К сожалению, подробной и полной документации к библиотеке нет, т. к. её просто некому было написать. Ниже приводится краткое введение в библиотеку, поясняющее основные аспекты её функционирования.

В библиотеке два главных класса: ManagedClient64 и IrbisRecord – собственно, они и выполняют основную работу, а остальные классы являются «обвязкой» к ним.

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 к серверу и отключение от него**

Параметры подключения к серверу определяются следующими свойствами

public string Host { get; set; } // Адрес сервера, по умолчанию "127.0.0.1", т. е. локальный  
public int Port { get; set; } // Порт сервера, по умолчанию 6666  
public string Username { get; set; } // Имя пользователя, по умолчанию "1"  
public string Password { get; set; } // Пароль, по умолчанию "1"  
public string Database { get; set; } // Имя базы данных, по умолчанию "IBIS"  
public IrbisWorkstation Workstation { get; set; } // АРМ, по умолчанию «Каталогизатор»

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 демо-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 );

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:

public string Configuration { get; }

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 ); // Необходимо указать имя базы данных

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  
}

**Многопоточность и состояние занятости**

Клиент написан в наивном однопоточном стиле, поэтому не поддерживает одновременное выполнение команд из разных потоков. Во время обращения к серверу устанавливается флаг

public bool Busy { get; }

**Подтверждение подключения**

Библиотека самостоятельно не посылает на сервер подтверждений того, что клиент всё ещё подключен. Этим должно заниматься приложение, например, по таймеру.

Подтверждение посылается серверу методом NoOp:

public void NoOp ();

**Чтение/сохранение записей**

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

public IrbisRecord ReadRecord ( int mfn ); // Чтение одной записи по её MFN  
public IrbisRecord[] ReadRecords ( IEnumerable<int> mfns ); // Чтение массива записей с указанными MFN  
public void WriteRecord ( IrbisRecord record, bool needLock, bool ifUpdate ); // Сохранение одной записи  
 // needLock – оставить запись заблокированной  
 // ifUpdate – обновить инвертированный файл (поисковый индекс)  
public void WriteRecords ( IrbisRecord[] records, bool ifUpdate ); // Сохранение массива записей  
 // ifUpdate – обновить инвертированный файл (поисковый индекс)

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.

**Поиск записей**

Метод для поиска записей:

public int[] Search ( string format, params object[] args ); // возвращает массив MFN записей,   
 // удовлетворяющих запросу

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}$\"", authorName ); // поиск по соответствующему префиксу  
// обратите внимание на знак доллара, означающий усечение справа  
// и на кавычки, обрамляющие запрос

public IrbisRecord[] SearchRead ( string format, params object[] args ); // Возвращает записи,  
 // удовлетворяющие запросу. По факту совмещает методы Search и ReadRecord

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

Пример вызова:

foreach ( var record in client.SearchRead ( "J=CHI" ) ) // ищем книги на китайском языке  
{  
 // 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:

public string FormatRecord ( string format, int mfn ); // Форматирует запись по её MFN  
public string FormatRecord ( string format, IrbisRecord record ); // Форматирует заданную запись  
public string[] FormatRecords ( string format, int[] mfns ); // Форматирует массив записей по их MFN  
public string[] SearchFormat ( string expression, string format ); // Форматирует записи,   
 // удовлетворяющие поисковому запросу

Пример вызова:

foreach ( var description in client.SearchFormat ( "G=2000", "@brief" ) ) // Ищем книги 2000-года издания  
{  
 // Выводим их краткое биб. описание  
 Console.WriteLine ( description );  
}

**Классы IrbisRecord, RecordField и SubField**

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 содержит null  
public int Mfn { get; } // Номер записи в мастер-файле.  
 // Для вновь созданных записей равен 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

Запись хранит поля в списке Fields. Каждое поле соответствует экземпляру класса RecordField, который содержит следующие элементы:

public string Tag { get; set;} // Метка поля  
public string Text { get; set; } // Значение поля  
public string List<SubField> SubFields { get; } // Список подполей

Поле хранит подполя в списке SubFields. Каждое подполе соответствует экземпляру класса SubField, который содержит следующие элементы:

public char Code { get; set; } // Код подполя  
public string Text { get; set; } // Значение подполя

**Манипуляции с записями**

Получаем значение первого повторения поля (Field Match):

string language = record.FM ( "101" ); // Язык основного текста документа

Если такого поля нет, возвращается null.

Получаем массив всех повторений данного поля (Field Match All):

string[] keywords = record.FMA ( "610" ); // Все ключевые слова

Если таких полей нет, возвращается массив из 0 элементов.

Получаем значение первого повторения подполя:

string title = record.FM ( "200", 'a' ); // Основное заглавие

Если такого поля/подполя нет, возвращается null.

Получаем массив всех повторений данного подполя:

string[] subjects = record.FMA ( "606", 'a' ); // Все предметные заголовки

Если таких полей/подполей нет, возвращается массив из 0 элементов.

Выясняем, есть ли в записи хоть одно поле из перечисленных:

if (record.HaveField ( "606", "607" ))  
{  
 Console.WriteLine ( "Есть рубрики!");  
}

Проверяем отсутствие всех перечисленных полей:

if (record.HaveNotField ( "700", "701", "702" ))  
{  
 Console.WriteLine ( "Нет точек доступа на авторов!" );  
}

Удаляем все повторения указанного поля (если поля нет, ничего не происходит):

record.RemoveField ( "331" ); // Удалили аннотацию

Добавляем поле с текстом:

record.AddField ( "300", "Авторы указаны на корешке" );

Добавляем поле с подполями:

record.AddField ( "200", 'a', "Заглавие",  
 'e', "подзаголовочные сведения",  
 'f', "сведения об ответственности" );

Устанавливаем первое повторение поля с текстом (если поля с такой меткой нет, оно будет создано):

record.SetField ( "102", "RU" ); // Код страны

Обратите внимание, что вызовы AddField, SetField и RemoveField могут «цепляться» друг за друга:

record.RemoveField ( "331" )  
 .SetField ( "102", "RU" )  
 .AddField ( "300", "Авторы указаны на корешке" );

Конечно же, запись и все её поля/подполя могут быть созданы «с нуля» с помощью соответствующих конструкторов, после чего запись можно сохранить:

var record = new IrbisRecord (); // Создали запись  
  
// Создаём поле 900 и подполя в нём  
var field900 = new RecordField ( "900" ); // Коды: тип, вид, характер документа  
var sub900b = new SubField ( 'b', "05" ); // Однотомное издание  
var sub900t = new SubField ( 't', "a" ); // Текстовый материал  
field900.SubFields.Add ( sub900b ); // Добавили подполя в поле  
field900.SubFields.Add ( sub900t );  
record.Fields.Add ( field900 ); // Добавили поле в запись  
  
var field102 = new RecordField ( "102" ); // Страна  
field102.Text = "RU"; // Россия  
record.Fields.Add ( field102 );  
  
var field101 = new RecordField ("101"); // Язык основного текста  
field101.Text = "rus"; // Русский язык  
record.Fields.Add ( field101 );  
  
. . .  
  
// Сохраняем запись, блокировка записи нам не нужна,   
// зато нужно обновление поискового индекса  
client.WriteRecord ( record, false, true );

Надеюсь, этот короткий экскурс поможет освоить библиотеку-клиент ИРБИС64.