|  |
| --- |
| Иркутская областная государственная  универсальная научная библиотека  им. И. И. Молчанова-Сибирского |
| **Библиотека доступа к ресурсам ИРБИС64 на платформе .Net**  **версия 1.3.0.24**  от «13» февраля 2014 г.  **описание для программиста** |
| **Иркутск 2014** |

Оглавление

[Введение 4](#_Toc380071147)

[Совместимость 4](#_Toc380071148)

[Инструментарий 4](#_Toc380071149)

[Системные требования 5](#_Toc380071150)

[Сайт 5](#_Toc380071151)

[Лицензия 6](#_Toc380071152)

[Благодарности 6](#_Toc380071153)

[Версия и совместимость 6](#_Toc380071154)

[Хроника версий 7](#_Toc380071155)

[1.0.0.1 7](#_Toc380071156)

[1.1.0.12 7](#_Toc380071157)

[1.2.0.20 7](#_Toc380071158)

[1.3.0.24 7](#_Toc380071159)

[Замеченные ошибки и неопределённое поведение 7](#_Toc380071160)

[Компоненты сторонних разработчиков 8](#_Toc380071161)

[Примеры 8](#_Toc380071162)

[Пример 1: простейшая программа 8](#_Toc380071163)

[Пример 2: литература, числящаяся за читателем 10](#_Toc380071164)

[Пример 3: создание заказа на книгу 12](#_Toc380071165)

[Архитектура библиотеки 14](#_Toc380071166)

[Сборки 14](#_Toc380071167)

[Многопоточность и состояние занятости 14](#_Toc380071168)

[Автоматическое восстановление связи 15](#_Toc380071169)

[Кеширование 15](#_Toc380071170)

[Прерывание длинных запросов 15](#_Toc380071171)

[Подтверждение подключения 16](#_Toc380071172)

[Исключения и их обработка 16](#_Toc380071173)

[Класс IrbisRecord 16](#_Toc380071174)

[Класс SubField 16](#_Toc380071175)

[Класс RecordField 17](#_Toc380071176)

[Класс IrbisRecord 19](#_Toc380071177)

[LINQ: манипуляции с записями, полями и подполями 22](#_Toc380071178)

[Класс ManagedClient64 25](#_Toc380071179)

[Подключение и отключение от сервера 26](#_Toc380071180)

[Определение версии сервера и клиента 28](#_Toc380071181)

[Переключение контекстов клиента 29](#_Toc380071182)

[Основные операции с записями 31](#_Toc380071183)

[Чтение записей 31](#_Toc380071184)

[Поиск записей 31](#_Toc380071185)

[Форматирование записей 32](#_Toc380071186)

[Сохранение записей 33](#_Toc380071187)

[Массовое чтение записей 34](#_Toc380071188)

[Массовое сохранение записей 34](#_Toc380071189)

[Получение списка терминов 34](#_Toc380071190)

[Административные функции 35](#_Toc380071191)

[Сериализация состояния клиента 38](#_Toc380071192)

[Элементы пользовательского интерфейса 39](#_Toc380071193)

[Класс IrbisBusyManager 39](#_Toc380071194)

[Класс IrbisAuthenticationManager 40](#_Toc380071195)

[Класс IrbisHeartbeatManager 40](#_Toc380071196)

[Стандартные структуры данных 40](#_Toc380071197)

[Классы ReaderInfo и VisitInfo 40](#_Toc380071198)

[Автоматизация при помощи JavaScript 40](#_Toc380071199)

[Работа с RFID 40](#_Toc380071200)

# Введение

Библиотека «ManagedClient64» (в дальнейшем – «библиотека») предназначена для организации программного доступа к ресурсам, находящихся под управлением сервера ИРБИС64, и может использоваться для как для расширения функциональности стандартных АРМ, входящих в поставку АБИС ИРБИС64, так и для создания собственных программных продуктов, совмесимых с ИРБИС64.

## Совместимость

Библиотека совместима со следующими версиями ИРБИС64:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 2004 | 2005 | 2006 | 2007 | 2008 | 2009 | 2010 | 2011 | 2012 | 2013 |
| + | **?** | **+** | **+** | **+** | **+** | **?** | **+** | **+** | **?** |

Совместимость с конкретной версией сервера ИРБИС64 устанавливается по результатам прогона набора стандартных тестов: подключение к серверу, получение служебной информации (версия сервера, количество лицензий и т. д.), чтение записей, форматирование записей, сохранение записей и т. д.

## Инструментарий

Библиотека написана на языке C# в среде Microsoft Visual Studio 2008. Для сборки библиотеки из исходных текстов необходим совместимый инструментарий: Visual Studio 2008 или более новой версии как бесплатной редакции (Express), так и платной (Stadard, Professional и т. д.).

Библиотека должна без модификации успешно собираться средами MonoDevelop (версия не ниже 4.0) и SharpDevelop (версия не ниже 4.4).

Однако всё многообразие альтернативного инструментария не было протестировано авторами (и они не ставили перед собой подобной задачи), и авторы рекомендуют использовать для сборки Visual Studio 2008 или Visual Studio 2013 – две проверенные авторами среды.

## Системные требования

Основным системным требованием библиотеки является наличие Microsoft.Net framework 3.5 SP1 или совместимой с ним среды исполнения управляемого кода.

Библиотека (основная сборка ManagedClient.dll) должна функционировать в следующем окружении:

|  |  |
| --- | --- |
| Окружение | Функционирование, требования |
| Microsoft Windows XP SP3 (только x86) | Необходимо установить .Net framework 3.5 SP1 |
| Microsoft Windows Vista (только x86) | Необходимо установить .Net framework 3.5 SP1 |
| Microsoft Windows Server 2003 (x86/x64) | Необходимо установить .Net framework 3.5 SP1 |
| Microsoft Windows 7/8/8.1 (x86/x64) | Необходимо разрешить .Net framework как системный компонент |
| Microsoft Windows Server 2008/2008 R2/2012 (x86/x64) | Необходимо разрешить .Net framework как системный компонент |
| Linux (x86/x64) | Необходимо установить Mono версии не ниже 3.2.6 |
| Android | Необходим Xamarin Mono for Android версии не ниже 4.0.4 |
| Silverlight | Поддерживается Silverlight версии не ниже 4 |
| Windows Mobile 6/6.5 | Необходимо установить .Net Compact framework 3.5 |
| Windows Phone | Не проверялось |
| Windows RT | Не проверялось |
| iOS (iPhone/iPad) | Не проверялось, возможно будет работать под Xamarin Mono for iOS |
| MacOS | Не проверялось |

## Сайт

Библиотека размещена на SVN-хостинге Assembla.com по адресу <https://www.assembla.com/spaces/arsmagna>. Там же размещена документация по проекту (в формате Wiki) и релизные сборки проекта. Сайт Assembla требует регистрации, однако она бесплатна.

Доступ SVN-репозиторию библиотеки, расположенному по адресу <https://subversion.assembla.com/svn/arsmagna/>, открыт на чтение для всех зарегистрированных на сайте Assembla.

По адресу <https://www.assembla.com/spaces/arsmagna/documents> хранится актуальная релизная сборка библиотеки.

## Лицензия

Библиотека распространяется как продукт с открытым исходным кодом. Любой желающий может:

* Использовать бинарный релиз библиотеки в своих проектах в неизменном виде – в этом случае требуется лишь указание на авторство библиотеки.
* Адаптировать исходный код для собственных нужд и использовать в своих проектах модифицированную версию библиотеки или [модифицированные] фрагменты кода из неё – в этом случае требуется указание на авторство библиотеки и факт модификации её кода.

Никаких лицензионных отчислений в вышеперечисленных случаях не требуется.

## Благодарности

Авторы выражают благодарность:

* Ивану Батраку (СФУ), протестировавшему библиотеку на совместимость со старыми версиями ИРБИС-сервера;
* Шувалову Арсению Валентиновичу (Саратовская государственная консерватория им. Л. В. Собинова), выявившему ошибки в библиотеке;
* Артёму Васильевичу Гончарову (Научная музыкальная библиотека Санкт-Петербургской Консерватории им. Н. А. Римского-Корсакова), выявившему некоторые досадные ошибки в библиотеке.

## Версия и совместимость

Данное руководство описывает версию 1.3.0.24 библиотеки. Версия библиотеки физически хранится как ресурс VERSION сборки ManagedClient.dll и как статическое свойство Version класса ManagedClient64.

Подробнее о проверке версий см. пункт «Определение версии сервера и клиента».

На данный момент несовместимых версий библиотеки нет, поэтому обновление может осуществляться простым копированием новой сборки поверх старой.

Все будущие несовместимости, если таковые появятся, будут описаны в данном разделе.

## Хроника версий

Обозначения:

|  |  |
| --- | --- |
| **+** | **Добавлена функциональность** |
| **–** | **Устранена ошибка** |
| **\*** | **Изменения** |
| **!** | **Внимание! Несовместимость или неопределённое поведение** |

### 1.0.0.1

Первоначальная версия библиотеки.

### 1.1.0.12

|  |  |
| --- | --- |
| – | Убраны лишние символы, отсылавшиеся на сервер с запросом. |
| – | Правильная работа с сервером версий 2004-2006. |
| + | Реализовано массовое чтение и массовое сохранение записей: классы BatchRecordReader и RecordBuffer. |
| + | Реализовано прямое чтение базы данных (экспериментально). |

### 1.2.0.20

|  |  |
| --- | --- |
| – | Логин и пароль пользователя, содержащие не-латинские символы, теперь передаются на сервер правильно. |
| – | Методы ReadRecords и SearchRead в ManagedClient64 правильно определяют версию записи. Теперь записи, считанные этими методами, стали модифицируемыми (т. е. их можно отсылать обратно на сервер). |
| + | Реализовано чтение двоичных файлов с сервера. |
| + | Событие BusyChanged и свойство WaitHandle в ManagedClient64. |

### 1.3.0.24

|  |  |
| --- | --- |
| + | Класс IrbisBusyManager. |
| + | Классы ReaderInfo и VisitInfo. |

## Замеченные ошибки и неопределённое поведение

1. Метод FormatRecord ( string format, IrbisRecord record ) класса ManagedClient64 возвращает пустую строку при format="@". К сожалению, таков результат, возвращаемый сервером, и на данный момент нет идей, как обойти ошибочное поведение сервера.
2. Прерывание длинных запросов на сервер пока не работает, т. к. сокеты ещё не переведены в неблокирующий режим.

## Компоненты сторонних разработчиков

Базовый функционал библиотеки самодостаточен и не требует компонентов, не входящих в стандартную среду исполнения.

Для исполнения скриптов применяется свободно распространяемый компонент RemObjects Script 2.0 for .Net компании RemObjects Software (<http://www.remobjects.com/script.aspx>). Компонент представляет собой полностю управляемую реализацию языка ECMAScript 1.5 (также известного как JavaScript) с открытым исходным текстом.

Для работы с бесконтактными идентификационными картами в стандарте SCSC (также известном как WinSCard) применяется свободно распространяемая библиотека pcsc-sharp (<https://code.google.com/p/pcsc-sharp/>).

# Примеры

## Пример 1: простейшая программа

Ниже прилагается пример простой программы. Сначала находятся и загружаются 10 первых библиографических записей, в которых автор начинается на букву "A" (кириллица). Показано нахождение значения поля с заданным тегом и подполя с заданным кодом. Также показано расформатирование записи в формат brief.

Затем создается и отправляется на сервер 10 записей. Показано добавление в запись полей с подполями.

using System;

using System.Linq;

using ManagedClient;

namespace TestApplication

{

class Program

{

private static void Main()

{

try

{

using (ManagedClient64 client = new ManagedClient64())

{

// Подключаемся к серверу

сlient.ParseConnectionString("host=127.0.0.1;"

+ "port=6666;user=1;password=1;");

client.Connect();

// Ищем все книги, у которых автор

// начинается на А (кириллица)

int[] foundRecords = client.Search("\"A={0}$\"", "А");

// Чтобы не распечатывать все найденные записи,

// ограничимся первыми 10

int recordsToShow = Math.Min(foundRecords.Length, 10);

for (int i = 0; i < recordsToShow; i++)

{

int thisMfn = foundRecords[i];

// Считываем запись

IrbisRecord record = client.ReadRecord(thisMfn);

// Получаем основное заглавие

string mainTitle = record

.Fields

.GetField("200")

.GetSubField('a')

.GetSubFieldText()

.FirstOrDefault();

// Можно было просто написать:

// string mainTitle = record.FM("200", 'a');

Console.WriteLine

(

"MFN={0}, Main title={1}",

thisMfn,

mainTitle

);

// Расформатируем запись

Console.WriteLine

(

"BRIEF: {0}",

client.FormatRecord("@brief", record)

);

// Создаем запись

IrbisRecord newRecord = new IrbisRecord();

newRecord.AddField

(

"700",

'a',

"Управляемый клиент ИРБИС64"

)

.AddField

(

"200",

'a',

string.Format ("Новая Запись от {0}",

DateTime.Now),

'f',

"Управляемый клиент"

);

// Отсылаем вновь созданную запись на сервер

client.WriteRecord

(

newRecord,

false,

true

);

Console.WriteLine(new string('-', 60));

}

// По выходу из using автоматически

// вызывается client.Disconnect ()

}

}

catch (Exception ex)

{

Console.WriteLine(ex);

}

}

}

}

## Пример 2: литература, числящаяся за читателем

В качестве примера рассмотрим конкретную проблему: нам необходимо составить список литературы, числящейся за данным читателем и не возвращённой им в библиотеку.

Для этого мы должны:

1. В базе данных RDR найти читателя по его идентификатору или ФИО.

2. В найденной записи найти все повторения поля 40, в которых присутствует подполе a (шифр выданного документа), а подполе f (дата фактического возврата) содержит звёздочки (т. е. возврата ещё не было).

3. В отобранных повторениях 40-го поля подполе c содержит краткое библиографическое описание выданной книги. Выведем его на экран.

Ниже приводится текст программы, которая реализует данный алгоритм:

using System;

using System.Linq;

using ManagedClient;

class Program

{

static void Main ()

{

// Отредактируйте эту строку!!!

string connectionString =

"Host=127.0.0.1;User=1;Password=1;Db=RDR;";

using (var client = new ManagedClient64 ())

{

client.ParseConnectionString (connectionString);

client.Connect ();

while (true)

{

Console.Write ("ФИО или идентификатор читателя: ");

string id = Console.ReadLine();

if (string.IsNullOrEmpty (id))

{

// Пустая строка означает завершение работы

// с программой

break;

}

// Ищем читателя с указанным идентификатором

IrbisRecord record = client.SearchRead

(

"\"RI={0}\"",

id

)

.FirstOrDefault ();

if (record == null)

{

Console.WriteLine ("Нет такого читателя!");

continue;

}

// Выбираем поля 40, в которых присутствует поле a

// (шифр документа в базе), а подполе f (дата

// фактического возврата) состоит из звёздочек

RecordField[] loans = record.Fields

.GetField ( "40" )

.Where ( field => field.HaveSubField ( 'a' )

&& field.GetSubFieldText ( 'f', 0 ).StartsWith ("\*") )

.ToArray ();

Console.WriteLine ( "Список задолженных книг "

+ "для читателя {0}:", id );

foreach ( var loan in loans )

{

// Получаем краткое библиографическое описание книги

string description = loan.GetSubFieldText ( 'c', 0 );

Console.WriteLine (description);

}

Console.WriteLine ( new string ( '-', 60 ) );

}

}

}

}

## Пример 3: создание заказа на книгу

Заказы создаются в базе данных RQST, записи в ней содержат следующие поля (конечно же, не все сразу):

|  |  |
| --- | --- |
| 1 | Имя базы данных электронного каталога |
| 30 | Идентификатор читателя |
| 31 | Краткое описание читателя |
| 40 | Дата создания заказа |
| 41 | Дата выполнения заказа |
| 42 | Дата предполагаемого возврата издания |
| 43 | Дата бронирования |
| 44 | Сведения об отказе |
| 50 | Ответственное лицо |
| 100 | Дополнительные сведения при заказе периодики |
| 101 | Примечания читателя |
| 102 | Место выдачи |
| 200 | Заказ по ретрофонду |
| 201 | Краткое описание заказанного издания |
| 903 | Шифр заказанного издания |
| 910 | Сведения о выданном/забронированном экземпляре |
| 911 | Пометка о распечатке заказа |
| 952 | Ссылка на образ КК |

Подполей в этих полях нет.

Для создания заказа достаточно полей 1, 30, 31, 40, 102, 201 и 903.

Заказ литературы заключается в получении сведений сначала из базы RDR (вход читателя в систему), затем из базы IBIS (поиск нужного издания), затем записи в базу данных RQST.

Для работы попеременно с несколькими базами данных может пригодиться метод PushDatabase(string) у класса ManagedClient64. Он временно меняет название базы данных, к которой подключен клиент. Последующий вызов метода PopDatabase восстанавливает предыдущее имя базы данных.

Предполагая, что на данный момент уже известны идентификатор читателя (ведь он как-то вошёл в систему?) и заказываемое издание, сформировать заказ можно следующим образом:

try

{

client.PushDatabase ( "RQST" ); // Временно переключаемся

// на базу RQST

string thisMoment = DateTime.Now // Формируем строку

// с текущими датой

// и временем

.ToString ( "dd-MM-yyyy HH:mm:ss");

IrbisRecord record = new IrbisRecord();

record.AddField ( "1", "IBIS") // Имя электронного каталога

.AddField ( "30", "C42678F6" ) // Идентификатор читателя

.AddField ( "31", "Власова Наталья Александровна" )

// ФИО читателя

.AddField ( "40", thisMoment ) // Дата создания заказа

.AddField ( "102", "ЧЗ" ) // Предполагаемое место выдачи

.AddField ( "201", "Брэдбери, Рей. Вино из "

+ "одуванчиков, 1988. – 656 с." )

// Биб. описание книги

.AddField ( "903", "84(7США)/Б89-041205" );

// Шифр заказываемого издания

client.WriteRecord ( record, false, true );

// Сохраняем запись

}

finally

{

client.PopDatabase (); // Возвращаем предыдущий

// контекст работы

}

# Архитектура библиотеки

## Сборки

Основная сборка библиотеки – **ManagedClient.dll**. Она содержит функционал, не зависящий от платформы. В этой сборке два главных класса: ManagedClient64 и IrbisRecord – собственно, они и выполняют основную работу, а остальные классы являются «обвязкой» к ним.

Сборка **IrbisUI.dll** основывается на Windows Forms и содержит диалоговые окна, необходимые для отображения основных элементов пользовательского интерфейса: диалога входа в систему, рабочих листов, справочников и т. д. Примечание: сборка доступна только в среде Windows.

Подробнее о сборке IrbisUI.dll см. главу «Элементы пользовательского интерфейса».

Сборка **IrbisAutomation.dll** предоставляет функционал автоматизации основных операций при помощи скриптов на языке JavaScript.

Поскольку в этой сборке применяется полностью переносимый компонент RemObjects Script, она должна быть работоспособной в том же окружении, что и сборка ManagedClient.dll.

Подробнее о сборке см. главу «Автоматизация при помощи JavaScript».

Сборка **IrbisRFID.dll** предоставляет возможность считывания бесконтактных идентификационных карт (читательских билетов и RFID-меток на книгах). Примечание: сборка доступна только в среде Windows.

Подробнее о сборке см. главу «Работа с RFID».

## Многопоточность и состояние занятости

Библиотека написана в наивном однопоточном стиле, поэтому не поддерживает одновременное выполнение команд из разных потоков. При попытке вызвать метод сервера во время обработки запроса библиотека выбрасывает исключение.

Обязанность по сериализации запросов к серверу возлагается на приложение.

Во время обращения к серверу экземпляр класса ManagedClient64 устанавливает флаг Busy, по завершению (в том числе неуспешному) – сбрасывает его. Для отслеживания состояния этого флага предусмотрено событие BusyChanged. Кроме того, для эффективного многопоточного ожидания предусмотрено поле WaitHandle, которое переходит в состояние Nonsignailed при обращении к серверу и в состояние Signailed по окончании запроса. Поэтому дожидаться окончания запроса можно так:

if (client.WaitHandle.WaitOne(1000)) // Ждём не больше секунды

{

// Дождались окончания запроса

}

else

{

// Клиент всё ещё чем-то занят

}

Можно ожидать завершения нескольких операций:

WaitHandle[] handles = { client1.WaitHandle, client2.WaitHandle };

int index = WaitHandle.WaitAny(handles, 1000);

if (index >= 0)

{

// Дождались одного из клиентов

}

## Автоматическое восстановление связи

При любом сбое связи с сервером библиотека выбрасывает исключение. Обязанность по восстановлению после сбоев возложена на приложение.

## Кеширование

Библиотека не кеширует данные, полученные от сервера (ни записи, ни форматы, ни результаты поиска). Эта обязанность возложена на приложение.

## Прерывание длинных запросов

В библиотеке не предусмотрено самостоятельно обработки слишком долгих или слишком ресурсоёмких операций. Эта обязанность возложена на

Длинный запрос к серверу может быть прерван вызовом метода Interrupt у соответствующего экземпляра класса ManagedClient. О том, что запрос был прерван свидетельствует флаг Interrupted, установленный по окончанию отработки запроса.

**ВНИМАНИЕ: в настоящее время прерывание длинных запросов не работает!**

## Подтверждение подключения

Сервер ИРБИС64 ожидает от клиента периодического подтверждения активности. При длительном отсутствии активности клиента сервер автоматически отключает его, и все последующие запросы от него отвергаются (пока клиент не выполнит процедуру подключения снова).

Библиотека не отслеживает состояние активности клиента и не посылает самостоятельно подтверждений на сервер. Эта задача возложена на использующее библиотеку приложение.

Для отсылки подтверждений необходимо применять метод NoOp класса ManagedClient.

## Исключения и их обработка

**TOBEDONE**

# Класс IrbisRecord

## Класс SubField

Каждое подполе записи в формате MARC соответствует экземпляру класса SubField, который содержит следующие элементы:

public char Code { get; set; } // Код подполя

public string Text { get; set; } // Значение подполя

Конструкторы:

public SubField();

public SubField(char code);

public SubField(char code, string text);

Обратите внимание: библиотека сохраняет регистр символов в поле Code, однако игнорирует его при сравнении.

Методы:

public SubField Clone (); // Полная копия подполя

public static int Compare // Сравнение двух подполей

(

SubField subField1,

SubField subField2,

bool verbose

);

## Класс RecordField

Каждое поле записи в формате MARC соответствует экземпляру класса RecordField, который содержит следующие элементы:

public string Tag { get; set;} // Метка поля

public string Text { get; set; } // Значение поля до первого

// разделителя подполей

public string List<SubField> SubFields { get; } // Список подполей

Обратите внимание:

1. Библиотека допускает создание полей с произвольными, в том числе буквенными, метками, т. к. это допускает оригинальный формат MARC. Однако записи с такими метками полей вызовут ошибку ИРБИС-сервера.
2. Метки полей следует задавать без ведущих нулей: «10», а не «010».
3. Поле Text содержит текст до первого разделителя подполей. У полей вроде «700: Первый автор» Text всегда будет равно null (т. к. все значимые элементы уйдут в коллекцию SubFields). А у полей, подобных «300: Общие примечания» Text будет содержать весь текст, введённый поле.
4. Указатель на список подполей SubFields никогда не равен null. Если поле не содержит подполей, список просто будет пуст.

Конструкторы:

public RecordField();

public RecordField(string tag);

public RecordField(string tag, string text);

Методы:

public SubField[] GetSubField(char code); // Получаем все подполя

// с указанным кодом. Если таких подполей нет, возвращается

// пустой массив.

public SubField GetSubField(char code); // Указанное повторение

// подполя с данным кодом. Если подполя с таким кодом нет,

// возвращается пустой массив.

public string GetSubFieldText // Текст указанного повторения

( // подполя

char code, // код подполя

int occurrence // номер повторения

);

public string GetFirstSubFieldText // Текст первого повторения

( // подполя

char code

);

public RecordField AddSubField // Добавляет (в конец списка)

( // подполе с указанными кодом и текстом

char code, // Код создаваемого подполя

string text // Текст

);

public RecordField SetSubField // Устанавливает значение подполя.

( // Если подполей с указанным кодом в поле несколько,

// устанавливается только первое повторение. Если таких подполей

// нет, экземпляр подполя создаётся и добавляется в список.

char code, // Код подполя

string text // Устанавливаемое значение

);

public RecordField ReplaceSubField // Замена указанного значения

( // во всех повторениях подполя с данным кодом.

// Меняется весь Text, а не подстроки!

char code, // Код подполя

string oldValue, // Старое значение

string newValue // Новое значение

);

public RecordField RemoveSubField // Удаление всех повторений

( // подполя с данным кодом

char code // Код подполя

);

public bool HaveSubField // Есть ли в поле хоть одно подполе

( // с указанными кодами

params char[] codes

);

public bool HaveNotSubField // В поле нет ни одного подполя

( // с указанными кодами

params char[] codes

);

public string ToText(); // Конвертация в строковое представление

public RecordField Clone(); // Полная копия поля со всеми подполями

Обратите внимание:

1. Нумерация повторений подполей начинается с 0.
2. Если подполя с указанным кодом нет, или нет повторения с
3. ИРБИС плохо поддерживает поля с более чем одним повторением подполя. В частности, при визуальном редактировании второе и последующие повторения подполя теряются.
4. Хотя подполя с пустым текстом сохраняются в записи, при визуальном редактировании ИРБИС их удалит из поля.

Методы AddSubField, SetSubField, ReplaceSubField и RemoveSubField возвращают указатель на поле, чтобы сделать возможным их вызов «по цепочке». Пример:

field

.AddSubField('a', "Подполе a")

.SetSubField('b', "Подполе b")

.ReplaceSubField('c', "Было", "Стало")

.RemoveSubField('d');

Методы HaveSubField и HaveNotSubField принимают открытый массив кодов, что позволяет перечислить при вызове метода произвольное число кодов через запятую:

if (field.HaveSubField('a', 'e', 'f'))

...

Переопределённый стандартный метод ToString конвертирует поле в строковое представление с меткой, в то время как метод ToText не включает метку в строковое представление:

field.ToText(); // ^aЗаглавие^eПодзаголовочные сведения

field.ToString(); // 200#^aЗаглавие^eПодзаголовочные сведения\r\n

## Класс IrbisRecord

Каждый экземпляр класса IrbisRecord соответствует одной записи в базе данных ИРБИС. Он содержит следующие элементы:

public string Database { get; } // Имя базы данных, из которой

// считана запись. Для вновь созданных записей null

public int Mfn { get; } // Номер записи в мастер-файле.

// Для вновь созданных записей 0

public RecordStatus Status { get; } // Статус записи – логически

// удалена, физически удалена и т. д.

public int Version { get; } // Версия записи.

// Устанавливается сервером автоматически

public bool Deleted { get; set; } // Признак удалённой записи

// (чтобы не возиться лишний раз с полем Status)

public List<RecordField> Fields { get; } // Список полей записи

public object UserData { get; set; } // Произвольные

// пользовательские данные

Обратите внимание:

1. Запись «помнит» базу данных, из которой она была считана, и MFN, который она имеет в этой базе.
2. Кроме того, ИРБИС-сервер поддерживает список версий для каждой записи в своих базах данных. Таким образом, он пытается противодействовать противоречивым правкам, произведённым одновременно с разных рабочих мест. Неправильное указание номера версии записи приводит к тому, что сервер игнорирует запрос на сохранение записи.

Для класса IrbisRecord предусмотрен только конструктор по умолчанию:

public IrbisRecord ();

Методы конструирования записи:

public IrbisRecord AddField // Добавление поля (в конец списка

( // подполей)

string tag, // Метка создаваемого поля

string text // Текст поля до разделителей

);

public IrbisRecord AddField // Добавление поля (в конец списка

( // подполей)

string tag, // Метка создаваемого поля

char code, // Код первого подполя

string text, // Текст первого подполя

params object[] others // Коды и тексты последующих подполей

);

public IrbisRecord SetField // Установка значения для первого

( // повторения поля с указанной меткой. Если такого поля нет,

// оно будет создано

string tag, // Метка поля

string text // Текст поля до разделителей

);

public IrbisRecord SetSubField // Установка значения для первого

( // повторения подполя в первом повторении поля.

string tag, // Метка поля

char code, // Код подполя

string text // Значение подполя

);

public IrbisRecord RemoveField // Удаление из записи всех

( // повторений поля с указанной меткой

string tag // Метка поля

);

public IrbisRecord RemoveField // Удаление из записи данного

( // повторения поля с указанной меткой

string tag, // Метка поля

int occurrence // Номер повторения (с 0)

);

public IrbisRecord Clone (); // Создание полной копии записи

Обратите внимание:

1. Повторения полей и подполей нумеруются с 0.
2. Если поля с указанным повторением нет в записи, ошибки не происходит: метод просто возвращает управление.

Методы AddField, SetField, SetSubField, RemoveField возвращают указатель на запись, чтобы сделать возможным их вызов «по цепочке». Пример:

record

.AddField("200", 'a', "Заглавие", 'f', "Ответственность")

.AddField("300", "Общее примечание")

.SetField("999", "000000")

.SetSubField("900", 'a', "a")

.RemoveField("907", 2);

Простые методы поиска полей и подполей:

public bool HaveField // Есть ли в записи хоть одно поле

( // из перечисленных

params string[] tags

);

public bool HaveNotField // В записи нет ни одного поля

( // из перечисленных

params string[] tags

);

public string FM // Текст до разделителя первого повторения поля

( // с указанной меткой. Если такого поля нет, выдаётся null.

string tag

);

public string[] FMA // Текст до разделителя всех повторений поля

( // с указанной меткой. Если такого поля нет, выдаётся

// пустой массив.

string tag

);

public string FM // Значение первого повторения указанного подполя.

( // Если его нет, выдаётся null.

string tag, // Метка поля

char code // Код подполя

);

public string[] FMA // Значения всех повторений указанного подполя.

( // Если таких подполей нет, выдаётся пустой массив.

string tag, // Метка поля

char code // Код подполя

);

Примеры:

record.FM("999"); // Счётчик выдач

record.FM("200", 'a'); // Заглавие

record.FMA("300"); // Общие примечания массивом

record.FMA("910", 'b'); // Инвентарные номера массивом

## LINQ: манипуляции с записями, полями и подполями

Кроме перечисленных в предыдущем разделе простых методов отбора записей, библиотека предлагает набор методов расширения (на основе технологии LINQ), которые позволяют отбирать поля и подполя записи, исходя из сколь угодно сложных критериев, причём форма записи (благодаря LINQ) остаётся довольно удобочитаемой.

Пример сложного условия, отбирающего поля записи:

var v910 = record.Fields

.GetField("910")

.GetField('a', "0");

Вышеприведённое выражение отбирает из записи поля с меткой «910», в которых значение подполя «a» равно «0».

Ниже приводится перечень основных

| Метод | Назначение, примеры применения |
| --- | --- |
| Применяется к T[] и IList<T> | |
| T GetOccurrence ( int n ) | Получить n-ое повторение. Если n < 0, то отсчёт с конца (т. е. -1 означает последний элемент, -2 предпоследний и т. д.)  fields.GetOccurrence(-1); |
| Применяется к IEnumerable<RecordField> | |
| SubField[] AllSubFields () | Получить общий список подполей |
| RecordField[] GetField  (  string tag  ) | Отобрать только поля с указанной меткой  fields.GetField("910"); |
| RecordField[] GetField  (  params string[] tags  ) | Отобрать только поля с указанными метками  fields.GetField("700", "701"); |
| RecordField[] GetField  (  Func<RecordField, bool>  predicate  ) | Отобрать поля по предикату  fields.GetField  (  field => field.Tag > "200"  ); |
| RecordField[] GetField  (  char[] codes,  Func<SubField, bool> predicate  ) | Отобрать поля по предикату, применяющемуся к подполям  fields.GetField  (  new[] { 'a', 'c' },  sub => sub.Length > 2  ); |
| RecordField[] GetField  (  char[] codes,  params string[] values  ) | Отобрать поля, в которых любое из указанных подполей принимает любое из перечисленных значений  fields.GetField  (  new[] { 'a', 'c' },  new[] { "да", "нет", "не знаю" }  ); |
| RecordField[] GetField  (  char code,  string value  ) | Отобрать поля, в которых указанное подполе имеет строго определённое значение  fields.GetField  (  'a',  "Иркутск"  ); |
| RecordField[] GetField  (  string[] tags,  char[] codes,  string[] values  ) | Отобрать поля с указанными метками, в которых любое из перечисленных полей принимает любое из перечисленных значений  fields.GetField  (  new[] { "700", "701", "702" },  new[] { 'a', 'r' },  new[] { "Иванов", "Иванидзе" }  ); |
| RecordField[] GetField  (  Func<RecordField, bool>  fieldPredicate,  Func<SubField, bool>  subPredicate  ) | Отобрать поля по предикату, при этом для любого из подполей должно выполняться условие, заданное вторым предикатом  fields.GetField  (  field => field.Tag > "200",  sub => sub.Text.Length == 2  ); |
| RecordField[] GetFieldRegex  (  string tagRegex  ) | Отобрать поля по регулярному выражению для их меток  fields.GetFieldRegex ("6$");  // Метки полей заканчиваются на 6 |
| RecordField[] GetFieldRegex  (  string[] tags,  string textRegex  ) | Отобрать поля с указанными метками, текст до разделителя должен удовлетворять регулярному выражению  fields.GetFieldRegex  (  new[] { "300", "327" },  @"\bБайкал\b" // слово целиком  ); |
| RecordField[] GetFieldRegex  (  string[] tags,  char[] codes,  string textRegex  ) | Отобрать поля с указанными метками, в которых значение любого из перечисленных подполей удовлетворяет регулярному выражению  fields.GetFieldRegex  (  new[] { "200", "517", "921" },  new[] { 'a' },  @"\b\d+\b" // слово целиком  ); // состоит из цифр |
| string[] GetFieldText () | Получить текст поля (отбираются только значения, не равные null)  fields  .GetField ("300", "327")  .GetFieldText (); |
| SubField[] GetSubField  (  Func<RecordField, bool>  fieldPredicate,  Func<SubField, bool>  subPredicate  ) | Отобрать подполя по предикату среди полей, отобранных по предикату  fields.GetSubField  (  field => field.Tag > "200",  sub => sub.Code > 'x'  ); |
| SubField[] GetSubField[]  (  string tag,  char code  ) | Отобрать подполя с указанным кодом среди полей с указанной меткой  fields.GetSubField("910", 'b'); |
| SubField[] GetSubField  (  string[] tags,  char[] codes  ) | Отобрать подполя с указанными кодами среди полей, отобранных по указанным меткам  fields.GetSubField  (  new[] { "200", "517", "921" },  new[] { 'a' }  ); |
| SubField GetSubField  (  string tag,  char code,  int occurrence  ) | Отобрать определённое повторение подполя с указанным кодом среди полей с указанной меткой  fields.GetSubField("910", 'b', 3); |
| SubField GetSubField  (  string tag,  int fieldOccurrence,  char code,  int subOccurrence  ) | Отобрать определённое повторение подполя с указанным кодом в определённом повторении поля с указанной меткой  fields.GetSubField  (  "910",  2, // нумерация с 0 !!!  'b',  1 // нумерация с 0 !!!  ); |
| Применяется к IEnumerable<SubField> | |
| SubField[] GetSubField  (  char code  ) | Отобрать подполя строго с указанным кодом  subFields.GetSubField('c'); |
| SubField[] GetSubField  (  params char[] codes  ) | Отобрать подполя с указанными кодами  subFields.GetSubFields('a', 'c'); |
| SubField[] GetSubFieldRegex  (  string codeRegex  ) | Отобрать подполя с кодами, удовлетворяющими регулярному выражению  subFields.GetSubFieldRegex  (  @"[a-e3-5]"  ); |
| SubField[] GetSubFieldRegex  (  char[] codes,  string textRegex  ) | Отобрать подполя с указанными кодами и значениями, удовлетворяющими регулярному выражению  subFields.GetSubFieldRegex  (  new[] { 'a', 'c' },  @"\bБайкал" // граница слова  ); // слева (в начале) |
| string[] GetSubFieldText () | Получить текст подполя  subFields  .GetSubField('a', 'c')  .GetSubFieldText(); |

# Класс ManagedClient64

ManagedClient64 – «рабочая лошадка». Этот класс осуществляет связь с сервером, всю необходимую «перепаковку» данных и прочее и прочее. Собственно, это и есть управляемый клиент ИРБИС64.   
Экземпляр клиента создаётся конструктором по умолчанию:

var client = new ManagedClient64 ();

## Подключение и отключение от сервера

Параметры подключения к серверу определяются следующими свойствами

|  |  |  |
| --- | --- | --- |
| Свойство | Назначение | Значение по умолчанию |
| public string Host { get; set; } | Адрес сервера | "127.0.0.1" |
| public int Port { get; set; } | Порт сервера | 6666 |
| public string Username { get; set; } | Имя пользователя | null |
| public string Password { get; set; } | Пароль | null |
| public string Database { get; set; } | Имя базы данных | "IBIS" |
| public IrbisWorkstation Workstation { get; set; } | АРМ | IrbisWorkstation.Cataloger |

Обратите внимание, что адрес сервера задаётся строкой, так что может принимать как значения вроде "192.168.1.1", так и "irbis.yourlib.com".

Если какой-либо вышеперечисленных из параметров подключения не задан явно, используется значение по умолчанию. По умолчанию, клиент готов к работе с локально установленной версией ИРБИС64.

За подключение и отключение отвечают два метода:

public void Connect ();

public void Disconnect ();

При возникновении ошибки при подключении метод Connect выбросит исключение.

После успешного установления соединения повторные вызовы Connect не выполняют никаких действий. Аналогично после успешного отключения повторные вызовы Disconnect не выполняют никаких действий.

Проверить, подключены ли мы в данный момент к серверу, можно с помощью свойства Connected:

public bool Connected { get; }

Вместо индивидуального задания каждого из параметров Host, Port, Username, Password и Database, можно использовать метод ParseConnectionString:

public void ParseConnectionString ( string connectionString );

Строка подключения состоит из элементов вида «Параметр=Значение», разделённых точкой с запятой. В следующей таблице приведены все возможные пары «Параметр=Значение», воспринимаемые методом ParseConnectionString. Имена параметров (но не значения!) регистронезависимы. Имена параметров, перечисленные через запятую, эквивалентны.

|  |  |  |  |
| --- | --- | --- | --- |
| Параметр | Тип | Назначение | Значение по умолчанию |
| host, server, port | строка | Адрес сервера | "127.0.0.1" |
| port | целое | Порт сервера | 6666 |
| user, username, name, login | строка | Имя пользователя | нет |
| pwd, password | строка | Пароль | нет |
| db, catalog, database | строка | База данных по умолчанию | "IBIS" |
| arm, workstation | символ | АРМ | 'C' |
| data | строка | Произвольное значение (помещается в поле UserData) | нет |

Рекомендуется заключать всю работу с клиентом в контекстные скобки using языка C# для корректного освобождения ресурсов при возникновении нештатных ситуаций.

После успешного подключения к серверу свойство Configuration содержит серверный INI-файл для указанного клиента, а свойство Settings организует удобный доступ к настройкам АРМ, заданным в INI-файле:

public string Configuration { get; }

public IniFile Settings { get; }

Пример получения настроек:

// Получаем интервал автоматического подтверждения

// активности клиента

int refreshInterval = client.Settings.Get<int>

(

"Main",

"CLIENT\_TIME\_LIVE",

15

);

Пример подключения к серверу:

using (var client = new ManagedClient64 ())

{

string connectionString = System.Configuration

.AppSettings["connection-string"];

client.ParseConnectionString ( connectionString );

client.Connect ();

// Теперь мы подключены к серверу и можем работать с ним.

// Например, можем получить количество записей в базе данных.

Console.WriteLine

(

"Записей в базе: {0}",

client.GetMaxMfn() - 1

);

// по выходу из блока using отключение от сервера произойдёт

// само, поэтому вызывать Disconnect необязательно

}

## Определение версии сервера и клиента

Метод GetVersion получает сведения о сервере: его версию, максимальное количество подключений и т. д. Класс IrbisVersion служит для хранения этих сведений.

public IrbisVersion GetVersion();

public sealed class IrbisVersion

{

// На какую организацию зарегистрирована данная копия сервера.

public string Organization { get; set; }

// Собственно версия сервера. Например, "64.2008.1".

public string Version { get; set; }

// Максимальное количество одновременных подключений.

public int MaxClients { get; set; }

// Текущее количество подключений (АРМ «Читатель»

// не учитывается).

public int ConnectedClients { get; set; }

}

Для упрощения работы с версией сервера имеется метод RequireServerVersion:

public bool RequireServerVersion

(

string minimumVersion, // Минимальная требуемая версия

bool throwException // Бросать ли исключение?

);

Пример:

if (!client.RequireServerVersion("2008",false))

{

MessageBox.Show("Ваш сервер слишком старый");

}

Так как библиотека поставляется в виде отдельной сборки, может возникнуть необходимость проверить версию библиотеки. Для этого предусмотрен метод RequireClientVersion:

public bool RequireClientVersion

(

Version minimumVersion, // Минимальная требуемая версия

bool throwException // Бросать ли исключение?

);

Пример:

if (!client.RequireClientVersion

(

new Version ( 1, 3 ),

false

))

{

MessageBox.Show("Версия клиента слишком старая");

}

## Переключение контекстов клиента

Имя текущей базы данных (каталога), с которой работает клиент, хранится в поле Database:

public string Database { get; set; }

Кроме того, имеются два полезных метода:

/// Временно устанавливает новое имя текущей базы данных.

/// Запоминает, к какой базе был подключен

/// клиент на момент смены.

/// Возвращает имя предыдущей текущей базы данных.

public string PushDatabase(string newDatabase);

/// Восстанавливает подключение к предыдущей базе данных,

/// сменённой методом PushDatabase().

/// Возвращает имя базы данных, к которой был подключен

/// клиент на момент восстановления состояния.

public string PopDatabase()

Оба метода работают по принципу стека: предыдущие базы данных запоминаются в стеке и постепенно возвращаются по мере вызова метода PopDatabase(). Пример:

// Мы работали с базой IBIS,

// но решили временно подключиться к RDR

client.PushDatabase ("RDR"); // IBIS запоминается в стеке

IrbisRecord reader = client.SearchReadOneRecord ("I=1234");

// Теперь временно подключаемся к RQST

client.PushDatabase ("RQST"); // RDR также запоминается в стеке

IrbisRecord request = new IrbisRecord ();

...

client.WriteRecord (request, false, true); // Запись пойдёт в RQST

...

// Возвращаемся к базе RDR

client.PopDatabase ();

client.WriteRecord (reader, false, true); // Запись пойдёт в RDR

...

// Возвращаемся к исходной базе IBIS

client.PopDatabase ();

Рекомендуется временные переключения между базами оформлять в блоке try-finally:

client.PushDatabase ("CMPL");

try

{

// Какие-то манипуляции с базой

}

finally

{

// Гарантированно возвращаемся в правильный контекст

// работы с базой данных

client.PopDatabase();

}

Следует иметь в виду, что запись может «помнить» свою базу данных (поле Database в IrbisRecord). В этом случае клиент отправит запись в её «родную» базу данных, независимо от текущего контекста.

## Основные операции с записями

### Чтение записей

Методы для чтения и сохранения записей (класс IrbisRecord будет рассмотрен позже, пока воспринимайте его как «чёрный ящик»):

// Чтение одной записи по её MFN

public IrbisRecord ReadRecord ( int mfn );

// Чтение массива записей с указанными MFN

public IrbisRecord[] ReadRecords

(

IEnumerable<int> mfns

);

Запись «помнит» свой MFN, поэтому заботиться о том, чтобы модифицированная запись встала на нужное место, не нужно. Вновь созданная запись имеет MFN=0, поэтому будет помещена в конец мастер-файла.

Кроме того, запись «помнит» свою базу данных (поле Database), поэтому клиент направит её в «родную» базу данных, независимо от текущего контекста.

### Поиск записей

Метод для поиска записей:

// возвращает массив MFN записей, удовлетворяющих запросов

public int[] Search

(

string format,

params object[] args

);

Если ни одной записи не найдено, возвращается массив из 0 элементов.

Пример вызова:

string authorName = "Иванов"; // фамилия автора книги

int[] mfns = client.Search

(

"\"A={0}$\"", // поиск по соответствующему префиксу

authorName

);

// обратите внимание на знак доллара, означающий усечение справа

// и на кавычки, обрамляющие запрос

Поиск с одновременным чтением:

// Возвращает записи, удовлетворяющие запросу.

// По факту совмещает методы Search и ReadRecord

public IrbisRecord[] SearchRead

(

string format,

params object[] args

);

Если ни одной записи не найдено, возвращается массив из 0 элементов.

Пример вызова:

// ищем книги на китайском языке

foreach ( var record in client.SearchRead ( "J=CHI" ) )

{

// Обрабатываем записи по одной

Console.WriteLine ( "Найдено: {0}", record );

}

### Форматирование записей

Во всех нижеперечисленных методах строка format может принимать одно из трёх значений:

* Собственно формат на языке ИРБИС, например "v200^a, ' : ', v200^e";
* Ссылку на серверный pft-файл, начинающуюся с символа "@" (без расширения pft), например: "@brief";
* Ссылку на оптимизированный формат (какой формат использовать, определит сам сервер, основываясь на механизме оптимизации форматов просмотра), состоящую из единственного символа "@".

Методы для форматирования записей:

// Форматирует запись по её MFN

public string FormatRecord

(

string format,

int mfn

);

// Форматирует заданную запись

public string FormatRecord

(

string format,

IrbisRecord record

);

// Форматирует массив записей по их MFN

public string[] FormatRecords

(

string format,

int[] mfns

);

// Форматирует записи,

// удовлетворяющие поисковому запросу

public string[] SearchFormat

(

string expression,

string format

);

Пример вызова:

foreach ( var description in client.SearchFormat

(

"G=2000", // Ищем книги 2000-го года издания

"@brief" // Выводим их краткое биб. описание

) )

{

Console.WriteLine ( description );

}

**ВНИМАНИЕ: формат «@» не работает для метода   
FormatRecord(string,IrbisRecord) !!!**

### Сохранение записей

// Сохранение одной записи

public void WriteRecord

(

IrbisRecord record,

bool needLock, // оставить запись заблокированной

bool ifUpdate

);

// Сохранение массива записей

public void WriteRecords

(

IrbisRecord[] records,

bool ifUpdate // обновить инвертированный файл (поисковый индекс)

);

### Массовое чтение записей

**TOBEDONE**

### Массовое сохранение записей

**TOBEDONE**

## Получение списка терминов

Список терминов можно получить с сервера вызовом метода GetSearchTerms.

public SearchTermInfo[] GetSearchTerms

(

string startTerm,

int count

);

Здесь startTerm — начальный элемент списка (включая префикс вроде «K=»).

Класс SearchTermInfo содержит всю необходимую информацию:

public sealed class SearchTermInfo

{

// Количество ссылок.

public int Count { get; set; }

// Поисковый термин.

public string Text { get; set; }

}

Полную информацию по данному термину можно запросить при помощи метода GetSearchPostings:

public SearchPostingInfo[] GetSearchPostings

(

string startTerm,

int count,

int first

);

class SearchPostingInfo

{

public int Mfn { get; set; }

public string Tag { get; set; }

public int Occurences { get; set; }

public int Count { get; set; }

}

## Административные функции

Метод NoOp посылает на сервер служебный пакет с целью подтверждения активности клиента. Подробнее см. «Класс IrbisHeartbeatManager».

public void NoOp()

Метод GetMaxMfn возвращает максимальный номер MFN в базе данных:

public int GetMaxMfn();

Метод IsDatabaseLocked, определяет, не установлена ли в данный момент монопольная блокировка на базу данных:

public bool IsDatabaseLocked(string databaseName);

Заблокированную базу данных можно разблокировать с помощью метода UnlockDatabase (метод доступен только АРМ «Администратор»):

public void UnlockDatabase(string databaseName);

Удаление и восстановление записей:

public void DeleteRecord

(

IrbisRecord record,

bool ifUpdate // Актуализировать словарь?

);

public void DeleteRecords

(

IEnumerable<IrbisRecord> records,

bool ifUpdate // Актуализировать словарь?

);

public void UndeleteRecord

(

IrbisRecord record,

bool ifUpdate // Актуализировать словарь?

);

Создание и удаление баз данных, реорганизация файла документов, пересоздание словаря:

public void CreateDatabase

(

string databaseName,

string description,

bool readerAccess

);

public void DeleteDatabase

(

string databaseName

);

public void ReorganizeMasterFile

(

string database

);

public void ReorganizeDictionary

(

string database

);

public void RebuildDictionary

(

string database

);

public void ActualizeDatabase

(

string database

);

Перезапуск сервера:

public void RestartServer();

Метод WriteIni сохраняет состояние клиента в серверном INI-файле.

public void WriteIni(string[] iniText);

Получение списка процессов, выполняющихся на сервере:

public IrbisProcessInfo[] GetProcessList();

// Информация о серверном процессе.

public sealed class IrbisProcessInfo

{

// Просто порядковый номер процесса.

public string Number { get; set; }

// С каким клиентом взаимодействует.

public string IPAddress { get; set; }

// Логин оператора.

public string Name { get; set; }

// Идентификатор клиента.

public string ID { get; set; }

// Тип АРМ.

public string Workstation { get; set; }

// Время запуска.

public string Started { get; set; }

// Последняя выполненная (или выполняемая) команда.

public string LastCommand { get; set; }

// Порядковый номер последней команды.

public string CommandNumber { get; set; }

// Идентификатор процесса.

public string ProcID { get; set; }

// Состояние.

public string State { get; set; }

}

Получение и модификация списка пользователей, имеющих доступ к АРМ:

public IrbisUserInfo[] GetUserList();

public void SetUserList(IrbisUserInfo[] users);

// Информация о доступе пользователя к АРМ.

public sealed class IrbisUserInfo

{

// Порядковый номер в списке.

public string Number { get; set; }

// Логин.

public string Name { get; set; }

// Пароль.

public string Password { get; set; }

// INI-файл для АРМ «Каталогизатор».

public string Cataloguer { get; set; }

// INI-файл для АРМ «Читатель».

public string Reader { get; set; }

// INI-файл для АРМ «Книговыдача».

public string Circulation { get; set; }

// INI-файл для АРМ «Комплектатор».

public string Acquisitions { get; set; }

// INI-файл для АРМ «Книгообеспеченность».

public string Provision { get; set; }

// INI-файл для АРМ «Администратор».

public string Administrator { get; set; }

}

## Сериализация состояния клиента

Состояние клиента можно сериализовать в строку для восстановления в другом контексте (это актуально, например, для ASP.NET).

// Сохранение состояния клиента в строку

public static string SerializeToString

(

ManagedClient64 client

);

// Восстановление состояния клиента из строки

public static ManagedClient64 DeserializeFromString

(

string text

);

// Данный метод нужен, чтобы клиент

// не пытался самостоятельно закрыть соединение с сервером.

// Этот метод может пригодиться при сохранении состояния

// клиента с последующим восстановлением.

public void Shutdown ();

Пример сохранения/восстановления состояния клиента:

client = new ManagedClient64();

client.ParseConnectionString("...");

client.Connect();

...

// Поработали с клиентом, сохраняем его состояние

string savedState = ManagedClient64.SerializeToString(client);

database.Save(savedState); // Сохранили, например, в базу данных

client.Shutdown(); // Чтобы клиент не пытался

// отключиться от сервера

...

// Где-то в другом контексте. Например, на другом компьютере.

string savedState = database.Restore ();

ManagedClient64 client = ManagedClient64.DeserializeFromString

(

savedState

);

// Продолжаем работу с сервером

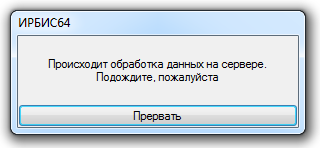
IrbisRecord record = client.ReadRecord(mfn);

# Элементы пользовательского интерфейса

Типовые элементы пользовательского интерфейса хранятся в сборке IrbisUI.dll, основанной на Microsoft Windows Forms. Обратите внимание: эта сборка работает только на платформе Windows!

## Класс IrbisBusyManager

Класс IrbisBusyManager предназначен для отслеживания длительных запросов к серверу и демонстрации пользователю окна «Подождите» с возможностью прервать «зависший» запрос.



**ВНИМАНИЕ: в настоящее время прерывание длительных запросов не работает!**

Типичное использование:

using (ManagedClient64 client = new ManagedClient64())

using (new IrbisBusyManager(client))

{

...

client.Connect();

...

// Выполняем какой-то длительный запрос

// Окно «Подождите» появится и исчезнет автоматически

IrbisRecord[] records = client.SearchRead("A=Пушкин$");

}

Обратите внимание: если длительность запроса не превысит порогового значения, окно «Подождите» не появится! Такой подход уменьшает паразитное «мелькание» на экране.

## Класс IrbisAuthenticationManager

**TOBEDONE**

## Класс IrbisHeartbeatManager

**TOBEDONE**

# Стандартные структуры данных

## Классы ReaderInfo и VisitInfo

**TOBEDONE**

# Автоматизация при помощи JavaScript

**TOBEDONE**

# Работа с RFID

**TOBEDONE**