява на множество потребители да модифицират данните едновременно, без да блокират един друг, като все още се гарантира поддържането на

4. Ръководство на потребителя

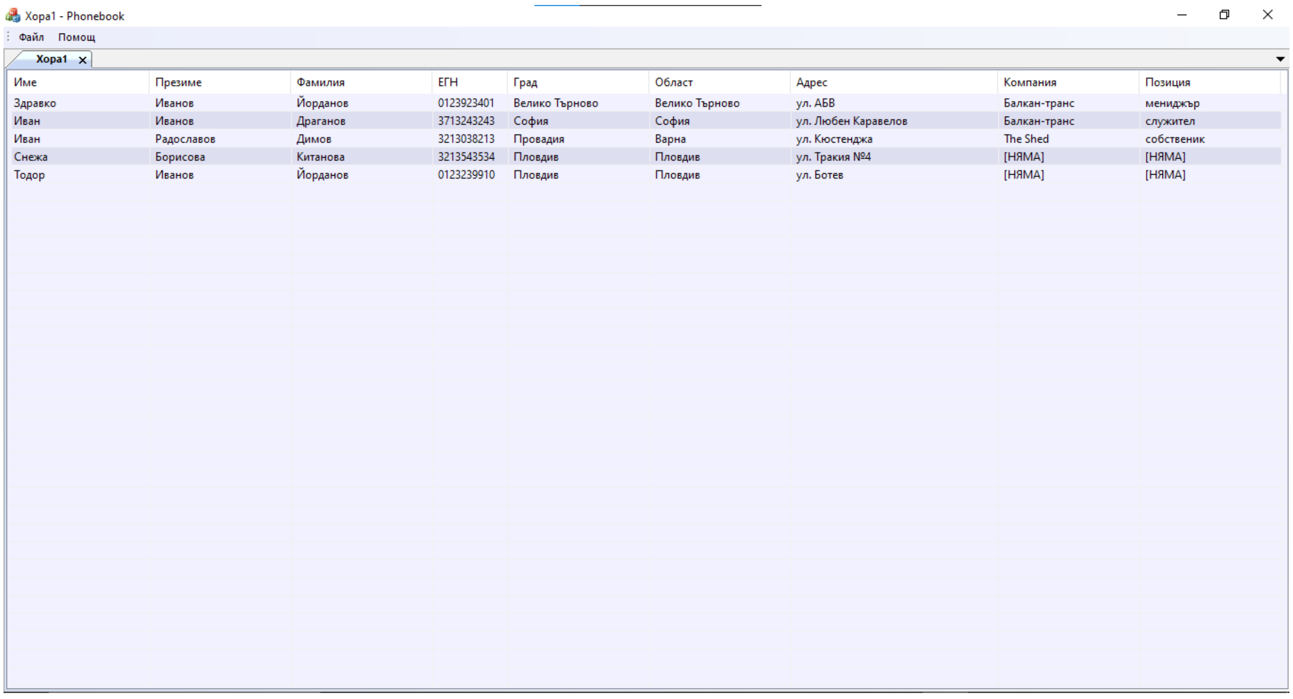
При стартиране на приложението се показва диалогов прозорец за въвеждане на потребителско име и парола. След въвеждане на креденциалите и натискане бутона „Потвърди“ се показва екрана за избор на таблица.



Чрез използване на клавишите със стрелки или мишката се селектира таблица. От бутона „OK“ се показва селектираната таблица.

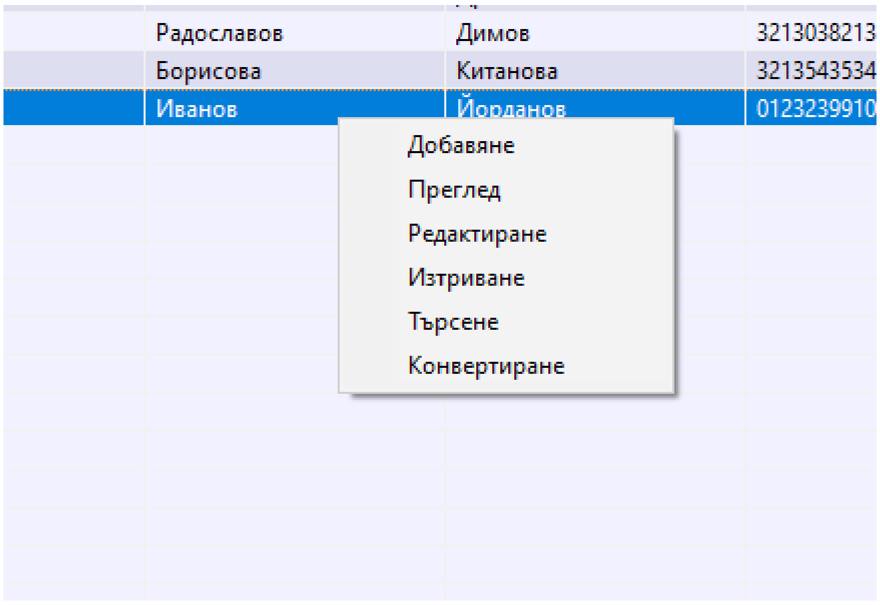


На екрана на селектираната таблица се показват всички записите и всичките полета.

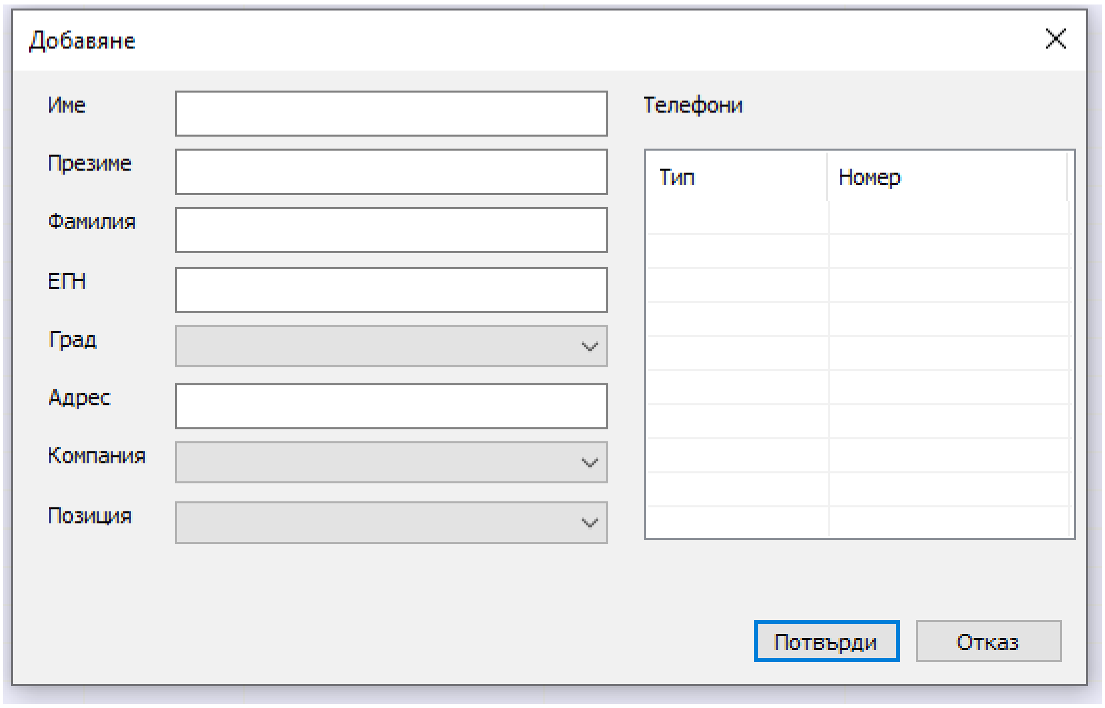


Име	Презиме	Фамилия	ЕГН	Град	Област	Адрес	Компания	Позиция
Здравко	Иванов	Йорданов	0123923401	Велико Търново	Велико Търново	ул. АБВ	Балкан-транс	мениджър
Иван	Иванов	Драганов	3713243243	София	София	ул. Любен Каравелов	Балкан-транс	служител
Иван	Радославов	Димов	3213038213	Провадия	Варна	ул. Кюстенджа	The Shed	собственик
Снежа	Борисова	Китанова	3213543534	Пловдив	Пловдив	ул. Тракия №4	[НЯМА]	[НЯМА]
Тодор	Иванов	Йорданов	0123239910	Пловдив	Пловдив	ул. Ботев	[НЯМА]	[НЯМА]

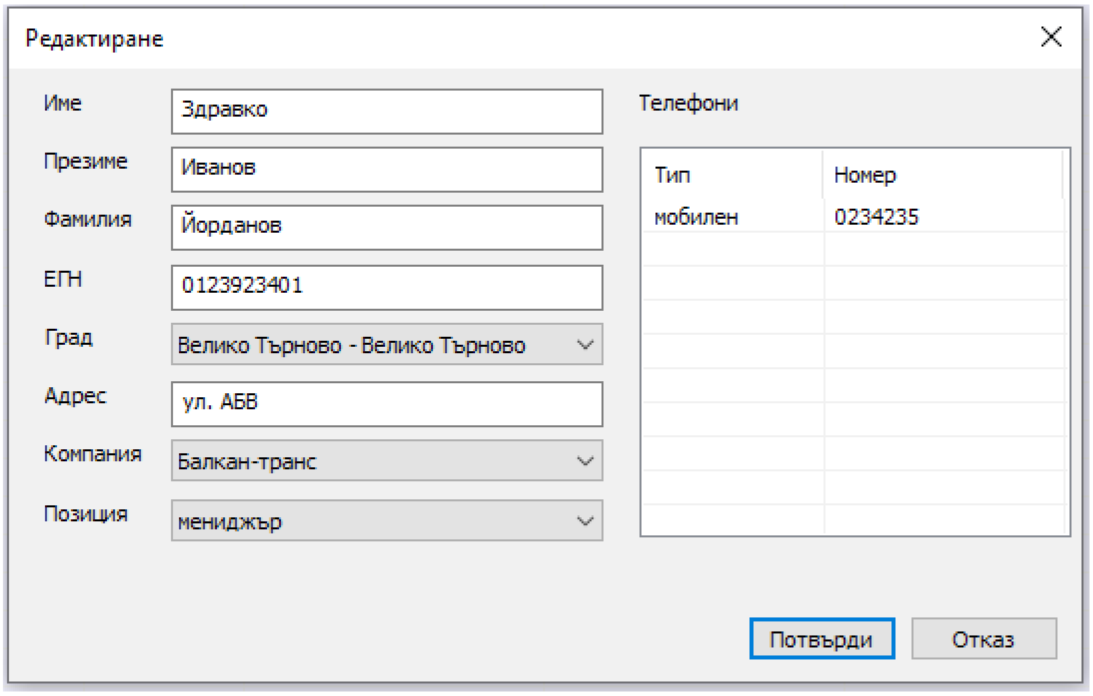
При натискане на десен бутон на мишката върху ред от таблицата се показва контекстно меню. Опциите са „Добавяне“, „Преглед“, „Редактиране“, „Изтриване“, „Търсене“ и „Конвертиране“. Ако не е предварително селектиран ред от таблицата, опциите „Преглед“, „Редактиране“ и „Изтриване“ се засивяват и не могат да бъдат селектирани.



При избор на опцията „Добавяне“ се показва диалога за добавяне на запис. След въвеждане на всички данни в съответните полета и контроли се избира бутона „Потвърди“ за да се добави записа. При натискане на „Отказ“ записа не се добавя. Ако някое поле не е въведено се изписва съобщение за грешка указващо кое поле трябва да се попълни. Допълнително при таблицата за хората може да се добавят телефонни номера чрез десен бутон на мишката, което извежда подобен диалогов прозорец.



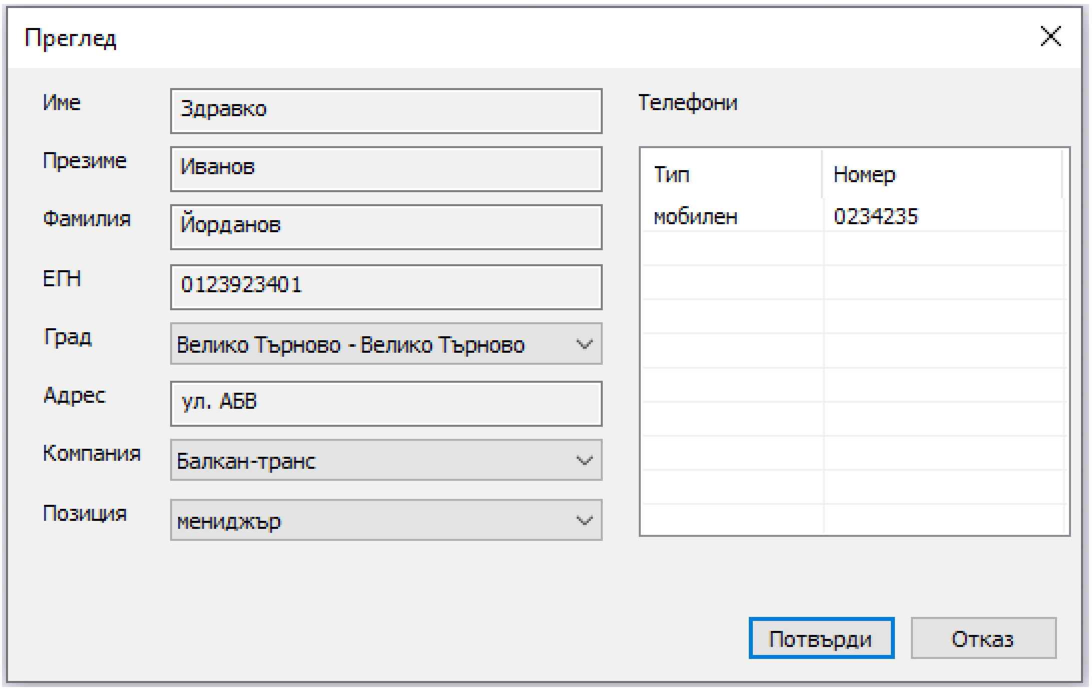
При избор на бутона „Редактиране“ се показва подобен диалог за редактиране на записа. След промяна на стойностите чрез бутона потвърди се запазват данните.



Име	Здравко
Презиме	Иванов
Фамилия	Йорданов
ЕГН	0123923401
Град	Велико Търново - Велико Търново
Адрес	ул. АБВ
Компания	Балкан-транс
Позиция	мениджър

Тип	Номер
мобилен	0234235

Бутона „Преглед“ показва данните само за преглед без опция за редактиране. Диалогът за преглед може също да бъде достъпен чрез двойно натискане на левия бутон на мишката върху съответния запис.

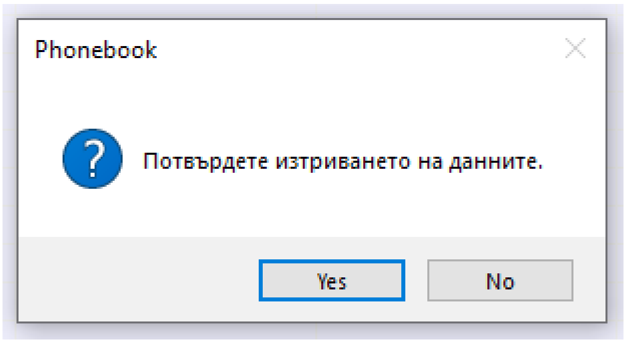

Преглед

Име	Здравко
Презиме	Иванов
Фамилия	Йорданов
ЕГН	0123923401
Град	Велико Търново - Велико Търново
Адрес	ул. АБВ
Компания	Балкан-транс
Позиция	мениджър

Телефони

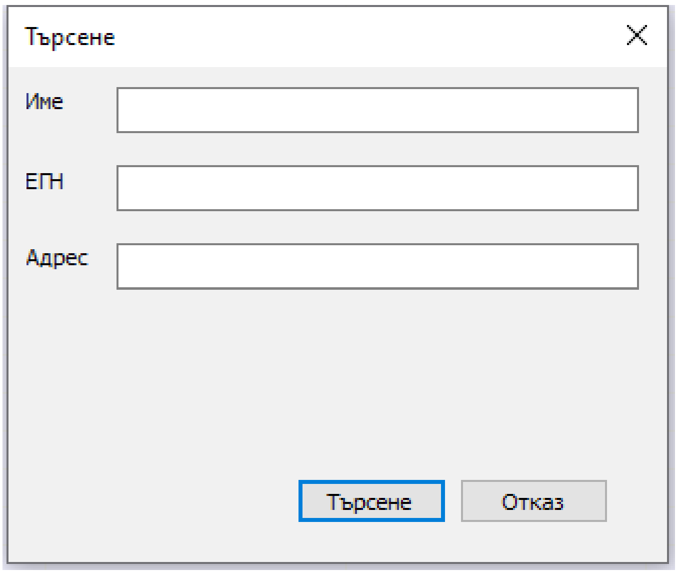
Тип	Номер
мобилен	0234235

Чрез опцията „Изтриване“ можем да премахнем запис от таблицата, като се извежда диалог за потвърждаване.


Phonebook

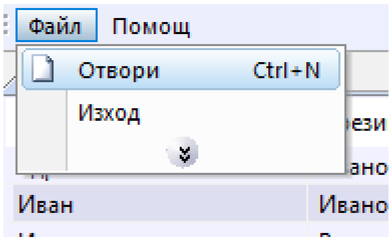
Потвърдете изтриването на данните.

Чрез опцията „Търсене“ се показва диалог за търсене на записи. Въвеждат се търсените стойности във съответните полета и се избира бутона „Търсене“. Всички непопълнени полета се игнорират. На екрана на таблицата се извеждат записите отговарящи на търсените стойности. От опцията „Отказ“ премахваме критериите за търсене и се показват всички записи.



От опцията „Конвертиране“ можем да запазим търсените или всички стойности в HTML формат. След което може да се разпечатат данните чрез принтер на хартиен носител.

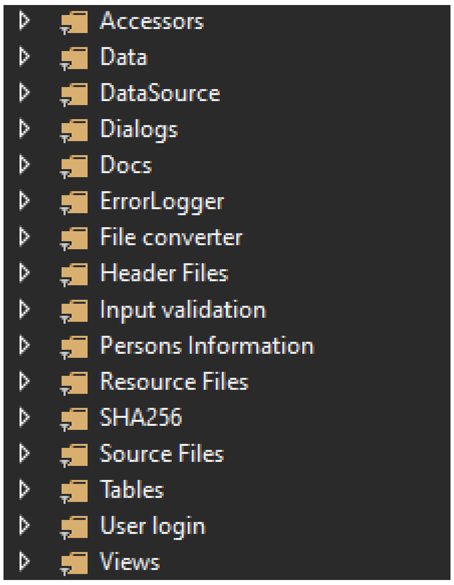
В линията с инструменти от опцията „Файл“ може да се избере „Отвори“, което дава опции за показване на друга таблица.



От опцията „Помощ“ се извежда помощен екран с инструкции за ползване на програмата.

5. Изходен код

Програмата се състои от модули съдържащи класове със сходно предназначение.



За осъществяване на достъп до базата данни се използват модулите Accessors, Data, DataSource и Tables. Модулът Docs съдържа „Документ“ класовете от архитектурата Document-View, които отговарят за кеширане на данните от базата данни и осъществяване на обратна връзка с “View” класовете.

Класовете отговорни за потребителския интерфейс се съдържат в модулите Views, User login и Dialogs.

В следния раздел са разгледани класовете отговарящи за таблицата „Persons“. Останалите класове работят по сходен начин. Пълният изходен код е достъпен в гитхъб: https://github.com/IvanDimovSIT/Phonebook

DataSourceSingleton.h

```
#pragma once
#include <atldbcli.h>

/////////////////////////////////////////////////////////////////////////////
// CDataSourceSingleton

/// <summary>
/// Singleton клас за връзка с базата данни
/// </summary>
class CDataSourceSingleton
{
// Constants
// ----------------
// Constructor / Destructor
// ----------------
private:
      CDataSourceSingleton();


// Methods
// ----------------
public:
    CDataSourceSingleton(CDataSourceSingleton& oDataSourceSingleton) = delete;
    void operator=(const CDataSourceSingleton&) = delete;
private:

      /// <summary> Показва съобщение за грешка </summary>
      void ShowErrorMessage(const HRESULT hResult) const;

      /// <summary> Логване в БД </summary>
      BOOL Login(CDBPropSet& oDBPropSet);

      BOOL ReadConfigFile(const CString& strFilePath);
public:

      /// <summary> Свързване с базата данни чрез Windows Authentication </
summary>
      BOOL OpenConnectionWindowsAuthentication();

      /// <summary> Свързване с базата данни чрез име и парола </summary>
      BOOL OpenConnectionLogin(const CString& strUsername, const CString&
strPassword);

    /// <summary> Метод за достъп до инстанцията на класа </summary>
    static CDataSourceSingleton* GetInstance();

      /// <summary> Достъп до CDataSource </summary>
      CDataSource* GetDataSource();

      /// <summary> Затваря връзката с базата данни </summary>
      void CloseConnection();

// Overrides
// ----------------

// Members
// ----------------
private:
      /// <summary> Указател към Singleton обекта </summary>
      static CDataSourceSingleton* _pInstance;
      /// <summary> Връзката с БД </summary>
      CDataSource m_oDataSource;
      /// <summary> Името на БД </summary>
      CString m_strDatasourceName;
};
```

DataSourceSingleton.cpp

```
#include "pch.h"
#include "DataSourceSingleton.h"
#include "ErrorLogger.h"
#include "iostream"
#include "fstream"
#include "PtrAutoArray.h"
```

```
//#define DATASOURCE_NAME _T("DESKTOP-VQF53N4")
#define DATABASE_AUTHENTIFICATION _T("SSPI")
#define DATABASE_NAME _T("PHONEBOOK")
#define DATABASE_PERSIST_SENSITIVE_AUTHINFO false
#define DATABASE_INIT_LCID 1033L
#define DATABASE_CLSID _T("SQLOLEDB.1")

#define CONFIG_FILE _T("config.txt")
#define LINES_TO_READ 3

enum ConfigLines
{
	ConfigLinesDataSource = 0,
	ConfigLinesUsername,
	ConfigLinesPassword
};

//#define MAX_FILE_LENGTH 100


/////////////////////////////////////////////////////////////////////////////
// CDataSourceSingleton

CDataSourceSingleton* CDataSourceSingleton::_pInstance = NULL;

// Constants
// ----------------


// Constructor / Destructor
// ----------------

CDataSourceSingleton::CDataSourceSingleton()
{
	if(!ReadConfigFile(CONFIG_FILE))
		AfxGetMainWnd()->PostMessage(WM_CLOSE);
}

// Methods
// ----------------

BOOL CDataSourceSingleton::ReadConfigFile(const CString& strFilePath)
{
	CString strLines[LINES_TO_READ];
	int nLinesRead;
	CStdioFile oFile(strFilePath, CFile::modeRead);

	for (nLinesRead = 0; nLinesRead < LINES_TO_READ; nLinesRead++)
	{
		if (!oFile.ReadString(strLines[nLinesRead]))
			break;
	}

	if (nLinesRead <= ConfigLinesDataSource)
		return FALSE;

	m_strDatasourceName = strLines[ConfigLinesDataSource];

	if (nLinesRead < ConfigLinesPassword)
	{
		return OpenConnectionWindowsAuthentication();
	}
	if	(!OpenConnectionLogin(strLines[ConfigLinesUsername],	strLines[Con-
figLinesPassword]))
	{
		return OpenConnectionWindowsAuthentication();
	}

	return TRUE;
}


CDataSourceSingleton* CDataSourceSingleton::GetInstance()
{
	if (_pInstance == NULL)
	{
		_pInstance = new CDataSourceSingleton();
	}

	return _pInstance;
}

void CDataSourceSingleton::ShowErrorMessage(const HRESULT hResult) const
{
	CString strError;
	strError.Format(_T(
		"Error connecting to DB. Error: %d\n"
		"DATASOURCE_NAME: %s\n"
		"DATABASE_NAME: %s"
	), hResult, m_strDatasourceName, DATABASE_NAME);
	CErrorLogger::LogMessage(strError, TRUE, FALSE);
}

BOOL CDataSourceSingleton::Login(CDBPropSet& oDBPropSet)
{
	const HRESULT hResult = m_oDataSource.Open(DATABASE_CLSID, &oDBPropSet);
	if (FAILED(hResult))
	{
		ShowErrorMessage(hResult);

		return FALSE;
	}

	return TRUE;
}

BOOL CDataSourceSingleton::OpenConnectionWindowsAuthentication()
{
	CDBPropSet oDBPropSet(DBPROPSET_DBINIT);
	oDBPropSet.AddProperty(DBPROP_INIT_DATASOURCE, m_strDatasourceName);
	oDBPropSet.AddProperty(DBPROP_AUTH_INTEGRATED, DATABASE_AUTHENTIFICATION);
	oDBPropSet.AddProperty(DBPROP_INIT_CATALOG, DATABASE_NAME);
	oDBPropSet.AddProperty(DBPROP_AUTH_PERSIST_SENSITIVE_AUTHINFO,
DATABASE_PERSIST_SENSITIVE_AUTHINFO);
	oDBPropSet.AddProperty(DBPROP_INIT_LCID, DATABASE_INIT_LCID);
	oDBPropSet.AddProperty(DBPROP_INIT_PROMPT, static_cast<short>(4));

	return Login(oDBPropSet);
}

BOOL CDataSourceSingleton::OpenConnectionLogin(const CString& strUsername, const
CString& strPassword)
{
	CDBPropSet oDBPropSet(DBPROPSET_DBINIT);
	oDBPropSet.AddProperty(DBPROP_INIT_DATASOURCE, m_strDatasourceName);
	oDBPropSet.AddProperty(DBPROP_AUTH_USERID, strUsername.GetString());
	oDBPropSet.AddProperty(DBPROP_AUTH_PASSWORD, strPassword.GetString());
	oDBPropSet.AddProperty(DBPROP_INIT_CATALOG, DATABASE_NAME);
	oDBPropSet.AddProperty(DBPROP_AUTH_PERSIST_SENSITIVE_AUTHINFO, 
DATABASE_PERSIST_SENSITIVE_AUTHINFO);
	oDBPropSet.AddProperty(DBPROP_INIT_LCID, DATABASE_INIT_LCID);
	oDBPropSet.AddProperty(DBPROP_INIT_PROMPT, static_cast<short>(4));

	return Login(oDBPropSet);
}


CDataSource* CDataSourceSingleton::GetDataSource()
{
	return &m_oDataSource;
}

void CDataSourceSingleton::CloseConnection()
{
	if (_pInstance == NULL)
	{
		return;
	}

	m_oDataSource.Close();
	delete _pInstance;
	_pInstance = NULL;
}

// Overrides
// ----------------
```

PersonsAccessor.h

```
#pragma once

#include <atldbcli.h>
#include "Structures.h"

#define PERSONS_ACCESSOR_MAP_VALUE 2

enum PersonsAccessorValues
{
	PersonsAccessorValuesFirst = 0,
	PersonsAccessorValuesSecond
};

enum PersonsAccessorColumnEntries
{
	PersonsAccessorColumnEntriesID = 1,
	PersonsAccessorColumnEntriesUpdateCounter,
	PersonsAccessorColumnEntriesFirstName,
	PersonsAccessorColumnEntriesMiddleName,
	PersonsAccessorColumnEntriesLastName,
	PersonsAccessorColumnEntriesUCN,
	PersonsAccessorColumnEntriesCityID,
	PersonsAccessorColumnEntriesAddress,
	PersonsAccessorColumnEntriesCompanyID,
	PersonsAccessorColumnEntriesPositionID

};

/////////////////////////////////////////////////////////////////////////////
// CPersonsAccessor
```

```
/// <summary>
/// Accessor за таблицата PERSONS
/// </summary>
class CPersonsAccessor
{
protected:
      PERSONS m_recPerson;
      BEGIN_ACCESSOR_MAP(CPersonsAccessor, PERSONS_ACCESSOR_MAP_VALUE)
            BEGIN_ACCESSOR(PersonsAccessorValuesFirst, true)
                  COLUMN_ENTRY(PersonsAccessorColumnEntriesID, m_recPerson.lID)
            END_ACCESSOR()

            BEGIN_ACCESSOR(PersonsAccessorValuesSecond, true)
                  COLUMN_ENTRY(PersonsAccessorColumnEntriesUpdateCounter,
m_recPerson.lUpdateCounter)
                  COLUMN_ENTRY(PersonsAccessorColumnEntriesFirstName,
m_recPerson.szFirstName)
                  COLUMN_ENTRY(PersonsAccessorColumnEntriesMiddleName,
m_recPerson.szMiddleName)
                  COLUMN_ENTRY(PersonsAccessorColumnEntriesLastName,
m_recPerson.szLastName)
                  COLUMN_ENTRY(PersonsAccessorColumnEntriesUCN,
m_recPerson.szUCN)
                  COLUMN_ENTRY(PersonsAccessorColumnEntriesCityID,
m_recPerson.lCityID)
                  COLUMN_ENTRY(PersonsAccessorColumnEntriesAddress,
m_recPerson.szAddress)
                  COLUMN_ENTRY(PersonsAccessorColumnEntriesCompanyID,
m_recPerson.lCompanyID)
                  COLUMN_ENTRY(PersonsAccessorColumnEntriesPositionID,
m_recPerson.lPositionID)
            END_ACCESSOR()

      END_ACCESSOR_MAP()
};
```

DBTable.h

```
#pragma once
#include "pch.h"
#include <atldbcli.h>
#include "ErrorLogger.h"
#include "PtrAutoArray.h"
#include "DataSourceSingleton.h"

#define DB_TABLE_OPEN_CONNECTION_ERROR_MESSAGE _T("Unable to open session")
#define  DB_TABLE_QUERY_ERROR_MESSAGE  _T("Error  when  executing  query.\n  Error:
%#lx.\nQuery: \"%s\"")
#define DB_TABLE_ERROR_CODE_MESSAGE _T("\nError: %#lx")

#define DB_TABLE_SELECT_ALL_QUERY  _T("SELECT * FROM %s")
#define DB_TABLE_SELECT_BY_ID_QUERY  _T("SELECT * FROM %s WHERE [ID] = %d")
#define DB_TABLE_SELECT_BY_ID_WITH_LOCK_QUERY _T("SELECT * FROM %s WITH(UPDLOCK)
WHERE ID = %d")
#define DB_TABLE_SELECT_NOTHING_QUERY _T("SELECT TOP 0 * FROM %s")

#define DB_TABLE_NO_RESULTS_ERROR_MESSAGE _T("Error no results found.")
#define DB_TABLE_UPDATE_ERROR_MESSAGE _T("Error when updating.")
#define  DB_TABLE_UPDATE_COUNTER_ERROR_MESSAGE  _T("Error  when  updating.  Update
counter doesn't match.")
```

```
#define DB_TABLE_INSERT_ERROR_MESSAGE _T("Error when inserting.")
#define DB_TABLE_SELECT_ROW_TO_DELETE_ERROR_MESSAGE _T("Error when selecting the
row to delete")
#define DB_TABLE_DELETE_ERROR_MESSAGE  _T("Error when deleting.")


/////////////////////////////////////////////////////////////////////////////
// CDBTable

/// <summary>
/// Базов клас за работа с таблица от БД.
/// </summary>
/// <typeparam name="C"> Аксесор съответстващ на таблицата </typeparam>
template <class C, class R>
class CDBTable:protected
      CCommand<CAccessor<C>>
{
// Constants
// ----------------
private:
    /// <summary> Името на таблицата </summary>
    const TCHAR* m_szTableName;
// Constructor / Destructor
// ----------------
public:
    CDBTable(R& recRecord, const TCHAR* szTableName, CSession& oSession);
    CDBTable(R& recRecord, const TCHAR* szTableName);
    ~CDBTable();

// Methods
// ----------------
private:
    /// <summary> Задава стойностите на m_oDBPropSet </summary>
    void AddPropSetProperties();

protected:
      /// <summary> Отваря нова сесия </summary>
      BOOL OpenConnection();

    /// <summary> Започва транзакция </summary>
    BOOL StartTransaction();

    /// <summary> Спира транзакция </summary>
    BOOL Abort();

    /// <summary> Завършва транзакция </summary>
    BOOL Commit();

      /// <summary> Затваря сесията </summary>
      void CloseConnection();

      /// <summary> Показва съобщение за грешка </summary>
      void ShowErrorMessage(const HRESULT hResult, const CString& strError)
const;

    /// <summary> Показва съобщение за грешка </summary>
    void ShowErrorMessage(const CString& strError) const;

      /// <summary> Показва съобщение за грешка </summary>
      void ShowErrorMessageQuery(const HRESULT hResult, const CString& strQuery)
const;

      /// <summary> Изпълнява заявка </summary>
      BOOL ExecuteQuery(const CString& strQuery);
    /// <summary> Намира запис по зададено ID, изпълнява се в текущата сесия </
summary>
    BOOL SelectWhereIDInSession(const long lID, const BOOL bWithLock);
public:
    /// <summary> Получава всички записи и пълни масива </summary>
    BOOL SelectAll(CPtrAutoArray<R>& oAutoArray);

    /// <summary> Намира запис по зададено ID </summary>
    BOOL SelectWhereID(const long lID, R& recRecord);

    /// <summary> Променя запис по ID </summary>
    BOOL UpdateWhereID(const long lID, const R& recRecord);

    /// <summary> Създава нов запис </summary>
    BOOL InsertRecord(R& recCity);

    /// <summary> Изтрива запис по ID </summary>
    BOOL DeleteWhereID(const long lID);


// Overrides
// ----------------

// Members
// ----------------
private:
    /// <summary> Указател към записа </summary>
    R* m_pRecord;
protected:
    BOOL m_bExternalConnection;

      /// <summary> Указател към връзката за БД </summary>
      CDataSource* m_pDataSource;
      /// <summary> Текущата сесия </summary>
      CSession* m_pSession;
      /// <summary> DBPropSet за заявките </summary>
      CDBPropSet m_oDBPropSet;
};


// Constants
// ----------------

// Constructor / Destructor
// ----------------

template<class C, class R>
CDBTable<C,  R>::CDBTable(R&  recRecord,  const  TCHAR*  szTableName,  CSession&
oSession) :
    m_oDBPropSet(DBPROPSET_ROWSET)
{
    m_bExternalConnection = TRUE;

    m_pSession = &oSession;
    m_pDataSource = CDataSourceSingleton::GetInstance()->GetDataSource();
    m_szTableName = szTableName;
    m_pRecord = &recRecord;

    AddPropSetProperties();
}

template<class C, class R>
CDBTable<C, R>::CDBTable(R& recRecord, const TCHAR* szTableName) :
    m_oDBPropSet(DBPROPSET_ROWSET)
{
    m_bExternalConnection = FALSE;
    m_pSession = new CSession;

    m_pDataSource = CDataSourceSingleton::GetInstance()->GetDataSource();
    m_szTableName = szTableName;
    m_pRecord = &recRecord;

    AddPropSetProperties();
}

template<class C, class R>
CDBTable<C, R>::~CDBTable()
{
    if (!m_bExternalConnection)
        delete m_pSession;
}

// Methods
// ----------------

template<class C, class R>
void CDBTable<C, R>::AddPropSetProperties()
{
    m_oDBPropSet.AddProperty(DBPROP_CANFETCHBACKWARDS, true);
    m_oDBPropSet.AddProperty(DBPROP_IRowsetScroll, true);
    m_oDBPropSet.AddProperty(DBPROP_IRowsetChange, true);
          m_oDBPropSet.AddProperty(DBPROP_UPDATABILITY,   DBPROPVAL_UP_CHANGE   |
DBPROPVAL_UP_INSERT | DBPROPVAL_UP_DELETE);
}

template<class C, class R>
BOOL CDBTable<C, R>::OpenConnection()
{
    if (m_bExternalConnection)
        return TRUE;

    const HRESULT hResult = m_pSession->Open(*m_pDataSource);
    if (FAILED(hResult))
    {
        ShowErrorMessage(hResult, DB_TABLE_OPEN_CONNECTION_ERROR_MESSAGE);
        return FALSE;
    }

    return TRUE;
}

template<class C, class R>
BOOL CDBTable<C, R>::StartTransaction()
{
    if (m_bExternalConnection)
        return TRUE;

    return !FAILED(m_pSession->StartTransaction());
}

template<class C, class R>
BOOL  CDBTable<C, R>::Abort()
{
    if (m_bExternalConnection)
        return TRUE;

    return !FAILED(m_pSession->Abort());
```

```
}

template<class C, class R>
BOOL  CDBTable<C, R>::Commit()
{
    if (m_bExternalConnection)
        return TRUE;

    return !FAILED(m_pSession->Commit());
}

template<class C, class R>
void CDBTable<C, R>::CloseConnection()
{
    if (m_bExternalConnection)
        return;

    m_pSession->Close();
}

template<class C, class R>
void CDBTable<C, R>::ShowErrorMessageQuery(const HRESULT hResult, const CString&
strQuery) const
{
    CString strError;
    strError.Format(DB_TABLE_QUERY_ERROR_MESSAGE, hResult, strQuery);

    CErrorLogger::LogMessage(strError, TRUE, FALSE);
}

template<class C, class R>
void  CDBTable<C,  R>::ShowErrorMessage(const  HRESULT  hResult,  const  CString&
strError) const
{
    CString strMessage;
    strMessage.Format(DB_TABLE_ERROR_CODE_MESSAGE, hResult);
    strMessage = strError + strMessage;

    CErrorLogger::LogMessage(strMessage, TRUE, FALSE);
}

template<class C, class R>
void CDBTable<C, R>::ShowErrorMessage(const CString& strError) const
{
    CErrorLogger::LogMessage(strError, TRUE, FALSE);
}

template<class C, class R>
BOOL CDBTable<C, R>::ExecuteQuery(const CString& strQuery)
{
    const HRESULT hResult = Open(*m_pSession, strQuery, &m_oDBPropSet);
    if (FAILED(hResult))
    {
        ShowErrorMessageQuery(hResult, strQuery);
        return FALSE;
    }

    return TRUE;
}

template<class C, class R>
BOOL   CDBTable<C,   R>::SelectWhereIDInSession(const   long   lID,   const   BOOL
bWithLock)
{
    CString strQuery;
    if (bWithLock)
    {
           strQuery.Format(DB_TABLE_SELECT_BY_ID_WITH_LOCK_QUERY, m_szTableName,
lID);
    }
    else
    {
        strQuery.Format(DB_TABLE_SELECT_BY_ID_QUERY, m_szTableName, lID);
    }

    return ExecuteQuery(strQuery.GetString());
}

template<class C, class R>
BOOL CDBTable<C, R>::SelectAll(CPtrAutoArray<R>& oAutoArray)
{
    if (!OpenConnection())
        return FALSE;


    CString strQuery;
    strQuery.Format(DB_TABLE_SELECT_ALL_QUERY, m_szTableName);
    if (!ExecuteQuery(strQuery.GetString()))
    {
        CloseConnection();
        return FALSE;
    }

    while (MoveNext() == S_OK)
    {
        R* pRecordToAdd = new R((*m_pRecord));

        oAutoArray.Add(pRecordToAdd);
    }

    Close();
    CloseConnection();
    return TRUE;
}

template<class C, class R>
BOOL CDBTable<C, R>::SelectWhereID(const long lID, R& recRecord)
{
    if (!OpenConnection())
        return FALSE;

    if (!SelectWhereIDInSession(lID, FALSE))
    {
        CloseConnection();
        return FALSE;
    }

    if (MoveNext() != S_OK)
    {
        ShowErrorMessage(DB_TABLE_NO_RESULTS_ERROR_MESSAGE);
        Close();
        CloseConnection();
        return FALSE;
    }

    recRecord = (*m_pRecord);

    Close();
    CloseConnection();

    return TRUE;
}

template<class C, class R>
BOOL CDBTable<C, R>::UpdateWhereID(const long lID, const R& recRecord)
{
    if (!OpenConnection())
        return FALSE;
    StartTransaction();

    // Начало на транзакцията

    if (!SelectWhereIDInSession(lID, TRUE))
    {
        Abort();
        CloseConnection();

        return FALSE;
    }

    HRESULT hResult = MoveFirst();
    if (FAILED(hResult))
    {
        ShowErrorMessage(hResult, DB_TABLE_NO_RESULTS_ERROR_MESSAGE);

        Abort();
        Close();
        CloseConnection();

        return FALSE;
    }

    // Проверка на Update counter
    if (recRecord.lUpdateCounter != (*m_pRecord).lUpdateCounter)
    {
        ShowErrorMessage(DB_TABLE_UPDATE_COUNTER_ERROR_MESSAGE);

        Abort();
        Close();
        CloseConnection();
        return FALSE;
    }

    (*m_pRecord) = recRecord;
    (*m_pRecord).lUpdateCounter++;

    hResult = SetData(1);
    if (FAILED(hResult))
    {
        // Неуспешно приключване на транзакцията
        Abort();
        ShowErrorMessage(hResult, DB_TABLE_UPDATE_ERROR_MESSAGE);
        Close();
        CloseConnection();

        return FALSE;
    }

    // Успешно приключване на транзакцията
    Commit();
    Close();
    CloseConnection();
    return TRUE;
}

template<class C, class R>
BOOL CDBTable<C, R>::InsertRecord(R& recRecord)
{
    if (!OpenConnection())
        return FALSE;

    // изпълняване на фиктивна команда за настройване на PropSet
    CString strQuery;
    strQuery.Format(DB_TABLE_SELECT_NOTHING_QUERY, m_szTableName);
    if (!ExecuteQuery(strQuery.GetString()))
    {
        CloseConnection();

        return FALSE;
    }

    (*m_pRecord) = recRecord;

    HRESULT hResult = Insert(1, false);

    if (FAILED(hResult))
    {
        Close();
        ShowErrorMessage(hResult, DB_TABLE_INSERT_ERROR_MESSAGE);
        CloseConnection();

        return FALSE;
    }

    hResult = MoveFirst();
    if (FAILED(hResult))
    {
        Close();
        ShowErrorMessage(hResult, DB_TABLE_INSERT_ERROR_MESSAGE);
        CloseConnection();

        return FALSE;
    }

    recRecord = (*m_pRecord);

    Close();
    CloseConnection();
    return TRUE;
}

template<class C, class R>
BOOL CDBTable<C, R>::DeleteWhereID(const long lID)
{

    if (!OpenConnection())
        return FALSE;


    CString strQuery;
    strQuery.Format(DB_TABLE_SELECT_BY_ID_QUERY, m_szTableName, lID);

    // Селектираме записа за изтриване
    if (!ExecuteQuery(strQuery))
    {
        CloseConnection();
        return FALSE;
    }

    HRESULT hResult = MoveFirst();
    if (FAILED(hResult))
    {
        ShowErrorMessage(hResult, DB_TABLE_SELECT_ROW_TO_DELETE_ERROR_MESSAGE);
        Close();
        CloseConnection();

        return FALSE;
    }

    // Изтриваме селектирания запис
    hResult = Delete();

    if (FAILED(hResult))
    {
        ShowErrorMessage(hResult, DB_TABLE_DELETE_ERROR_MESSAGE);
        Close();
        CloseConnection();

        return FALSE;
    }

    Close();
    CloseConnection();
    return TRUE;
}


// Overrides
// ----------------
```

PersonsTable.h

```
#pragma once

#include <atldbcli.h>
#include "DBTable.h"
#include "PersonsAccessor.h"

/////////////////////////////////////////////////////////////////////////////
// CPersonsTable

/// <summary>
/// Клас за работа с таблицата PERSONS
/// </summary>
class CPersonsTable : public
      CDBTable<CPersonsAccessor, PERSONS>
{
      // Constants
      // ----------------

      // Constructor / Destructor
      // ----------------
public:
      CPersonsTable(CSession& oSession);
      ~CPersonsTable();

      // Methods
      // ----------------

      /// <summary> Получава всички записи по име </summary>
      BOOL   SelectByNameUCNAddress(CPtrAutoArray<PERSONS>&   oAutoArray,   const
CString& strName, const CString& strUCN, const CString& strAddress);


      /// <summary> Получава всички записи по азбучен ред и пълни масива </sum-
mary>
      BOOL SelectAllSorted(CPtrAutoArray<PERSONS>& oAutoArray);


      // Overrides
      // ----------------

      // Members
      // ----------------
};
```

PersonsTable.cpp

```
#include "pch.h"
#include "PersonsTable.h"



/////////////////////////////////////////////////////////////////////////////
// CPersonsTable

// Constants
// ----------------

#define PERSONS_TABLE_NAME _T("PERSONS")
#define PERSON_TABLE_SELECT_ALL_SORTED_QUERY _T("SELECT * FROM PERSONS ORDER BY
[FIRST_NAME] DESC, [MIDDLE_NAME] DESC, [LAST_NAME] DESC")
#define SELECT_BY_NAME _T("SELECT * FROM PERSONS WHERE (UPPER([FIRST_NAME]) LIKE
UPPER(N'%%%s%%') AND '%s' != '' ) OR \
                                 (UPPER([MIDDLE_NAME]) LIKE UPPER(N'%%%s%%') AND
'%s' != '' ) OR \
                           (UPPER([LAST_NAME]) LIKE UPPER(N'%%%s%%') AND '%s' !=
'' ) OR \
                            (UPPER([UCN]) LIKE UPPER(N'%%%s%%') AND '%s' != '' )
OR \
                             (UPPER([ADDRESS]) LIKE UPPER(N'%%%s%%') AND '%s' !=
'' ) \
                           ORDER BY \
                           [FIRST_NAME] DESC, \
                           [MIDDLE_NAME] DESC, \
                           [LAST_NAME] DESC")

// Constructor / Destructor
// ----------------

CPersonsTable::CPersonsTable(CSession& oSession) :
    CDBTable(m_recPerson, PERSONS_TABLE_NAME, oSession)
{
}

CPersonsTable::~CPersonsTable()
{
}

// Methods
// ----------------

BOOL CPersonsTable::SelectAllSorted(CPtrAutoArray<PERSONS>& oAutoArray)
{
    if (!OpenConnection())
        return FALSE;

    if (!ExecuteQuery(PERSON_TABLE_SELECT_ALL_SORTED_QUERY))
    {
        CloseConnection();
        return FALSE;
    }

    while (MoveNext() == S_OK)
    {
        PERSONS* pRecordToAdd = new PERSONS(m_recPerson);

        oAutoArray.Add(pRecordToAdd);
    }


    Close();
    CloseConnection();
    return TRUE;
}

BOOL  CPersonsTable::SelectByNameUCNAddress(CPtrAutoArray<PERSONS>&  oAutoArray,
const CString& strName, const CString& strUCN, const CString& strAddress)
{
    if (!OpenConnection())
        return FALSE;

    CString strQuery;
    strQuery.Format(SELECT_BY_NAME, strName, strName, strName, strName, strName,
strName, strUCN, strUCN, strAddress, strAddress);
    if (!ExecuteQuery(strQuery))
    {
        CloseConnection();
        return FALSE;
    }

    while (MoveNext() == S_OK)
    {
        PERSONS* pRecordToAdd = new PERSONS(m_recPerson);

        oAutoArray.Add(pRecordToAdd);
    }


    Close();
    CloseConnection();
    return TRUE;
}



// Overrides
// ----------------
```

PersonsData.h

```
#pragma once
#include "Person.h"
#include "PersonDisplay.h"
#include "PhoneNumbersTable.h"

/////////////////////////////////////////////////////////////////////////////
// CPersonsData

/// <summary>
/// Клас съдържащ бизнес логиката за PERSONS
/// </summary>
class CPersonsData
{
	// Constants
	// ----------------

	// Constructor / Destructor
	// ----------------
public:
	CPersonsData();
	~CPersonsData();

	// Methods
	// ----------------
private:
	/// <summary> Записва и показва грешка </summary>
	void LogError(const HRESULT hResult) const;

	/// <summary> Намира съответстващите телефонни номера </summary>
	BOOL SelectPhoneNumbersForPerson(
		CPtrAutoArray<CPerson>& oPersonsArrayComplete,
		const CPtrAutoArray<PERSONS>& oPersonsArray,
		CPhoneNumbersTable& oPhoneNumbersTable) const;

	/// <summary> Намира запис по ID </summary>
	BOOL PhoneNumberInArray(const CPtrAutoArray<PHONE_NUMBERS>& oPhoneNumbers,
		const PHONE_NUMBERS& recSearched,
		PHONE_NUMBERS& recFound) const;

	/// <summary> Проверява за изтрити записи </summary>
	BOOL DeleteCheck(const long lPersonID,
		const CPtrAutoArray<PHONE_NUMBERS>& oPhoneNumbers,
		const CPtrAutoArray<PHONE_NUMBERS>& oPhoneNumbersDB,
		CPhoneNumbersTable& oPhoneNumbersTable) const;

	/// <summary> Извършва съответната операция на телефонните номера според
състоянието им </summary>
	BOOL        UpdatePersonsPhoneNumbers(const        long        lPersonID,
CPtrAutoArray<PHONE_NUMBERS>& oPhoneNumbers, CPhoneNumbersTable& oPhoneNumber-
sTable) const;

public:
	/// <summary> Получава всички записи за PERSONS и пълни масива </summary>
	BOOL SelectAll(CPtrAutoArray<CPerson>& oPersonAutoArray) const;

	/// <summary> Получава всички записи по име </summary>
	BOOL   SelectByNameUCNAddress(CPtrAutoArray<CPerson>&   oAutoArray,   const
CString& strName, const CString& strUCN, const CString& strAddress);

	/// <summary> Намира запис по зададено ID </summary>
	BOOL SelectWhereID(const long lID, CPerson& oPerson) const;
	/// <summary> Променя запис по ID </summary>
	BOOL UpdateWhereID(const long lID, CPerson& oPerson) const;

	/// <summary> Създава нов запис </summary>
	BOOL InsertPerson(CPerson& oPerson) const;

	/// <summary> Изтрива запис по ID </summary>
	BOOL DeleteWhereID(const long lID) const;

	/// <summary> Намира допълнителна информацията, нужна за визуализиране на
PERSONS </summary>
	BOOL SelectDislpayInformation(CPersonDisplay& oPersonDisplay);
	// Overrides
	// ----------------


	// Members
	// ----------------
private:
};
```

PersonsData.cpp

```
#include "pch.h"
#include "PersonsData.h"
#include "PersonsTable.h"
#include "PersonDisplay.h"
#include "PhoneNumbersTable.h"
#include "CitiesTable.h"
#include "CompaniesTable.h"
#include "PositionsTable.h"
#include "PhoneTypesTable.h"
#include "ErrorLogger.h"
#include "DataSourceSingleton.h"


/////////////////////////////////////////////////////////////////////////////
// CPersonsData

// Constants
// ----------------

#define DATA_ERROR_MESSAGE _T("Error in PersonsData. Error %#lx")

// Constructor / Destructor
// ----------------
CPersonsData::CPersonsData()
{
}

CPersonsData::~CPersonsData()
{
}

// Methods
// ----------------

void CPersonsData::LogError(const HRESULT hResult) const
{
	CString strError;
	strError.Format(DATA_ERROR_MESSAGE, hResult);
	CErrorLogger::LogMessage(strError, TRUE, TRUE);
}


BOOL CPersonsData::SelectPhoneNumbersForPerson(
	CPtrAutoArray<CPerson>& oPersonsArrayComplete,
	const CPtrAutoArray<PERSONS>& oPersonsArray,
	CPhoneNumbersTable& oPhoneNumbersTable) const
{
	for (INT_PTR i = 0; i < oPersonsArray.GetCount(); i++)
	{
		CPerson* pPersonToAdd = new CPerson;
		pPersonToAdd->m_recPerson = *oPersonsArray.GetAt(i);

		if  (!oPhoneNumbersTable.SelectWherePersonID(pPersonToAdd->m_recPer-
son.lID, pPersonToAdd->m_oPhoneNumbers))
		{
			delete pPersonToAdd;
			oPersonsArrayComplete.RemoveAll();
			return FALSE;
		}

		oPersonsArrayComplete.Add(pPersonToAdd);
	}

	return TRUE;
}

BOOL  CPersonsData::PhoneNumberInArray(const  CPtrAutoArray<PHONE_NUMBERS>&  oPho-
neNumbers,
	const PHONE_NUMBERS& recSearched,
	PHONE_NUMBERS& recFound) const
{
	for (INT_PTR i = 0; i < oPhoneNumbers.GetCount(); i++)
	{
		if (oPhoneNumbers.GetAt(i)->lID == recSearched.lID)
		{
			recFound = *oPhoneNumbers.GetAt(i);
			return TRUE;
		}
	}

	return FALSE;
}

BOOL CPersonsData::DeleteCheck(const long lPersonID,
	const CPtrAutoArray<PHONE_NUMBERS>& oPhoneNumbers,
	const CPtrAutoArray<PHONE_NUMBERS>& oPhoneNumbersDB,
	CPhoneNumbersTable& oPhoneNumbersTable) const
{
	PHONE_NUMBERS recFound;

	for (INT_PTR i = 0; i < oPhoneNumbersDB.GetCount(); i++)
	{
		if (oPhoneNumbersDB.GetAt(i)->lPersonID != lPersonID)
			continue;

		if   (PhoneNumberInArray(oPhoneNumbers,   *oPhoneNumbersDB.GetAt(i),
recFound))
			continue;

		if      (!oPhoneNumbersTable.DeleteWhereID(oPhoneNumbersDB.GetAt(i)-
>lID))
			return FALSE;
	}

	return TRUE;
}

BOOL CPersonsData::UpdatePersonsPhoneNumbers(const long lPersonID,
	CPtrAutoArray<PHONE_NUMBERS>& oPhoneNumbers,
	CPhoneNumbersTable& oPhoneNumbersTable) const
{
	CPtrAutoArray<PHONE_NUMBERS> oPhoneNumbersDB;
	if (!oPhoneNumbersTable.SelectWherePersonID(lPersonID, oPhoneNumbersDB))
		return FALSE;

	PHONE_NUMBERS recFound;
	for (INT_PTR i = 0; i < oPhoneNumbers.GetCount(); i++)
	{
		PHONE_NUMBERS* pCurrent = oPhoneNumbers.GetAt(i);


		if (!PhoneNumberInArray(oPhoneNumbersDB, *pCurrent, recFound)) //за
insert
		{
			if (!oPhoneNumbersTable.InsertRecord(*pCurrent))
				return FALSE;
		}
		else //за update
		{
			if (recFound == (*pCurrent))
				continue;

			if  (!oPhoneNumbersTable.UpdateWhereID(pCurrent->lID,  *pCur-
rent))
				return FALSE;
		}
	}

	return DeleteCheck(lPersonID, oPhoneNumbers, oPhoneNumbersDB, oPhoneNum-
bersTable);
}

BOOL CPersonsData::SelectAll(CPtrAutoArray<CPerson>& oPersonAutoArray) const
{
	CSession oSession;
	CPersonsTable oPersonsTable(oSession);
	CPhoneNumbersTable oPhoneNumbersTable(oSession);
	oSession.Open(*CDataSourceSingleton::GetInstance()->GetDataSource());


	CPtrAutoArray<PERSONS> oPersonsPartialArray;
	if (!oPersonsTable.SelectAllSorted(oPersonsPartialArray))
	{
		oSession.Close();
		return FALSE;
	}


	BOOL bResult = SelectPhoneNumbersForPerson(oPersonAutoArray, oPersonsPar-
tialArray, oPhoneNumbersTable);
	oSession.Close();
	return bResult;
}

BOOL  CPersonsData::SelectByNameUCNAddress(CPtrAutoArray<CPerson>&  oAutoArray,
const CString& strName, const CString& strUCN, const CString& strAddress)
```

```
{
      CSession oSession;
      CPersonsTable oPersonsTable(oSession);
      CPhoneNumbersTable oPhoneNumbersTable(oSession);
      oSession.Open(*CDataSourceSingleton::GetInstance()->GetDataSource());


      CPtrAutoArray<PERSONS> oPersonsPartialArray;
      if  (!oPersonsTable.SelectByNameUCNAddress(oPersonsPartialArray,  strName,
strUCN, strAddress))
      {
            oSession.Close();
            return FALSE;
      }


      BOOL bResult = SelectPhoneNumbersForPerson(oAutoArray, oPersonsPartialAr-
ray, oPhoneNumbersTable);
      oSession.Close();
      return bResult;
}



BOOL CPersonsData::SelectWhereID(const long lID, CPerson& oPerson) const
{
      CSession oSession;
      CPersonsTable oPersonsTable(oSession);
      CPhoneNumbersTable oPhoneNumbersTable(oSession);
      oSession.Open(*CDataSourceSingleton::GetInstance()->GetDataSource());


      if (!oPersonsTable.SelectWhereID(lID, oPerson.m_recPerson))
      {
            oSession.Close();
            return FALSE;
      }

      BOOL bResult = oPhoneNumbersTable.SelectWherePersonID(lID, oPerson.m_oPho-
neNumbers);
      oSession.Close();
      return bResult;
}

BOOL CPersonsData::UpdateWhereID(const long lID, CPerson& oPerson) const
{
      CSession oSession;
      CPersonsTable oPersonsTable(oSession);
      CPhoneNumbersTable oPhoneNumbersTable(oSession);

      oSession.Open(*CDataSourceSingleton::GetInstance()->GetDataSource());

      HRESULT hResult = oSession.StartTransaction();
      if (FAILED(hResult))
      {
            LogError(hResult);
            oSession.Close();
            return FALSE;
      }

      if (!oPersonsTable.UpdateWhereID(lID, oPerson.m_recPerson))
      {
            oSession.Abort();
            oSession.Close();
```

```
		return FALSE;
	}

	if  (!UpdatePersonsPhoneNumbers(oPerson.m_recPerson.lID,  oPerson.m_oPho-
neNumbers, oPhoneNumbersTable))
	{
		oSession.Abort();
		oSession.Close();
		return FALSE;
	}

	hResult = oSession.Commit();
	if (FAILED(hResult))
	{
		LogError(hResult);
		oSession.Close();
		return FALSE;
	}

	oSession.Close();
	return TRUE;
}

BOOL CPersonsData::InsertPerson(CPerson& oPerson) const
{
	CSession oSession;
	CPersonsTable oPersonsTable(oSession);
	CPhoneNumbersTable oPhoneNumbersTable(oSession);
	oSession.Open(*CDataSourceSingleton::GetInstance()->GetDataSource());


	HRESULT hResult = oSession.StartTransaction();
	if (FAILED(hResult))
	{
		LogError(hResult);
		oSession.Close();
		return FALSE;
	}

	if (!oPersonsTable.InsertRecord(oPerson.m_recPerson))
	{
		oSession.Abort();
		oSession.Close();
		return FALSE;
	}

	// Задаваме ID-то на собственика на телефоните номера
	for (INT_PTR i = 0; i < oPerson.m_oPhoneNumbers.GetCount(); i++)
	{
		oPerson.m_oPhoneNumbers.GetAt(i)->lPersonID  =  oPerson.m_recPerson-
.lID;
	}

	for (INT_PTR i = 0; i < oPerson.m_oPhoneNumbers.GetCount(); i++)
	{
		if                                                                (!
oPhoneNumbersTable.InsertRecord(*oPerson.m_oPhoneNumbers.GetAt(i)))
		{
			oSession.Abort();
			oSession.Close();
			return FALSE;
		}
	}
	hResult = oSession.Commit();
	if (FAILED(hResult))
	{
		LogError(hResult);
		oSession.Close();
		return FALSE;
	}

	oSession.Close();
	return TRUE;
}

BOOL CPersonsData::DeleteWhereID(const long lID) const
{
	CSession oSession;
	CPersonsTable oPersonsTable(oSession);
	CPhoneNumbersTable oPhoneNumbersTable(oSession);
	oSession.Open(*CDataSourceSingleton::GetInstance()->GetDataSource());

	HRESULT hResult = oSession.StartTransaction();
	if (FAILED(hResult))
	{
		LogError(hResult);
		oSession.Close();
		return FALSE;
	}

	if (!oPhoneNumbersTable.DeleteWherePersonID(lID))
	{
		oSession.Abort();
		oSession.Close();
		return FALSE;
	}

	if (!oPersonsTable.DeleteWhereID(lID))
	{
		oSession.Abort();
		oSession.Close();
		return FALSE;
	}

	hResult = oSession.Commit();
	if (FAILED(hResult))
	{
		LogError(hResult);
		oSession.Close();
		return FALSE;
	}

	oSession.Close();
	return TRUE;
}

BOOL CPersonsData::SelectDislpayInformation(CPersonDisplay& oPersonDisplay)
{
	CSession oSession;
	CPhoneTypesTable oPhoneTypesTable(oSession);
	CCitiesTable oCitiesTable(oSession);
	CCompaniesTable oCompaniesTable(oSession);
	CPositionsTable oPositionsTable(oSession);
	oSession.Open(*CDataSourceSingleton::GetInstance()->GetDataSource());


	BOOL bResult =
            oPhoneTypesTable.SelectAll(*oPersonDisplay.GetPhoneTypes()) &&
            oCitiesTable.SelectAll(*oPersonDisplay.GetCities()) &&
            oCompaniesTable.SelectAll(*oPersonDisplay.GetCompanies()) &&
            oPositionsTable.SelectAll(*oPersonDisplay.GetPositions());

      oSession.Close();

      return bResult;
}


// Overrides
// ----------------
```

PhonebookDoc.h

```
#pragma once

#include "pch.h"
#include "PtrAutoArray.h"
#include "ErrorLogger.h"


#define ROW_INDEX_NOT_FOUND -1

/////////////////////////////////////////////////////////////////////////////
// CPhonebookDoc

/// <summary>
/// Базов Document клас
/// </summary>

template<class T>
class CPhonebookDoc : public CDocument
{
protected: // create from serialization only
      CPhonebookDoc() noexcept;

      // Attributes
protected:
      /// <summary> Масив с данните </summary>
      CPtrAutoArray<T> m_oArray;
public:

      // Operations
protected:

      /// <summary> Намира индекса на записа по ID </summary>
      virtual INT_PTR GetRowIndexByID(const long lID) const = 0;

      /// <summary> Зарежда запис от Data класа </summary>
      virtual BOOL LoadFromData(const long lID) = 0;
public:
      /// <summary> Получаване на данни за четене </summary>
      const CPtrAutoArray<T>* GetData() const;

      /// <summary> Получаване на запис по ID </summary>
      T* GetRowByID(const long lID);


      // Overrides
public:
	virtual BOOL OnNewDocument() = 0;
public:
	virtual ~CPhonebookDoc();

};

template<class T>
CPhonebookDoc<T>::CPhonebookDoc() noexcept
{
	theApp.CloseAllDocuments(TRUE);
}

template<class T>
CPhonebookDoc<T>::~CPhonebookDoc()
{
}

template<class T>
const CPtrAutoArray<T>* CPhonebookDoc<T>::GetData() const
{
	return &m_oArray;
}

template<class T>
T* CPhonebookDoc<T>::GetRowByID(const long lID)
{
	if (!LoadFromData(lID))
		CErrorLogger::LogMessage(_T("Error  loading  row  from  Database"),
TRUE, TRUE);

	const INT_PTR nIndex = GetRowIndexByID(lID);
	if (nIndex == ROW_INDEX_NOT_FOUND)
		return NULL;

	return m_oArray.GetAt(nIndex);
```

PersonsDoc.h

```
// CitiesDoc.h : interface of the CCitiesDoc class
//

#pragma once
#include "PersonsData.h"
#include "PersonDisplay.h"
#include "PtrAutoArray.h"
#include "Structures.h"
#include "Person.h"
#include "PhonebookDoc.h"


////////////////////////////////////////////////////////////////////////////
// CPersonsDoc

/// <summary>
/// Document клас за таблицата PERSONS
/// </summary>
class CPersonsDoc : public CPhonebookDoc<CPerson>
{
protected: // create from serialization only
	CPersonsDoc() noexcept;
```

```
      DECLARE_DYNCREATE(CPersonsDoc)

      // Attributes
private:
      /// <summary> Допълнителни данни за хората </summary>
      CPersonDisplay m_oPersonDisplay;

      /// <summary> Обект за достъп до данните на БД </summary>
      CPersonsData m_oPersonsData;

public:

      // Operations
private:

      void SetPhoneNumberFK(CPerson& oPerson);

      /// <summary> Намира индекса на записа по ID </summary>
      INT_PTR GetRowIndexByID(const long lID) const override;

      /// <summary> Зарежда запис от Data класа </summary>
      BOOL LoadFromData(const long lID) override;
public:

      /// <summary> Получаване на допълнителни данни за представяне </summary>
      CPersonDisplay* GetDisplayInformation();

      /// <summary> Получава всички записи по име </summary>
      BOOL   SelectByNameUCNAddress(CPtrAutoArray<CPerson>&   oAutoArray,   const
CString& strName, const CString& strUCN, const CString& strAddress);

      /// <summary> Промяна на запис по ID </summary>
      BOOL SetPersonByID(const long lID, CPerson& oPerson);

      /// <summary> Добавяне на запис </summary>
      BOOL AddPerson(const CPerson& oPerson);

      /// <summary> Изтриване на запис </summary>
      BOOL RemovePerson(const long lID);

      // Overrides
public:
      virtual BOOL OnNewDocument();
      virtual void Serialize(CArchive& ar);
#ifdef SHARED_HANDLERS
      virtual void InitializeSearchContent();
      virtual void OnDrawThumbnail(CDC& dc, LPRECT lprcBounds);
#endif // SHARED_HANDLERS

      // Implementation
public:
      virtual ~CPersonsDoc();
#ifdef _DEBUG
      virtual void AssertValid() const;
      virtual void Dump(CDumpContext& dc) const;
#endif

protected:

      // Generated message map functions
protected:
      DECLARE_MESSAGE_MAP()

#ifdef SHARED_HANDLERS
```

```
	// Helper function that sets search content for a Search Handler
	void SetSearchContent(const CString& value);
#endif // SHARED_HANDLERS
};
```

PersonsDoc.cpp

```
#include "pch.h"
#include "framework.h"
// SHARED_HANDLERS can be defined in an ATL project implementing preview, thumb-
nail
// and search filter handlers and allows sharing of document code with that
project.
#ifndef SHARED_HANDLERS
#include "Phonebook.h"
#endif

#include "PersonsDoc.h"
#include "DocumentDataOperation.h"
#include "ErrorLogger.h"

#include <propkey.h>

#ifdef _DEBUG
#define new DEBUG_NEW
#endif

/////////////////////////////////////////////////////////////////////////////
// CPersonsDoc


IMPLEMENT_DYNCREATE(CPersonsDoc, CDocument)

BEGIN_MESSAGE_MAP(CPersonsDoc, CDocument)
END_MESSAGE_MAP()


// Constants
// ----------------

// Constructor / Destructor
// ----------------

CPersonsDoc::CPersonsDoc() noexcept : CPhonebookDoc()
{
	// TODO: add one-time construction code here

}

CPersonsDoc::~CPersonsDoc()
{
}

// Methods
// ----------------
INT_PTR CPersonsDoc::GetRowIndexByID(const long lID) const
{
	for (INT_PTR i = 0; i < m_oArray.GetCount(); i++)
	{
		if (lID == m_oArray.GetAt(i)->m_recPerson.lID)
		{
			return i;
		}
	}

	return ROW_INDEX_NOT_FOUND;
}

BOOL CPersonsDoc::LoadFromData(const long lID)
{
	const INT_PTR nIndex = GetRowIndexByID(lID);
	if (nIndex != ROW_INDEX_NOT_FOUND)
	{
		return m_oPersonsData.SelectWhereID(lID, *(m_oArray.GetAt(nIndex)));
	}

	CPerson* pPerson = new CPerson;
	if (!m_oPersonsData.SelectWhereID(lID, (*pPerson)))
	{
		delete pPerson;
		return FALSE;
	}
	m_oArray.Add(pPerson);

	return TRUE;
}

CPersonDisplay* CPersonsDoc::GetDisplayInformation()
{
	return &m_oPersonDisplay;
}

void CPersonsDoc::SetPhoneNumberFK(CPerson& oPerson)
{
	for (INT_PTR i = 0; i < oPerson.m_oPhoneNumbers.GetCount(); i++)
	{
		oPerson.m_oPhoneNumbers.GetAt(i)->lPersonID  =  oPerson.m_recPerson-
.lID;
	}
}

BOOL CPersonsDoc::SetPersonByID(const long lID, CPerson& oPerson)
{
	const INT_PTR nIndex = GetRowIndexByID(lID);
	if (nIndex == ROW_INDEX_NOT_FOUND)
		return FALSE;

	SetPhoneNumberFK(oPerson);

	if (!m_oPersonsData.UpdateWhereID(lID, oPerson))
		return FALSE;

	if (!LoadFromData(lID))
		return FALSE;

	UpdateAllViews(NULL, (LPARAM)DocumentDataOperationUpdate, (CObject*)m_oAr-
ray.GetAt(nIndex));

	return TRUE;
}

BOOL CPersonsDoc::AddPerson(const CPerson& oPerson)
{

	CPerson* pAddedPerson = new CPerson(oPerson);
	if (!m_oPersonsData.InsertPerson(*pAddedPerson))
	{
		delete pAddedPerson;
		return FALSE;
	}

	m_oArray.Add(pAddedPerson);
	UpdateAllViews(NULL,			(LPARAM)DocumentDataOperationInsert,
(CObject*)pAddedPerson);
	return TRUE;
}

BOOL CPersonsDoc::RemovePerson(const long lID)
{
	CPerson* pPerson = GetRowByID(lID);

	if (pPerson == NULL)
		return FALSE;

	const INT_PTR nIndex = GetRowIndexByID(lID);

	if (!m_oPersonsData.DeleteWhereID(lID))
		return FALSE;

	UpdateAllViews(NULL,  (LPARAM)DocumentDataOperationDelete,  (CObject*)pPer-
son);

	m_oArray.RemoveAt(nIndex);

	return TRUE;
}

/// <summary> Получава всички записи по име </summary>
BOOL   CPersonsDoc::SelectByNameUCNAddress(CPtrAutoArray<CPerson>&   oAutoArray,
const CString& strName, const CString& strUCN, const CString& strAddress)
{
	oAutoArray.RemoveAll();
	return  m_oPersonsData.SelectByNameUCNAddress(oAutoArray,  strName,  strUCN,
strAddress);
}


BOOL CPersonsDoc::OnNewDocument()
{
	if (!CDocument::OnNewDocument())
		return FALSE;



	return m_oPersonsData.SelectAll(m_oArray) &&
		m_oPersonsData.SelectDislpayInformation(m_oPersonDisplay);
}
```

PhonebookListView.h

```
#pragma once

/////////////////////////////////////////////////////////////////////////////
// CPhonebookListView
```

```
#define INDEX_BY_ID_ERROR -1

/// <summary>
/// Помощен клас за ListView
/// </summary>
class CPhonebookListView : public CListView
{
// Constants
// ----------------

// Constructor / Destructor
// ----------------
public:
CPhonebookListView();
~CPhonebookListView();

// Methods
// ----------------
private:
      /// <summary> Задава стила на таблицата </summary>
      void SetStyle(CListCtrl& oListCtrl);

protected:
      /// <summary> Проверява дали е селектиран ред </summary>
      virtual BOOL IsSelectedRow() = 0;

      /// <summary> Намиране на ред по ID </summary>
      int GetIndexByID(CListCtrl& oListCtrl, const long lID) const;

      /// <summary> Задава колоните на таблицата </summary>
      virtual void SetColumns(CListCtrl& oListCtrl) = 0;

      /// <summary> Задава началнните стойности </summary>
      virtual void SetInitialData(CListCtrl& oListCtrl) = 0;

      virtual BOOL CanSearch() = 0;

      virtual BOOL CanConvert() = 0;

      /// <summary> Инитиализира контролата </summary>
      void Initialize(CListCtrl& oListCtrl);

      /// <summary> Задава цветовете </summary>
      void SetRowColors(NMHDR* pNMHDR, LRESULT* pResult);

      virtual afx_msg void OnContextDelete() = 0;
      virtual afx_msg void OnContextView() = 0;

protected:
      DECLARE_MESSAGE_MAP()
      afx_msg void OnKeyUp(UINT nChar, UINT nRepCnt, UINT nFlags);
      afx_msg void OnRButtonUp(UINT nFlags, CPoint point);
      afx_msg void OnContextMenu(CWnd* pWnd, CPoint point);
      afx_msg void OnLButtonDblClk(UINT nFlags, CPoint point);//double click

// Overrides
// ----------------

// Members
// ----------------
};
```

PhonebookListView.cpp

```
#include "pch.h"
#include "resource.h"
#include "PhonebookListView.h"
#include "ErrorLogger.h"

/////////////////////////////////////////////////////////////////////////////
// CPhonebookListView

// Constants
// ----------------

#define COLOR_TEXT RGB(0, 0, 0)
#define COLOR_BACKGROUND_LIGHT RGB(240, 240, 255)
#define COLOR_BACKGROUND_DARK RGB(220, 220, 240)

#define STYLE_ERROR_MESSAGE _T("Грешка при настройката на стила.")

// Constructor / Destructor
// ----------------

CPhonebookListView::CPhonebookListView()
{
}

CPhonebookListView::~CPhonebookListView()
{
}

// Methods
// ----------------

BEGIN_MESSAGE_MAP(CPhonebookListView, CListView)
      ON_WM_LBUTTONDBLCLK()//LMB double click
      ON_WM_KEYUP()
      ON_WM_CONTEXTMENU()
      ON_WM_RBUTTONUP()
END_MESSAGE_MAP()


int CPhonebookListView::GetIndexByID(CListCtrl& oListCtrl, const long lID) const
{
      for (int i = 0; i < oListCtrl.GetItemCount(); i++)
      {
            if (oListCtrl.GetItemData(i) == lID)
            {
                  return i;
            }
      }

      return INDEX_BY_ID_ERROR;
}


void CPhonebookListView::SetStyle(CListCtrl& oListCtrl)
{
      if (!oListCtrl.ModifyStyle(LVS_TYPEMASK, LVS_REPORT))
            CErrorLogger::LogMessage(STYLE_ERROR_MESSAGE, TRUE, TRUE);

      oListCtrl.SetExtendedStyle(oListCtrl.GetExtendedStyle() | LVS_EX_FULLROWS-
ELECT | LVS_EX_GRIDLINES);
      oListCtrl.SetBkColor(COLOR_BACKGROUND_LIGHT);
```

```
}

void CPhonebookListView::Initialize(CListCtrl& oListCtrl)
{
	SetColumns(oListCtrl);
	SetInitialData(oListCtrl);
	SetStyle(oListCtrl);
}


void CPhonebookListView::SetRowColors(NMHDR* pNMHDR, LRESULT* pResult)
{
	LPNMLVCUSTOMDRAW pLVCDCustomDraw = reinterpret_cast<LPNMLVCUSTOMDRAW>(pN-
MHDR);
	*pResult = CDRF_DODEFAULT;

	switch (pLVCDCustomDraw->nmcd.dwDrawStage)
	{
	case CDDS_PREPAINT:
		*pResult = CDRF_NOTIFYITEMDRAW;
		break;
	case CDDS_PREPAINT | CDDS_ITEM:
	{
		COLORREF dwEven = COLOR_BACKGROUND_LIGHT;
		COLORREF dwOdd = COLOR_BACKGROUND_DARK;
		COLORREF dwText = COLOR_TEXT;

		if ((pLVCDCustomDraw->nmcd.dwItemSpec % 2) == 0)
			pLVCDCustomDraw->clrTextBk = dwEven;
		else
			pLVCDCustomDraw->clrTextBk = dwOdd;
		pLVCDCustomDraw->clrText = dwText;
	}
	break;
	default:
		break;

	}
}

void CPhonebookListView::OnRButtonUp(UINT /* nFlags */, CPoint point)
{
	ClientToScreen(&point);
	OnContextMenu(this, point);
}

void CPhonebookListView::OnContextMenu(CWnd* /* pWnd */, CPoint point)
{
#ifndef SHARED_HANDLERS
	CMenu oContextMenu;
	oContextMenu.LoadMenu(IDR_POPUP);
	CMenu* oPopupMenu = oContextMenu.GetSubMenu(0);

	if (!IsSelectedRow())
	{
		oPopupMenu->EnableMenuItem(ID_POPUP_DELETE, MF_BYCOMMAND | MF_DIS-
ABLED | MF_GRAYED);
		oPopupMenu->EnableMenuItem(ID_POPUP_EDIT, MF_BYCOMMAND | MF_DISABLED
| MF_GRAYED);
		oPopupMenu->EnableMenuItem(ID_POPUP_VIEW, MF_BYCOMMAND | MF_DISABLED
| MF_GRAYED);
	}
```

```
      if (!CanSearch())
      {
            oPopupMenu->EnableMenuItem(ID_POPUP_SEARCH, MF_BYCOMMAND | MF_DIS-
ABLED | MF_GRAYED);
      }

      if (!CanConvert())
      {
            oPopupMenu->EnableMenuItem(ID_POPUP_CONVERT, MF_BYCOMMAND | MF_DIS-
ABLED | MF_GRAYED);
      }

      oPopupMenu->TrackPopupMenu(TPM_LEFTALIGN | TPM_RIGHTBUTTON, point.x,
point.y, this);

#endif
}

afx_msg void CPhonebookListView::OnKeyUp(UINT nChar, UINT nRepCnt, UINT nFlags)
{
      if (nChar == VK_DELETE)
      {
            OnContextDelete();
      }
}

void CPhonebookListView::OnLButtonDblClk(UINT nFlags, CPoint point)
{
      OnContextView();

      __super::OnLButtonDblClk(nFlags, point);
}


// Overrides
// ----------------
```

PersonsView.h

```
#pragma once
#include "PersonDisplay.h"
#include "PhonebookListView.h"

/////////////////////////////////////////////////////////////////////////////
// CPersonsView

/// <summary>
/// View клас за таблицата CITIES
/// </summary>
class CPersonsView : public CPhonebookListView
{
private:
      CPtrAutoArray<CPerson> m_oAutoArray;
      CString m_strSearchName;
      CString m_strSearchUCN;
      CString m_strSearchAddress;
      BOOL m_bIsSearch = FALSE;

protected: // create from serialization only
      CPersonsView() noexcept;
      DECLARE_DYNCREATE(CPersonsView)

      // Attributes
```

```
public:
	CPersonsDoc* GetDocument() const;

	// Operations
public:

private:
	void ShowSearch(CListCtrl& oListCtrl);

	/// <summary> Задава стойностите на елемент от ListCtrl </summary>
	void SetListViewItem(CListCtrl& oListCtrl, const CPerson& oPerson, const 
CPersonDisplay& oPersonDisplay, int nIndex);

	/// <summary> Намиране на селектирания град, NULL ако няма селектиран </
summary>
	CPerson* GetSelectedPerson() const;

	/// <summary> Промяна на запис </summary>
	void OperationUpdate(CListCtrl& oListCtrl, const CPerson& oPerson);

	/// <summary> Въвеждане на нов запис </summary>
	void OperationInsert(CListCtrl& oListCtrl, const CPerson& oPerson);

	/// <summary> Изтриване на запис </summary>
	void OperationDelete(CListCtrl& oListCtrl, const CPerson& oPerson);

	// Overrides
public:

	virtual BOOL PreCreateWindow(CREATESTRUCT& cs);
	void OnUpdate(CView* pSender, LPARAM lHint, CObject* pHint) override;
protected:
	/// <summary> Задаване на колоните </summary>
	void SetColumns(CListCtrl& oListCtrl) override;

	/// <summary> Задаване на първоначлните данни </summary>
	void SetInitialData(CListCtrl& oListCtrl) override;

	virtual void OnInitialUpdate(); // called first time after construct

	virtual BOOL CanSearch() override;

	virtual BOOL CanConvert() override;

// Implementation
public:
	virtual ~CPersonsView();
#ifdef _DEBUG
	virtual void AssertValid() const;
	virtual void Dump(CDumpContext& dc) const;
#endif

protected:
	BOOL IsSelectedRow() override;

	// Generated message map functions
protected:
	afx_msg void OnFilePrintPreview();
	DECLARE_MESSAGE_MAP()
public:
	afx_msg void OnContextEdit();
	afx_msg void OnContextAdd();
	afx_msg void OnContextDelete() override;
	afx_msg void OnContextView() override;
	afx_msg void OnContextSearch();
	afx_msg void OnContextConvert();
	afx_msg void OnNMCustomdraw(NMHDR* pNMHDR, LRESULT* pResult);
};

#ifndef _DEBUG  // debug version in CitiesView.cpp
inline CPersonsDoc* CPersonsView::GetDocument() const
{
	return reinterpret_cast<CPesonsDoc*>(m_pDocument);
}
#endif
```

PersonsView.cpp

```
#include "Structures.h"
#include "PersonsTable.h"
#include "pch.h"
#include "framework.h"
// SHARED_HANDLERS can be defined in an ATL project implementing preview, thumb-
nail
// and search filter handlers and allows sharing of document code with that
project.
#ifndef SHARED_HANDLERS
#include "Phonebook.h"
#endif

#include "PersonsDoc.h"
#include "PersonsView.h"
#include "PersonsDialog.h"
#include "DocumentDataOperation.h"
#include "PersonsSearchDialog.h"
#include "PersonsFileConverter.h"

/////////////////////////////////////////////////////////////////////////////
// CPersonsView

// Constants
// ----------------
#define ALL_ITEMS_NUMBER 1000000

/// <summary>
/// enum отговарящ на колона от Cities ListView
/// </summary>
enum PersonsViewColumn
{
	PersonsViewColumnFirstName = 0,
	PersonsViewColumnMiddleName,
	PersonsViewColumnLastName,
	PersonsViewColumnUCN,
	PersonsViewColumnCityName,
	PersonsViewColumnDistrict,
	PersonsViewColumnAddress,
	PersonsViewColumnCompany,
	PersonsViewColumnPosition
};

#define PERSON_FIRST_NAME_COLUMN_WIDTH 150
#define PERSON_MIDDLE_NAME_COLUMN_WIDTH 150
#define PERSON_LAST_NAME_COLUMN_WIDTH 150
#define PERSON_UCN_COLUMN_WIDTH 80
#define PERSON_CITY_NAME_COLUMN_WIDTH 150
#define PERSON_DISTRICT_COLUMN_WIDTH 150
```

```
#define PERSON_ADDRESS_COLUMN_WIDTH 220
#define PERSON_COMPANY_COLUMN_WIDTH 150
#define PERSON_POSITION_COLUMN_WIDTH 150


#define PERSON_FIRST_NAME_COLUMN_NAME _T("Име")
#define PERSON_MIDDLE_NAME_COLUMN_NAME _T("Презиме")
#define PERSON_LAST_NAME_COLUMN_NAME _T("Фамилия")
#define PERSON_UCN_COLUMN_NAME _T("ЕГН")
#define PERSON_CITY_NAME_COLUMN_NAME  _T("Град")
#define PERSON_DISTRICT_COLUMN_NAME _T("Област")
#define PERSON_ADDRESS_COLUMN_NAME _T("Адрес")
#define PERSON_COMPANY_COLUMN_NAME _T("Компания")
#define PERSON_POSITION_COLUMN_NAME _T("Позиция")


#define UPDATE_ERROR_MESSAGE  _T("Грешка при редактиране на данните. Моля рестартирайте приложението.")
#define INSERT_ERROR_MESSAGE  _T("Грешка при въвеждане на данните. Моля рестартирайте приложението.")
#define DELETE_ERROR_MESSAGE  _T("Грешка при изтриване на данните. Моля рестартирайте приложението.")
#define DELETE_CONFIRM_MESSAGE _T("Потвърдете изтриването на данните.")

#define SETTING_ITEM_TEXT_ERROR_MESSAGE _T("Грешка при въвеждането на текста във View.")
#define REMOVE_ITEM_ERROR_MESSAGE _T("Грешка при премахване на ред от View.")
#define OPERATION_NOT_RECOGNISED_ERROR_MESSAGE _T("Не е разпозната операцията.")

#ifdef _DEBUG
#define new DEBUG_NEW
#endif


IMPLEMENT_DYNCREATE(CPersonsView, CPhonebookListView)

BEGIN_MESSAGE_MAP(CPersonsView, CPhonebookListView)
	ON_NOTIFY_REFLECT(NM_CUSTOMDRAW, &CPersonsView::OnNMCustomdraw)
	ON_COMMAND(ID_POPUP_EDIT, &CPersonsView::OnContextEdit)
	ON_COMMAND(ID_POPUP_ADD, &CPersonsView::OnContextAdd)
	ON_COMMAND(ID_POPUP_DELETE, &CPersonsView::OnContextDelete)
	ON_COMMAND(ID_POPUP_VIEW, &CPersonsView::OnContextView)
	ON_COMMAND(ID_POPUP_SEARCH, &CPersonsView::OnContextSearch)
	ON_COMMAND(ID_POPUP_CONVERT, &CPersonsView::OnContextConvert)
END_MESSAGE_MAP()




// Constructor / Destructor
// ----------------

CPersonsView::CPersonsView() noexcept
{
}

CPersonsView::~CPersonsView()
{
}

// Methods
// ----------------

BOOL CPersonsView::PreCreateWindow(CREATESTRUCT& cs)
```

```
{
      return CListView::PreCreateWindow(cs);
}

CPerson* CPersonsView::GetSelectedPerson() const
{
      const CListCtrl& oListCtrl = GetListCtrl();
      POSITION oPosition = oListCtrl.GetFirstSelectedItemPosition();
      if (oPosition == NULL)
            return NULL;

      const int nItem = oListCtrl.GetNextSelectedItem(oPosition);
      const long lID = oListCtrl.GetItemData(nItem);

      return GetDocument()->GetRowByID(lID);
}

BOOL CPersonsView::IsSelectedRow()
{
      return GetSelectedPerson() != NULL;
}

void CPersonsView::SetListViewItem(CListCtrl& oListCtrl, const CPerson& oPerson,
const CPersonDisplay& oPersonDisplay, int nItemIndex)
{
      if  (!oListCtrl.SetItemText(nItemIndex,  PersonsViewColumnFirstName,  oPer-
son.m_recPerson.szFirstName))
            CErrorLogger::LogMessage(SETTING_ITEM_TEXT_ERROR_MESSAGE,       TRUE,
TRUE);

      if  (!oListCtrl.SetItemText(nItemIndex,  PersonsViewColumnMiddleName,  oPer-
son.m_recPerson.szMiddleName))
            CErrorLogger::LogMessage(SETTING_ITEM_TEXT_ERROR_MESSAGE,       TRUE,
TRUE);

      if  (!oListCtrl.SetItemText(nItemIndex,  PersonsViewColumnLastName,  oPer-
son.m_recPerson.szLastName))
            CErrorLogger::LogMessage(SETTING_ITEM_TEXT_ERROR_MESSAGE,       TRUE,
TRUE);

      if        (!oListCtrl.SetItemText(nItemIndex,        PersonsViewColumnUCN,
oPerson.m_recPerson.szUCN))
            CErrorLogger::LogMessage(SETTING_ITEM_TEXT_ERROR_MESSAGE,       TRUE,
TRUE);

      CITIES recCity;
      if (!oPersonDisplay.GetCityByID(oPerson.m_recPerson.lCityID, recCity))
      {
            CErrorLogger::LogMessage(SETTING_ITEM_TEXT_ERROR_MESSAGE,       TRUE,
TRUE);
            return;
      }

      COMPANIES recCompany;
      if (!oPersonDisplay.GetCompanyByID(oPerson.m_recPerson.lCompanyID, recCom-
pany))
      {
            CErrorLogger::LogMessage(SETTING_ITEM_TEXT_ERROR_MESSAGE,       TRUE,
TRUE);
            return;
      }

      POSITIONS recPosition;
```

```
	if (!oPersonDisplay.GetPositionByID(oPerson.m_recPerson.lPositionID, rec-
Position))
	{
		CErrorLogger::LogMessage(SETTING_ITEM_TEXT_ERROR_MESSAGE,	TRUE,
TRUE);
		return;
	}

	if (!oListCtrl.SetItemText(nItemIndex, PersonsViewColumnCityName, recCi-
ty.szCityName))
		CErrorLogger::LogMessage(SETTING_ITEM_TEXT_ERROR_MESSAGE,	TRUE,
TRUE);

	if (!oListCtrl.SetItemText(nItemIndex, PersonsViewColumnDistrict, recCi-
ty.szDistrict))
		CErrorLogger::LogMessage(SETTING_ITEM_TEXT_ERROR_MESSAGE,	TRUE,
TRUE);

	if (!oListCtrl.SetItemText(nItemIndex, PersonsViewColumnAddress, oPer-
son.m_recPerson.szAddress))
		CErrorLogger::LogMessage(SETTING_ITEM_TEXT_ERROR_MESSAGE,	TRUE,
TRUE);

	if (!oListCtrl.SetItemText(nItemIndex, PersonsViewColumnCompany, recCompa-
ny.szCompanyName))
		CErrorLogger::LogMessage(SETTING_ITEM_TEXT_ERROR_MESSAGE,	TRUE,
TRUE);

	if (!oListCtrl.SetItemText(nItemIndex, PersonsViewColumnPosition, recPosi-
tion.szPositionName))
		CErrorLogger::LogMessage(SETTING_ITEM_TEXT_ERROR_MESSAGE,	TRUE,
TRUE);

}


void CPersonsView::SetColumns(CListCtrl& oListCtrl)
{
	oListCtrl.InsertColumn(PersonsViewColumnFirstName, PERSON_FIRST_NAME_COL-
UMN_NAME, LVCFMT_LEFT, PERSON_FIRST_NAME_COLUMN_WIDTH);
	oListCtrl.InsertColumn(PersonsViewColumnMiddleName,
PERSON_MIDDLE_NAME_COLUMN_NAME, LVCFMT_LEFT, PERSON_MIDDLE_NAME_COLUMN_WIDTH);
	oListCtrl.InsertColumn(PersonsViewColumnLastName,
PERSON_LAST_NAME_COLUMN_NAME, LVCFMT_LEFT, PERSON_LAST_NAME_COLUMN_WIDTH);
	oListCtrl.InsertColumn(PersonsViewColumnUCN,	PERSON_UCN_COLUMN_NAME,
LVCFMT_LEFT, PERSON_UCN_COLUMN_WIDTH);
	oListCtrl.InsertColumn(PersonsViewColumnCityName,
PERSON_CITY_NAME_COLUMN_NAME, LVCFMT_LEFT, PERSON_CITY_NAME_COLUMN_WIDTH);
	oListCtrl.InsertColumn(PersonsViewColumnDistrict,
PERSON_DISTRICT_COLUMN_NAME, LVCFMT_LEFT, PERSON_DISTRICT_COLUMN_WIDTH);
	oListCtrl.InsertColumn(PersonsViewColumnAddress,
PERSON_ADDRESS_COLUMN_NAME, LVCFMT_LEFT, PERSON_ADDRESS_COLUMN_WIDTH);
	oListCtrl.InsertColumn(PersonsViewColumnCompany,
PERSON_COMPANY_COLUMN_NAME, LVCFMT_LEFT, PERSON_COMPANY_COLUMN_WIDTH);
	oListCtrl.InsertColumn(PersonsViewColumnPosition,
PERSON_POSITION_COLUMN_NAME, LVCFMT_LEFT, PERSON_POSITION_COLUMN_WIDTH);

}


void CPersonsView::SetInitialData(CListCtrl& oListCtrl)
{
	const CPtrAutoArray<CPerson>* pPersonsArray = GetDocument()->GetData();
	if (pPersonsArray == NULL)
	{
		CErrorLogger::LogMessage(_T("Error setting inital data"), TRUE,
TRUE);
		return;
	}

	for (INT_PTR i = 0; i < pPersonsArray->GetCount(); i++)
	{
		CPerson* pPerson = pPersonsArray->GetAt(i);

		OperationInsert(oListCtrl, *pPerson);
	}

}

void CPersonsView::ShowSearch(CListCtrl& oListCtrl)
{
	if (!m_bIsSearch)
		return;

	SetRedraw(FALSE);
	oListCtrl.DeleteAllItems();
	SetRedraw(TRUE);

	GetDocument()->SelectByNameUCNAddress(m_oAutoArray, m_strSearchName,
m_strSearchUCN, m_strSearchAddress);
	for (INT_PTR i = 0; i < m_oAutoArray.GetCount(); i++)
	{
		CPerson* pPerson = m_oAutoArray.GetAt(i);

		OperationInsert(oListCtrl, *pPerson);
	}
}


void CPersonsView::OperationUpdate(CListCtrl& oListCtrl, const CPerson& oPerson)
{
	const int nIndex = GetIndexByID(oListCtrl ,oPerson.m_recPerson.lID);

	if (nIndex == INDEX_BY_ID_ERROR)
		return;

	CPersonDisplay* pPersonDisplay = GetDocument()->GetDisplayInformation();
	if (pPersonDisplay == NULL)
	{
		CErrorLogger::LogMessage(UPDATE_ERROR_MESSAGE, TRUE, TRUE);
		return;
	}

	SetListViewItem(oListCtrl, oPerson, *pPersonDisplay, nIndex);
}


void CPersonsView::OperationInsert(CListCtrl& oListCtrl, const CPerson& oPerson)
{
	const int nIndex = oListCtrl.InsertItem(LVIF_PARAM, 0, oPerson.m_recPer-
son.szFirstName, 0, 0, 0, oPerson.m_recPerson.lID);

	CPersonDisplay* pPersonDisplay = GetDocument()->GetDisplayInformation();
	if (pPersonDisplay == NULL)
	{
		CErrorLogger::LogMessage(INSERT_ERROR_MESSAGE, TRUE, TRUE);
		return;
	}
	SetListViewItem(oListCtrl, oPerson, *pPersonDisplay, nIndex);
}

void CPersonsView::OperationDelete(CListCtrl& oListCtrl, const CPerson& oPerson)
{
	const int nIndex = GetIndexByID(oListCtrl ,oPerson.m_recPerson.lID);
	if (!oListCtrl.DeleteItem(nIndex))
		CErrorLogger::LogMessage(REMOVE_ITEM_ERROR_MESSAGE, TRUE, TRUE);
}

void CPersonsView::OnUpdate(CView* pSender, LPARAM lHint, CObject* pHint)
{
	CListCtrl& oListCtrl = GetListCtrl();

	switch ((DocumentDataOperation)lHint)
	{
	case DocumentDataOperationUpdate:
		OperationUpdate(oListCtrl, *(CPerson*)pHint);
		break;
	case DocumentDataOperationInsert:
		OperationInsert(oListCtrl, *(CPerson*)pHint);
		break;
	case DocumentDataOperationDelete:
		OperationDelete(oListCtrl, *(CPerson*)pHint);
		break;
	default:
		CErrorLogger::LogMessage(OPERATION_NOT_RECOGNISED_ERROR_MESSAGE,
TRUE, TRUE);
		break;
	}
	if (m_bIsSearch)
		ShowSearch(oListCtrl);
	oListCtrl.RedrawItems(0, ALL_ITEMS_NUMBER);
}

void CPersonsView::OnInitialUpdate()
{
	//CListView::OnInitialUpdate();

	CListCtrl& oListCtrl = GetListCtrl();
	Initialize(oListCtrl);
}


// CCitiesView diagnostics

#ifdef _DEBUG
void CPersonsView::AssertValid() const
{
	__super::AssertValid();
}

void CPersonsView::Dump(CDumpContext& dc) const
{
	__super::Dump(dc);
}

CPersonsDoc* CPersonsView::GetDocument() const // non-debug version is inline
{
	ASSERT(m_pDocument->IsKindOf(RUNTIME_CLASS(CPersonsDoc)));
	return (CPersonsDoc*)m_pDocument;
}
#endif //_DEBUG
```

```
// CCitiesView message handlers


void CPersonsView::OnContextEdit()
{
	CPerson* pSelectedPerson = GetSelectedPerson();
	if (pSelectedPerson == NULL)
		return;

	CPersonsDialog oPersonsDialog(pSelectedPerson, *GetDocument()->GetDisplay-
Information() , PersonsDialogTypeEdit);
	if (oPersonsDialog.DoModal() != IDOK)
		return;

	CPerson* pPerson = oPersonsDialog.GetPerson();

	const  BOOL  bResult  =  GetDocument()->SetPersonByID(pPerson->m_recPerson-
.lID, *(pPerson));
	if (!bResult)
		CErrorLogger::LogMessage(UPDATE_ERROR_MESSAGE, TRUE, TRUE);
}


void CPersonsView::OnContextAdd()
{
	CPersonsDialog   oPersonsDialog(NULL,   *GetDocument()->GetDisplayInforma-
tion(), PersonsDialogTypeAdd);
	if (oPersonsDialog.DoModal() != IDOK)
		return;

	if (!GetDocument()->AddPerson(*oPersonsDialog.GetPerson()))
		CErrorLogger::LogMessage(INSERT_ERROR_MESSAGE, TRUE, TRUE);

}


void CPersonsView::OnContextDelete()
{
	const CPerson* pSelectedPerson = GetSelectedPerson();
	if (pSelectedPerson == NULL)
		return;

	const  int  nResult  =  AfxMessageBox(DELETE_CONFIRM_MESSAGE,  MB_YESNO,
MB_ICONINFORMATION);

	if (nResult != IDYES)
		return;

	if (!GetDocument()->RemovePerson(pSelectedPerson->m_recPerson.lID))
		CErrorLogger::LogMessage(DELETE_ERROR_MESSAGE, FALSE, TRUE);

}


void CPersonsView::OnContextView()
{
	CPerson* pSelectedPerson = GetSelectedPerson();
	if (pSelectedPerson == NULL)
		return;

	CPersonDisplay* oPersonDisplay = GetDocument()->GetDisplayInformation();
	CPersonsDialog oPersonsDialog(pSelectedPerson, *oPersonDisplay, PersonsDi-
alogTypeView);
	oPersonsDialog.DoModal();
}

void CPersonsView::OnContextSearch()
{
	CListCtrl& oListCtrl = GetListCtrl();
	CPersonsSearchDialog oDialog;
	if (oDialog.DoModal() != IDOK)
	{
		SetRedraw(FALSE);
		oListCtrl.DeleteAllItems();
		SetRedraw(TRUE);

		m_bIsSearch = FALSE;
		SetInitialData(oListCtrl);
		return;
	}
	m_bIsSearch = TRUE;
	m_strSearchName = oDialog.m_strName;
	m_strSearchUCN = oDialog.m_strUCN;
	m_strSearchAddress = oDialog.m_strAddress;
	ShowSearch(oListCtrl);
}


void CPersonsView::OnNMCustomdraw(NMHDR* pNMHDR, LRESULT* pResult)
{
	SetRowColors(pNMHDR, pResult);
}


BOOL CPersonsView::CanSearch()
{
	return TRUE;
}

BOOL CPersonsView::CanConvert()
{
	return TRUE;
}

void CPersonsView::OnContextConvert()
{
	CFileDialog oFileDialog(FALSE, NULL, NULL, OFN_OVERWRITEPROMPT, _T("HTML
Files (*.html)|*.html|All Files (*.*)|*.*||"));; // TRUE for open dialog, FALSE
for save dialog
	if (oFileDialog.DoModal() != IDOK) // display the dialog box and check for
cancel
		return;


	CString strFilePath = oFileDialog.GetPathName();
	CPersonsFileConverter oPersonsFileConverter;


	if (m_bIsSearch) {
		if (!oPersonsFileConverter.Convert(strFilePath, m_oAutoArray, *Get-
Document()->GetDisplayInformation()))
			CErrorLogger::LogMessage(_T("Error when converting file."),
TRUE, TRUE);
	}
	else {
			if			(!oPersonsFileConverter.Convert(strFilePath,
*(CPtrAutoArray<CPerson>*)GetDocument()->GetData(),  *GetDocument()->GetDisplay-
Information()))
				CErrorLogger::LogMessage(_T("Error  when  converting  file."),
TRUE, TRUE);
		}
}
```

PersonsDialog.h

```
#pragma once
#include "Person.h"
#include "PersonDisplay.h"
#include "PersonsDialogType.h"

//////////////////////////////////////////////////////////////////////////////
// CPersonsDialog

/// <summary>
/// Диалог за редактиране, преглеждане и добавяне на PERSONS
/// </summary>
class CPersonsDialog : public CDialogEx
{
	DECLARE_DYNAMIC(CPersonsDialog)

	// Constants
	// ----------------
private:
	/// <summary> Тип на диалога </summary>
	const PersonsDialogType m_ePersonsDialogType;

	// Constructor / Destructor
	// ----------------
public:
	CPersonsDialog(CPerson*  pPerson,  CPersonDisplay&  oPersonDisplay,  const
PersonsDialogType ePersonsDialogType, CWnd* pParent = nullptr);    // standard
constructor
	virtual ~CPersonsDialog();

	/// <summary> Достъп до новополучения запис </summary>
	CPerson* GetPerson();

	// Methods
	// ----------------
private:
	/// <summary> Задаване полетата да са само за четене </summary>
	void SetReadOnly();

	/// <summary> Задаване на колоните </summary>
	void SetColumns();

	/// <summary> Задаване на първоначлните данни </summary>
	void SetInitalData();

	/// <summary> Намира индекса на записа в масива по зададено ID </summary>
	INT_PTR FindArrayIndexByID(const long lID);

	/// <summary> Намира индекса на записа в CListCtrl по зададено ID </sum-
mary>
	int FindListIndexByID(const long lID);
	/// <summary> Намира селектирания запис </summary>
	const PHONE_NUMBERS* GetSelectedPhoneNumber();

	/// <summary> Редактиране на PHONE_NUMBERS </summary>
	BOOL UpdateItem(const PHONE_NUMBERS& recPhoneNumber, const int nIndexAr-
ray);

	/// <summary> Задава ред от CListCtrl </summary>
	BOOL SetItemText(const PHONE_NUMBERS& recPhoneNumber, const int nIn-
dexList);

	/// <summary> Добавя ред от CListCtrl </summary>
	BOOL AddItem(const PHONE_NUMBERS& recPhoneNumber);

	/// <summary> Проверява дали са зададени стойностите на контролите </sum-
mary>
	BOOL ValidateUserInput() const;

	/// <summary> Задава заглавието на диалога </summary>
	BOOL SetDialogName();

	/// <summary> Задава стойностите на полетата </summary>
	BOOL SetDialogFields();

	/// <summary> Представя CITIES като низ </summary>
	CString CityToString(const CITIES& recCity) const;

	/// <summary> Задава CITIES в комбо-бокс </summary>
	void SetCitiesComboBox();

	/// <summary> Задава COMPANIES в комбо-бокс </summary>
	void SetCompaniesComboBox();

	/// <summary> Задава POSITIONS в комбо-бокс </summary>
	void SetPositionsComboBox();


public:
	virtual BOOL OnInitDialog();
	afx_msg void OnContextMenu(CWnd* /*pWnd*/, CPoint /*point*/);
	afx_msg void OnRButtonUp(UINT nFlags, CPoint point);
	afx_msg void OnPopupAdd();
	afx_msg void OnPopupDelete();
	afx_msg void OnPopupEdit();
	afx_msg void OnPopupView();
	virtual void OnOK();
	virtual void OnCancel();


// Dialog Data
#ifdef AFX_DESIGN_TIME
	enum { IDD = IDD_PERSONS_DIALOG };
#endif

protected:
	virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support

	DECLARE_MESSAGE_MAP()
private:
	/// <summary> Указател към човека, когото редактираме. Има стойност NULL
при добавяне </summary>
	CPerson* m_pPerson;
	/// <summary> Указател към допълнителна информация за представяне </sum-
mary>
```

```
	CPersonDisplay* m_pPersonDisplay;
	/// <summary> Новополучения запис </summary>
	CPerson m_oNewPerson;
	/// <summary> Временно ID на телефонння номер </summary>
	long m_lTemporaryID;
public:
	//Полетата на диалога:
	CEdit m_edbFirstName;
	CEdit m_edbMiddleName;
	CEdit m_edbLastName;
	CEdit m_edbUCN;
	CComboBox m_cmbCity;
	CEdit m_edbAddress;
	CListCtrl m_lscPhoneNumbers;
	CComboBox m_cmbCompany;
	CComboBox m_cmbPosition;
};
```

PersonsDialog.cpp

```
// PersonsDialog.cpp : implementation file
//

#include "pch.h"
#include "Phonebook.h"
#include "PersonsDialog.h"
#include "afxdialogex.h"
#include "ErrorLogger.h"
#include "PhoneNumbersEditDialog.h"
#include "InputValidator.h"



/// <summary>
/// enum отговарящ на колона от PHONE_NUMBERS ListView
/// </summary>
enum PhoneNumbersColumn
{
	PhoneNumbersColumnPhoneType = 0,
	PhoneNumbersColumnPhoneNumber
};


/////////////////////////////////////////////////////////////////////////////
// CPersonsDialog

	// Constants
	// ----------------
#define DIALOG_ADD_NAME _T("Добавяне")
#define DIALOG_EDIT_NAME _T("Редактиране")
#define DIALOG_VIEW_NAME _T("Преглед")

#define PHONE_NUMBERS_TYPE_COLUMN_WIDTH 90
#define PHONE_NUMBERS_NUMBER_COLUMN_WIDTH 120

#define PHONE_NUMBERS_TYPE_COLUMN_NAME _T("Тип")
#define PHONE_NUMBERS_NUMBER_COLUMN_NAME _T("Номер")


#define NAME_MIN_FIELD_LENGTH 2
#define ADDRESS_MIN_FIELD_LENGTH 4
#define UCN_MIN_FIELD_LENGTH 10
```

```
#define TEMPORARY_ID_INITIAL_VALUE -1
#define INDEX_NOT_FOUND -1

#define SETTING_ITEM_TEXT_ERROR_MESSAGE _T("Грешка при въвеждането на текста във ListCtrl.")
#define CONFIRM_DELETE_MESSAGE _T("Сигурни ли сте, че искате да изтриете селектирания запис?")
#define VALIDATE_FIRST_NAME_MESSAGE _T("Моля въведете име.")
#define VALIDATE_MIDDLE_NAME_MESSAGE _T("Моля въведете презиме")
#define VALIDATE_LAST_NAME_MESSAGE _T("Моля въведете фамилия")
#define VALIDATE_UCN_MESSAGE _T("Моля въведете правилно ЕГН (10 цифри).")
#define VALIDATE_ADDRESS_MESSAGE _T("Моля въведете адрес")
#define VALIDATE_CITY_MESSAGE _T("Моля въведете град")
#define VALIDATE_COMPANY_MESSAGE _T("Моля въведете компания")
#define VALIDATE_POSITION_MESSAGE _T("Моля въведете позиция")


IMPLEMENT_DYNAMIC(CPersonsDialog, CDialogEx)


// Constructor / Destructor
// ----------------

CPersonsDialog::CPersonsDialog(CPerson* pPerson, CPersonDisplay& oPersonDisplay, const PersonsDialogType ePersonsDialogType, CWnd* pParent /*=nullptr*/)
	: CDialogEx(IDD_PERSONS_DIALOG, pParent),
	m_pPerson(pPerson),
	m_pPersonDisplay(&oPersonDisplay),
	m_ePersonsDialogType(ePersonsDialogType),
	m_lTemporaryID(TEMPORARY_ID_INITIAL_VALUE)
{
	if (pPerson == NULL)
		return;

	m_oNewPerson.m_oPhoneNumbers = pPerson->m_oPhoneNumbers;
}

CPersonsDialog::~CPersonsDialog()
{
}

// Methods
// ----------------
BOOL CPersonsDialog::ValidateUserInput() const
{
	CInputValidator oInputValidator;

	if (!oInputValidator.ValidateTextField(m_edbFirstName, PERSONS_NAME_LENGTH, NAME_MIN_FIELD_LENGTH))
	{
		AfxMessageBox(VALIDATE_FIRST_NAME_MESSAGE, MB_OK, MB_ICONERROR);
		return FALSE;
	}

	if(!oInputValidator.ValidateTextField(m_edbMiddleName, PERSONS_NAME_LENGTH, NAME_MIN_FIELD_LENGTH))
	{
		AfxMessageBox(VALIDATE_MIDDLE_NAME_MESSAGE, MB_OK, MB_ICONERROR);
		return FALSE;
	}

	if(!oInputValidator.ValidateTextField(m_edbLastName,  PERSONS_NAME_LENGTH, NAME_MIN_FIELD_LENGTH))
	{
		AfxMessageBox(VALIDATE_LAST_NAME_MESSAGE, MB_OK, MB_ICONERROR);
		return FALSE;
	}

	if(!oInputValidator.ValidateTextField(m_edbAddress,
PERSONS_ADDRESS_LENGTH, ADDRESS_MIN_FIELD_LENGTH))
	{
		AfxMessageBox(VALIDATE_ADDRESS_MESSAGE, MB_OK, MB_ICONERROR);
		return FALSE;
	}

	if(!oInputValidator.ValidateNumber(m_edbUCN,         UCN_MIN_FIELD_LENGTH,
UCN_MIN_FIELD_LENGTH))
	{
		AfxMessageBox(VALIDATE_UCN_MESSAGE, MB_OK, MB_ICONERROR);
		return FALSE;
	}

	if(!oInputValidator.ValidateComboBox(m_cmbCity))
	{
		AfxMessageBox(VALIDATE_CITY_MESSAGE, MB_OK, MB_ICONERROR);
		return FALSE;
	}

	if (!oInputValidator.ValidateComboBox(m_cmbCompany))
	{
		AfxMessageBox(VALIDATE_COMPANY_MESSAGE, MB_OK, MB_ICONERROR);
		return FALSE;
	}

	if (!oInputValidator.ValidateComboBox(m_cmbPosition))
	{
		AfxMessageBox(VALIDATE_POSITION_MESSAGE, MB_OK, MB_ICONERROR);
		return FALSE;
	}


	return TRUE;
}


CPerson* CPersonsDialog::GetPerson()
{
	return &m_oNewPerson;
}


BOOL CPersonsDialog::SetDialogName()
{
	switch (m_ePersonsDialogType)
	{
	case PersonsDialogTypeAdd:
		SetWindowText(DIALOG_ADD_NAME);
		break;
	case PersonsDialogTypeEdit:
		SetWindowText(DIALOG_EDIT_NAME);
		break;
	case PersonsDialogTypeView:
		SetWindowText(DIALOG_VIEW_NAME);
		break;
	default:
		return FALSE;
		break;
	}
	return TRUE;
}

CString CPersonsDialog::CityToString(const CITIES& recCity) const
{
	CString strCity = recCity.szCityName;
	strCity += _T(" - ");
	strCity += recCity.szDistrict;
	return strCity;
}


void CPersonsDialog::SetCitiesComboBox()
{
	CPtrAutoArray<CITIES>* oCitiesArray = m_pPersonDisplay->GetCities();

	for (INT_PTR i = 0; i < oCitiesArray->GetCount(); i++)
	{
		CITIES* pCurrentCity = oCitiesArray->GetAt(i);

		int  nIndex  =  m_cmbCity.AddString(CityToString(*pCurrentCity).Get-
String());
		m_cmbCity.SetItemData(nIndex, (DWORD_PTR)pCurrentCity->lID);

		if (m_pPerson != NULL && pCurrentCity->lID == m_pPerson->m_recPer-
son.lCityID)
			m_cmbCity.SetCurSel(nIndex);
	}

}

void CPersonsDialog::SetCompaniesComboBox()
{
	CPtrAutoArray<COMPANIES>*  oCompaniesArray  =  m_pPersonDisplay->GetCompa-
nies();

	for (INT_PTR i = 0; i < oCompaniesArray->GetCount(); i++)
	{
		COMPANIES* pCurrent = oCompaniesArray->GetAt(i);

		int nIndex = m_cmbCompany.AddString(pCurrent->szCompanyName);
		m_cmbCompany.SetItemData(nIndex, (DWORD_PTR)pCurrent->lID);

		if (m_pPerson != NULL && pCurrent->lID == m_pPerson->m_recPerson.l-
CompanyID)
			m_cmbCompany.SetCurSel(nIndex);
	}

}

void CPersonsDialog::SetPositionsComboBox()
{
	CPtrAutoArray<POSITIONS>*  oPositionsArray  =  m_pPersonDisplay->GetPosi-
tions();

	for (INT_PTR i = 0; i < oPositionsArray->GetCount(); i++)
	{
		POSITIONS* pCurrent = oPositionsArray->GetAt(i);

		int nIndex = m_cmbPosition.AddString(pCurrent->szPositionName);
		m_cmbPosition.SetItemData(nIndex, (DWORD_PTR)pCurrent->lID);

		if (m_pPerson != NULL && pCurrent->lID == m_pPerson->m_recPerson.l-
PositionID)
```

```
                m_cmbPosition.SetCurSel(nIndex);
        }


}


BOOL CPersonsDialog::SetDialogFields()
{
        switch (m_ePersonsDialogType)
        {
        case PersonsDialogTypeAdd:
                break;
        case PersonsDialogTypeEdit:
        case PersonsDialogTypeView:
                m_edbFirstName.SetWindowTextW(m_pPerson->m_recPerson.szFirstName);
                m_edbMiddleName.SetWindowTextW(m_pPerson->m_recPerson.szMiddleName);
                m_edbLastName.SetWindowTextW(m_pPerson->m_recPerson.szLastName);
                m_edbUCN.SetWindowTextW(m_pPerson->m_recPerson.szUCN);
                m_edbAddress.SetWindowTextW(m_pPerson->m_recPerson.szAddress);
                break;
        default:
                return FALSE;
                break;
        }

        return TRUE;
}

INT_PTR CPersonsDialog::FindArrayIndexByID(const long lID)
{
        for (INT_PTR i = 0; i < m_oNewPerson.m_oPhoneNumbers.GetCount(); i++)
        {
                if (m_oNewPerson.m_oPhoneNumbers.GetAt(i)->lID != lID)
                        continue;
                return i;
        }

        return INDEX_NOT_FOUND;
}

int CPersonsDialog::FindListIndexByID(const long lID)
{
        LVFINDINFOW oFindInfo;
        oFindInfo.flags = LVFI_PARAM;
        oFindInfo.lParam = lID;

        return m_lscPhoneNumbers.FindItem(&oFindInfo);
}


BOOL CPersonsDialog::SetItemText(const PHONE_NUMBERS& recPhoneNumber, const int
nIndexList)
{
        PHONE_TYPES recPhoneType;
        if (!m_pPersonDisplay->GetPhoneTypeByID(recPhoneNumber.lPhoneTypeID, rec-
PhoneType))
                return FALSE;

        if (!m_lscPhoneNumbers.SetItemText(nIndexList, PhoneNumbersColumnPhone-
Type, recPhoneType.szType))
                return FALSE;
```

```
	if (!m_lscPhoneNumbers.SetItemText(nIndexList, PhoneNumbersColumnPhoneNum-
ber, recPhoneNumber.szPhoneNumber))
		return FALSE;

	return TRUE;
}


BOOL CPersonsDialog::AddItem(const PHONE_NUMBERS& recPhoneNumber)
{
	PHONE_TYPES recPhoneType;
	if (!m_pPersonDisplay->GetPhoneTypeByID(recPhoneNumber.lPhoneTypeID, rec-
PhoneType))
		return FALSE;

	const int nIndex = m_lscPhoneNumbers.InsertItem(LVIF_PARAM, 0, recPhone-
Type.szType, 0, 0, 0, recPhoneNumber.lID);


	return SetItemText(recPhoneNumber, nIndex);
}


void CPersonsDialog::SetColumns()
{
	m_lscPhoneNumbers.InsertColumn(PhoneNumbersColumnPhoneType,
PHONE_NUMBERS_TYPE_COLUMN_NAME, LVCFMT_LEFT, PHONE_NUMBERS_TYPE_COLUMN_WIDTH);
	m_lscPhoneNumbers.InsertColumn(PhoneNumbersColumnPhoneNumber,   PHONE_NUM-
BERS_NUMBER_COLUMN_NAME, LVCFMT_LEFT, PHONE_NUMBERS_NUMBER_COLUMN_WIDTH);
	m_lscPhoneNumbers.SetExtendedStyle(m_lscPhoneNumbers.GetExtendedStyle()   |
LVS_EX_FULLROWSELECT | LVS_EX_GRIDLINES);
}


void CPersonsDialog::SetInitalData()
{
	for (INT_PTR i = 0; i < m_oNewPerson.m_oPhoneNumbers.GetCount(); i++)
	{
		if (!AddItem(*m_oNewPerson.m_oPhoneNumbers.GetAt(i)))
			CErrorLogger::LogMessage(_T("Error adding item to PHONE_NUM-
BERS ListCtrl"), TRUE, TRUE);
	}
}


BOOL CPersonsDialog::UpdateItem(const PHONE_NUMBERS& recPhoneNumber, const int
nIndexArray)
{
	if (nIndexArray < 0 || nIndexArray >= m_oNewPerson.m_oPhoneNumbers.Get-
Count())
		return FALSE;

	*m_oNewPerson.m_oPhoneNumbers.GetAt(nIndexArray) = recPhoneNumber;
	int nIndexList = FindListIndexByID(m_oNewPerson.m_oPhoneNumbers.GetAt(nIn-
dexArray)->lID);

	return SetItemText(recPhoneNumber, nIndexList);
}


const PHONE_NUMBERS* CPersonsDialog::GetSelectedPhoneNumber()
{
	POSITION oPosition = m_lscPhoneNumbers.GetFirstSelectedItemPosition();
	if (oPosition == NULL)
```

```
            return NULL;

      const int nItem = m_lscPhoneNumbers.GetNextSelectedItem(oPosition);
      const long lID = m_lscPhoneNumbers.GetItemData(nItem);

      for (INT_PTR i = 0; i < m_oNewPerson.m_oPhoneNumbers.GetCount(); i++)
      {
            if (m_oNewPerson.m_oPhoneNumbers.GetAt(i)->lID == lID)
                  return m_oNewPerson.m_oPhoneNumbers.GetAt(i);
      }

      return NULL;

}

void CPersonsDialog::SetReadOnly()
{
      m_edbFirstName.SetReadOnly(TRUE);
      m_edbMiddleName.SetReadOnly(TRUE);
      m_edbLastName.SetReadOnly(TRUE);
      m_edbUCN.SetReadOnly(TRUE);
      m_edbAddress.SetReadOnly(TRUE);
}


void CPersonsDialog::DoDataExchange(CDataExchange* pDX)
{
      CDialogEx::DoDataExchange(pDX);
      DDX_Control(pDX, IDC_EDB_PERSONS_FIRST_NAME, m_edbFirstName);
      DDX_Control(pDX, IDC_EDB_PERSONS_MIDDLE_NAME, m_edbMiddleName);
      DDX_Control(pDX, IDC_EDB_PERSONS_LAST_NAME, m_edbLastName);
      DDX_Control(pDX, IDC_EDB_PERSONS_UCN, m_edbUCN);
      DDX_Control(pDX, IDC_CMB_PERSONS_CITY, m_cmbCity);
      DDX_Control(pDX, IDC_EDB_PERSONS_ADDRESS, m_edbAddress);
      DDX_Control(pDX, IDC_LSC_PERSONS_PHONE_NUMBERS, m_lscPhoneNumbers);
      DDX_Control(pDX, IDC_COMBO1, m_cmbCompany);
      DDX_Control(pDX, IDC_COMBO2, m_cmbPosition);
}


BEGIN_MESSAGE_MAP(CPersonsDialog, CDialogEx)
      //ON_BN_CLICKED(IDC_BTN_PERSONS_PHONE_NUMBERS,
&CPersonsDialog::OnBnClickedBtnPersonsPhoneNumbers)
      ON_WM_CONTEXTMENU()
      ON_WM_RBUTTONUP()
      ON_COMMAND(ID_POPUP_ADD, &CPersonsDialog::OnPopupAdd)
      ON_COMMAND(ID_POPUP_DELETE, &CPersonsDialog::OnPopupDelete)
      ON_COMMAND(ID_POPUP_EDIT, &CPersonsDialog::OnPopupEdit)
      ON_COMMAND(ID_POPUP_VIEW, &CPersonsDialog::OnPopupView)

END_MESSAGE_MAP()


// CPersonsDialog message handlers


BOOL CPersonsDialog::OnInitDialog()
{
      CDialogEx::OnInitDialog();

      SetDialogName();
      SetCitiesComboBox();
      SetCompaniesComboBox();
      SetPositionsComboBox();
```

```
	SetColumns();
	if (m_pPerson == NULL)
		return TRUE;

	SetDialogFields();
	SetInitalData();

	if (m_ePersonsDialogType != PersonsDialogTypeView)
		return TRUE;

	SetReadOnly();

	return TRUE;  // return TRUE unless you set the focus to a control
	              // EXCEPTION: OCX Property Pages should return FALSE
}


void CPersonsDialog::OnOK()
{
	if (m_ePersonsDialogType == PersonsDialogTypeView)
	{
		CDialogEx::OnOK();
		return;
	}

	if (!ValidateUserInput())
		return;

	if (m_ePersonsDialogType == PersonsDialogTypeEdit && m_pPerson != NULL)
		m_oNewPerson.m_recPerson = m_pPerson->m_recPerson;

	m_edbFirstName.GetWindowTextW(m_oNewPerson.m_recPerson.szFirstName,   PER-
SONS_NAME_LENGTH);
	m_edbMiddleName.GetWindowTextW(m_oNewPerson.m_recPerson.szMiddleName, PER-
SONS_NAME_LENGTH);
	m_edbLastName.GetWindowTextW(m_oNewPerson.m_recPerson.szLastName,
PERSONS_NAME_LENGTH);
	m_edbUCN.GetWindowTextW(m_oNewPerson.m_recPerson.szUCN,
PERSONS_UCN_LENGTH);
	m_edbAddress.GetWindowTextW(m_oNewPerson.m_recPerson.szAddress,
PERSONS_ADDRESS_LENGTH);
	m_oNewPerson.m_recPerson.lCityID     =     m_cmbCity.GetItemData(m_cmbCi-
ty.GetCurSel());
	m_oNewPerson.m_recPerson.lCompanyID = m_cmbCompany.GetItemData(m_cmbCompa-
ny.GetCurSel());
	m_oNewPerson.m_recPerson.lPositionID  =  m_cmbPosition.GetItemData(m_cmbPo-
sition.GetCurSel());


	CDialogEx::OnOK();
}


void CPersonsDialog::OnCancel()
{
	// TODO: Add your specialized code here and/or call the base class

	CDialogEx::OnCancel();
}


void CPersonsDialog::OnContextMenu(CWnd* /*pWnd*/, CPoint point)
{
	// TODO: Add your message handler code here
```

```
#ifndef SHARED_HANDLERS
      CMenu oContextMenu;
      oContextMenu.LoadMenu(IDR_POPUP);
      CMenu* oPopupMenu = oContextMenu.GetSubMenu(0);

      if (m_ePersonsDialogType == PersonsDialogTypeView)
      {
            oPopupMenu->EnableMenuItem(ID_POPUP_ADD, MF_BYCOMMAND | MF_DISABLED
| MF_GRAYED);
            oPopupMenu->EnableMenuItem(ID_POPUP_DELETE, MF_BYCOMMAND | MF_DIS-
ABLED | MF_GRAYED);
            oPopupMenu->EnableMenuItem(ID_POPUP_EDIT, MF_BYCOMMAND | MF_DISABLED
| MF_GRAYED);
            oPopupMenu->EnableMenuItem(ID_POPUP_VIEW, MF_BYCOMMAND | MF_EN-
ABLED);
      }

      if (GetSelectedPhoneNumber() == NULL)
      {
            oPopupMenu->EnableMenuItem(ID_POPUP_DELETE, MF_BYCOMMAND | MF_DIS-
ABLED | MF_GRAYED);
            oPopupMenu->EnableMenuItem(ID_POPUP_EDIT, MF_BYCOMMAND | MF_DISABLED
| MF_GRAYED);
            oPopupMenu->EnableMenuItem(ID_POPUP_VIEW, MF_BYCOMMAND | MF_DISABLED
| MF_GRAYED);
      }

      oPopupMenu->TrackPopupMenu(TPM_LEFTALIGN | TPM_RIGHTBUTTON, point.x,
point.y, this);

      //theApp.GetContextMenuManager()->ShowPopupMenu(IDR_POPUP, point.x,
point.y, this, TRUE);
#endif
}


void CPersonsDialog::OnRButtonUp(UINT nFlags, CPoint point)
{
      // TODO: Add your message handler code here and/or call default
      ClientToScreen(&point);
      OnContextMenu(this, point);

      CDialogEx::OnRButtonUp(nFlags, point);
}


void CPersonsDialog::OnPopupAdd()
{
      CPhoneNumbersEditDialog oPhoneNumbersEditDialog(NULL, PhoneNumbersEditDi-
alogTypeAdd, *m_pPersonDisplay);
      if (oPhoneNumbersEditDialog.DoModal() != IDOK)
            return;

      PHONE_NUMBERS* pPhoneNumber = new PHONE_NUMBERS;
      oPhoneNumbersEditDialog.GetPhoneNumber(*pPhoneNumber);
      pPhoneNumber->lID = m_lTemporaryID--;

      m_oNewPerson.m_oPhoneNumbers.Add(pPhoneNumber);
      if (!AddItem(*pPhoneNumber))
            CErrorLogger::LogMessage(_T("Error adding item to ListCtrl."), TRUE,
TRUE);
}
```

```
void CPersonsDialog::OnPopupDelete()
{

      const PHONE_NUMBERS* pSelectedPhoneNumber = GetSelectedPhoneNumber();
      if (pSelectedPhoneNumber == NULL)
            return;

      if (AfxMessageBox(CONFIRM_DELETE_MESSAGE, MB_OKCANCEL, MB_ICONINFORMATION)
!= IDOK)
            return;

      INT_PTR nIndex = FindArrayIndexByID(pSelectedPhoneNumber->lID);
      if (nIndex == INDEX_NOT_FOUND)
      {
            CErrorLogger::LogMessage(_T("Error index not found."), TRUE, TRUE);
            return;
      }

      if   (!m_lscPhoneNumbers.DeleteItem(FindListIndexByID(pSelectedPhoneNumber-
>lID)))
      {
            CErrorLogger::LogMessage(_T("Error  removing  item  from  ListCtrl."),
TRUE, TRUE);
            return;
      }

      m_oNewPerson.m_oPhoneNumbers.RemoveAt(nIndex);
}


void CPersonsDialog::OnPopupEdit()
{
      // TODO: Add your command handler code here
      const PHONE_NUMBERS* pSelectedPhoneNumber = GetSelectedPhoneNumber();
      if (pSelectedPhoneNumber == NULL)
            return;

      CPhoneNumbersEditDialog oPhoneNumbersEditDialog(pSelectedPhoneNumber, Pho-
neNumbersEditDialogTypeEdit, *m_pPersonDisplay);
      if (oPhoneNumbersEditDialog.DoModal() != IDOK)
            return;

      const INT_PTR nIndex = FindArrayIndexByID(pSelectedPhoneNumber->lID);
      if (nIndex == INDEX_NOT_FOUND)
      {
            CErrorLogger::LogMessage(_T("Error phone number not found."), TRUE,
TRUE);
            return;
      }

      oPhoneNumbersEditDialog.GetPhoneNumber(*m_oNewPerson.m_oPhoneNumber-
s.GetAt(nIndex));

      SetItemText(*m_oNewPerson.m_oPhoneNumbers.GetAt(nIndex),        FindListIn-
dexByID(pSelectedPhoneNumber->lID));
}


void CPersonsDialog::OnPopupView()
{
      // TODO: Add your command handler code here
      const PHONE_NUMBERS* pSelectedPhoneNumber = GetSelectedPhoneNumber();
      if (pSelectedPhoneNumber == NULL)
            return;
```

```
	CPhoneNumbersEditDialog oPhoneNumbersEditDialog(pSelectedPhoneNumber, Pho-
neNumbersEditDialogTypeView, *m_pPersonDisplay);
	oPhoneNumbersEditDialog.DoModal();
}
```