ООО "Научно-производственное предприятие "Мера"

Программное обеспечение сбора измерительных данных Recorder. Разработка plug-in`ов. Руководство программиста

г. Королёв 2004 г.

Содержание

1 Практическое руководство по созданию модуля plug-in` a

Это документ содержит рекомендации и описание процесса разработки plug-in`ов для ПО «Recorder». Для разработки plug-in`а можно использовать любой из инструментов: Borland® Delphi™, Borland® C++ Builder™, Microsoft ® Visual C++. Какой бы язык и инструмент не был выбран, принципы обращения с интерфейсами программирования ПО «Recorder» и логика работы остаются неизменными. Естественно, что для каждого инструмента разработки есть свои особенности использования, которые касаются специфичных свойств этих инструментов.

Наиболее полное и детальное описание процесса разработки приведено для Borland® Delphi, описание содержит участки кода и объяснение логики работы объектов ядра ПО «Recorder» и plug-in`а.

2 Инструментальная среда для разработки plug-in`а

2.1 Borland® Delphi

Инструментарий Borland® Delphi™ является удобным средством для быстрой разработки приложений. Delphi™ позволяет за короткие сроки создать программу с достаточно сложным многооконным пользовательским интерфейсом. Delphi™ поддерживает COM технологии, и это позволяет разрабатывать plug-in`ы для ПО «Recorder». Для разработки plug-in`ов необходимы файлы с описанием программных интерфейсов, эти файлы являются частью комплекта ПО для разработки plug-in`ов.

2.1.1 Особенности использования интерфейсов в среде Borland® Delphi™

В описании интерфейсов для Delphi, есть следующая особенность: все функции, возвращающие как результат ссылку на интерфейс, описаны как возвращающие pointer (не типизированный указатель). Соответственно в месте вызова, функции должны быть использованы с приведением типа. Список таких функций:

1. CreatePluginClass,
2. IRecorder::GetIFormByName,
3. IRecorder::GetIFormByIndex,
4. IRecorder::GetTagByName,
5. IRecorder::GetTagByIndex,
6. IRecorder::CreateTag,
7. IRecorder::GetModuleByIndex,
8. IRecorder::GetGroupByIndex,
9. IRecorder::GetTagIndexByPointer,
10. IRecorder::GetGroupByName,
11. IRecorder::CreateTagsGroup,
12. IRecorder::GetConfigModeInitiator,
13. ITag::GetData.

В описании этих функций, в комментариях, указан тип интерфейса, который функция возвращает. Причина подобного описания функций и интерфейсов – особенности компилятора Delphi™, которые описаны ниже.

2.1.1.1 Результат выполнения функций и COM интерфейсы

Функции, возвращающие ссылку на интерфейс, описанные в языке C и C++, возвращают результат через регистры процессора (точнее через EAX). Функции, возвращающие ссылку на интерфейс, описанные на языке Delphi™, возвращают результат через стек.

К примеру, пусть в Delphi™ описана функция:

```
function GetInterface: IUnknown;
```

Компилятор Delphi, в процессе компиляции программы, преобразует описание, и все обращения к этой функции так, как будто функция описана следующим образом:

```
procedure GetInterface( Result: IUnknown);
```

то есть передача результата производится через стек.

2.1.1.2 Параметры функций. Параметры по ссылке и по значению

В данном пункте рассматриваются функции и процедуры с «соглашением о вызовах» «stdcall». Параметры процедур и функций следующих типов: структуры, массивы (за исключением динамических), тип Variant и OleVariant передаются по значению через стек. То есть данные этих типов копируются в стек.

Константные параметры, описанных выше, типов передаются по ссылке. То сеть в стек копируются не сами данные, а ссылка на них.

К примеру, участок кода:

```
function SetProperty( const ID: integer; Data: Variant);
```

описывает функцию, параметры которой передаются по значению.

Участок кода:

```
function SetProperty( const ID: integer; const Data: Variant);
```

описывает функцию, которой параметр **Data** передается по ссылке.

2.1.2 Особенности процесса отладки динамически линкуемых библиотек

Для отладки динамически линкуемых библиотек в IDE Borland® Delphi™, в параметрах проекта библиотеки, необходимо указать имя программы «Host Application». Эта программа должна использовать библиотеку, вызывать те функции, которые необходимо отладить. Оболочка IDE Borland® Delphi™ будет запускать программу «Host Application» перед процессом отладки.

Однако, часто IDE Borland® Delphi™ допускает ошибку, и останов на отладочных точках останова не происходит, не смотря на то, что программа «Host Application» запущена и отлаживаемая библиотека используется. В этой ситуации необходимо сохранить файлы проекта, закрыть IDE Borland® Delphi™, открыть вновь и повторить попытку отладки.

2.2 Borland® C++ Builder™

Использование Borland® C++ Builder™ дает возможность производить быструю разработку приложений на C++, приложений с развитым оконным интерфейсом и богатой функциональностью. Особенности использования Borland® C++ Builder™ есть только в создании и настройке проекта программы.

2.3 Microsoft® Visual C++6.0

Microsoft® Visual C++ поддерживает технологию COM. При помощи Visual C++ могут быть созданы plug-in`ы как с использованием MFC так и без использования MFC. Соответственно plug-in может иметь или не иметь оконный интерфейс. Особенности использования Microsoft® Visual C++ заключаются в создании и настройке проекта программы plug-in`а.

3 Разработка plug-in`а при помощи Borland® Delphi™

В этом разделе будут даны рекомендации по разработке plug-in`а, и в качестве примера разработано несколько «простых» тестовых plug-in`ов.

Разработка производится по шагам, сначала описывается минимальный программный код plug-in`а, который только «запускается» и не выполняет никаких других функций. Далее plug-in дорабатывается для получения измеренных данных и отображения, для изменения настройки, для генерации сигналов и т.п.

Описание рассчитано на программиста, который знаком с языком и IDE Borland® Delphi™.

3.1 Минимальный код программы plug-in`а

Как описано в спецификации интерфейса PluginAPI модуль plug-in`а – это динамически линкуемая библиотека, которая экспортирует несколько функций и реализует класс plug-in` а.

3.1.1 Создание и настройка проекта

Для разработки plug-in`а необходимо:

1. создать проект динамически линкуемой библиотеки (dll). Новый проект создается при помощи Wizard`а, файл созданного проекта необходимо сохранить;
2. добавить в проект все интерфейсные файлы для разработки plug-in`а;
3. указать в качестве директории поиска («Search path») путь к интерфейсным файлам. Интерфейсные файлы, описывающие программный интерфейс PluginAPI, поставляются в составе пакета Plug-in SDK ПО «Recorder».

3.1.2 Описание класса plug-in`а

ПО «Recorder» работает с объектом plug-in`а. Библиотека plug-in`а должна описывать соответствующий класс и реализовывать его методы. В методах класса plug-in`а может быть реализована любая функциональность необходимая для решения поставленных задач. Класс plug-in`а может производить обработку, сохранение, отображение измеренных данных.

Таким образом, необходимо создать класс, пусть это будет класс с именем TTestPlugin. Данное имя не является обязательным, имя класса может быть любым (ограничением являются только ограничения на имена идентификаторов в Delphi™).

Для описания нового класса создадим в проекте новый модуль, назовем его «PluginClass.pas». Модуль должен использовать интерфейсные модули plug-in`а, как показано на примере.

```
uses
        Windows,
        recorder,
        plugin;
```

Класс TTestPlugin должен реализовать интерфейс IRecorderPlugin, по средствам этого интерфейса ПО «Recorder» взаимодействует с объектом plug-in`a. Так как IRecorderPlugin COM интерфейс, и он наследует IUnknown, то класс TTestPlugin должен реализовывать и интерфейс IUnknown. Реализацию IUnknown можно наследовать от класса TInterfacedObject. Таким образом, класс TTestPlugin наследует классу TInterfacedObject и интерфейсу IRecorderPlugin.

Участок кода с описанием класса TTestPlugin приведен ниже

```
TTestPlugin = class (TInterfacedObject, IRecorderPlugin)
public//IRecorderPlugin
        function _Create(pOwner: IRecorder): boolean; stdcall;
        function Config: boolean; stdcall;
        function Edit: boolean; stdcall;
        function Execute: boolean; stdcall;
        function Suspend: boolean; stdcall;
        function Resume: boolean; stdcall;
        function Notify(const dwCommand: DWORD;
        const dwData: DWORD): boolean; stdcall;
        function GetName: LPCSTR; stdcall;
        function GetProperty (const dwPropertyID: DWORD; var
                Value: OleVariant): boolean; stdcall;
        function SetProperty (const dwPropertyID: DWORD; {const}
                Value: OleVariant): boolean; stdcall;
        function CanClose: boolean; stdcall;
        function Close:  boolean; stdcall;
end;
```

В данном коде, для краткости, опущен ряд несущественных деталей, к примеру, комментарии, конструктор и деструктор класса.

В описании класса TTestPlugin приведен список методов, которые описаны в IRecorderPlugin и которые должны быть реализованы в TTestPlugin. На данном этапе код реализации этих методов можно оставить пустым, как это показано ниже, для одного из методов.

```
function TTestPlugin.Execute: boolean; stdcall;
begin
        Result: = true;
end;
```

В результате проект должен успешно компилироваться.

3.1.3 Создание объекта plug-in` а

В предыдущем пункте был описан класс plug-in`а, в данном случае это класс TTestPlugin. Однако ПО «Recorder» работает не с «самим классом», а с объектом. То есть необходимо создать экземпляр класса TTestPlugin. Экземпляр класса plug-in`а может быть создан в различных ситуациях:

1. В библиотеке создается экземпляр класса plug-in`а при загрузке, соответственно удаляется при выгрузке. Библиотека использует только один экземпляр класса. По запросу ПО «Recorder» всегда передается ссылка на уже созданный экземпляр.
2. Экземпляр класса plug-in`а может создаваться и удаляться по запросу ПО «Recorder».

Способ создания объекта plug-in`а выбирается в зависимости от требований к функциональности plug-in`а.

В данном примере в библиотеке создается один единственный экземпляр класса TTestPlugin, создается он при загрузке библиотеки и удаляется при выгрузке. Для хранения ссылки на объект plug-in`а описывается глобальная переменная GPluginInstance.

Код описания переменной и создания объекта выглядит следующим образом:

```
interface
…
GPluginInstance: IRecorderPlugin = nil;
…
implementation
…
initialization
        GPluginInstance:= TTestPlugin.Create;
end.
```

Приведен участок кода из модуля «PluginClass.pas».

В данной ситуации переменная GPluginInstance хранит ссылку не на сам объект, переменная хранит ссылку на интерфейс объекта.

В Delphi™ указатели на интерфейс «автоматизированы», увеличение, и уменьшение счетчика ссылок производится автоматически. То есть приведенный выше код можно описать следующим образом. В секции инициализации модуля производится создание объекта класса TTestPlugin, у объекта запрашивается ссылка на интерфейс IRecorderPlugin, полученная ссылка передается в переменную GPluginInstance. При присвоении ссылки переменной GPluginInstance, автоматически производится увеличение счетчика ссылок созданного ранее объекта.

Так как переменная GPluginInstance глобальная, то она уменьшает счетчик ссылок интерфейса только по выгрузке библиотеки, счетчик должен стать равным нулю и объект в этом случае удаляется. То есть объект plug-in`а удаляется по выгрузке библиотеки.

3.1.4 Информация о plug-in` е

ПО «Recorder» имеет возможность отобразить оператору информацию о библиотеке plug-in`а, это строка описания, строка наименования разработчика, номер версии. Для того, чтобы всю эту информацию отобразить ПО «Recorder» запрашивает ее у самого plug-in`а (см. [3] пп. А.1.1.5).

Библиотека тестового plug-in`а будет предоставлять описательную информацию. Для хранения упомянутой выше информации в проекте описана структура и глобальная переменная:

```
type
        TInternalPluginInfo = record
                Name:       string; //наименование plug-in`a
                Dsc:        string; //строка описания
                Vendor:     string; //строка наименования фирмы
                Version:    integer; //номер версии
                SubVertion: integer; //номер подверсии
        end;
```

```
const
        //Константа описания plug-in`a
        GPluginInfo: TInternalPluginInfo = (
                Name:       'Delphi Тест';
                Dsc:        'Тестирование';
                Vendor:     'ООО НПП Мера';
                Version:    0;
                SubVertion: 1;
        );
```

Таким образом, глобальная константа GPluginInfo хранит данные описывающие plug-in.

3.1.5 Описание экспортируемых функций

В библиотеке необходимо описать (экспортировать) и реализовать функции:

1. GetPluginType,
2. CreatePluginClass,
3. GetPluginDescription,
4. DestroyPluginClass,
5. GetPluginInfo,

в соответствии со спецификацией (см. [3] пп. А.1.1.). Функции предназначены для того, чтобы ПО «Recorder» могло получить доступ к объекту plug-in`a, а также получить информацию описывающую объект plug-in`a. Реализацию функций можно поместить непосредственно в код проекта библиотеки, либо можно создать новый модуль исходного кода.

Функция GetPluginType. Эта функция должна возвращать тип plug-in`a, константу PLUGIN_CLASS. Пример кода функции приведен ниже.

```
function GetPluginType: integer; cdecl;
begin
        Result:= PLUGIN_CLASS;
end;
```

Функция CreatePluginClass. Эта функция предназначена для создания экземпляра plug-in`a и передачи ссылки на экземпляр в ПО «Recorder». В данном проекте объект plug-in`a создается при инициализации библиотеки, в ПО «Recorder» передается ссылка на него.

```
function CreatePluginClass: pointer{IRecorderPlugin}; cdecl;
begin
        Result := pointer(GPluginInstance);
end;
```

Причина, по которой ссылка на интерфейс plug-in`а преобразуется к типу pointer, описана в пп. 1.1.1.1.

Функция GetPluginDescription. При помощи этой функции ПО «Recorder» запрашивает строку описания plug-in`а (см [3] пп. А.1.1.3). В реализации этой функции достаточно вернуть ссылку на строку описания. Код функции приведен ниже:

```
function GetPluginDescription: LPCSTR; cdecl;
begin
        Result:= LPCSTR(GPluginInfo.Dsc);
end;
```

Используется строка описания из структуры GPluginInfo.

Функция DestroyPluginClass. Эта функция предназначена для удаления объекта plug-in`а. В том случае если в функции **CreatePluginClass** экземпляр plug-in`а создавался, он должен удаляться в функции **DestroyPluginClass**, если он более нигде не используется.

В описываемом здесь примере объект удаляется автоматически, поэтому тело функции **DestroyPluginClass** пусто.

Функция GetPluginInfo. При помощи вызова данной функции ПО «Recorder» запрашивает полную информацию о plug-in`е (см [3] пп. А.1.1.5). Параметром функции передается адрес структуры PLUGININFO, эту структуру необходимо заполнить. Вся требуемая информация хранится в константной переменной GPluginInfo**.**

Код функции приведен ниже

```
procedure GetPluginInfo(var lpPluginInfo: PLUGININFO); cdecl;
begin
        StrCopy( @lpPluginInfo.name, LPCSTR(GPluginInfo.Name));
        StrCopy( @lpPluginInfo.describe, LPCSTR(GPluginInfo.Dsc));
        StrCopy( @lpPluginInfo.vendor,
                LPCSTR(GPluginInfo.Vendor));
        lpPluginInfo.version:= GPluginInfo.Version;
        lpPluginInfo.subversion:= GPluginInfo.SubVertion;
end;
```

В данном коде в структуру по адресу из lpPluginInfo копируются строки имени plug-in`а, описания и наименования фирмы разработчика. Необходимо удостовериться, что строки имеют длину не более, указанных размеров в описании структуры PLUGININFO.

В завершении необходимо описать функции как экспортные:

```
exports GetPluginType          name 'GetPluginType';
exports CreatePluginClass      name 'CreatePluginClass';
exports DestroyPluginClass     name 'DestroyPluginClass';
exports GetPluginDescription   name 'GetPluginDescription';
exports GetPluginInfo          name 'GetPluginInfo';
```

3.1.6 Компиляция, запуск и отладка

На данном этапе создан проект с минимальным необходимым кодом для запуска plug-in`a. Данный код является «скелетом», на основе которого можно создать любой по функциональности plug-in. Проект следует скомпилировать.

Для запуска plug-in`a под отладчиком IDE Borland® Delphi™ необходимо:

1. указать полный путь и имя файла plug-in`a в файле настройки ПО «Recorder». Файл настройки (по умолчанию) «recorder.cfg» находится в корневой директории ПО «Recorder».
2. указать в IDE Borland® Delphi™ в параметрах запуска в качестве «Host Application» имя (и соответственно полный путь) программы «Recorder».

Для того чтобы убедиться, что ПО «Recorder» «вызывает» plug-in можно запустить режим отладки, предварительно установив несколько отладочных точек останова, к примеру, в теле методов: TTestPlugin._Create, TTestPlugin.Execute. После запуска режима отладки IDE Borland® Delphi™ укажет, что выполнение программы остановилось в теле метода TTestPlugin._Create, а за тем и в теле метода TTestPlugin.Execute.

Если останов в отладочных точках останова не произошел, то необходимо убедиться в правильной настройке IDE Borland® Delphi™ и корректности настройки ПО «Recorder», проверить содержимое файла «recorder.cfg» и убедиться, что имя библиотеки plug-in`a указано правильно. Так же следует обратить внимание на особенности отладки, описанные в пп. 2.1.2.

3.2 Plug-in с функциями обработки и отображения данных

3.2.1 Постановка задачи

В качестве примера необходимо разработать тестовый plug-in, который реализует следующие функции:

1. plug-in периодически получает измеренные данные всех тегов и отображает в специализированном окне;
2. в окне plug-in`a отображается в табличном виде список имен всех тегов, строки описания всех тегов, строковые представления измеренных данных, значение штампа времени для измеренных данных;
3. период получения и отображения измеренных данных должен быть не более 500 мс.
4. plug-in должен позволять управлять ПО «Recorder», производить запуск режима измерения и останов.

3.2.2 Функциональность plug-in` a

В качестве основы для разработки нового plug-in`a можно использовать код, описанный в пп. 3.1.

Дальнейшая доработка библиотеки и самого класса plug-in`а должна производится в соответствии с теми требованиями, которые предъявляются к функциональности plug-in`а. Основная задача, которую должен выполнять, описанный в этом разделе plug-in, это отображение в отдельном окне, в табличном виде, измеренных значений всех тегов.

3.2.3 Создание формы для отображения данных

Для отображения измеренных данных необходимо создать в проекте новую форму. Пусть новая форма называется TTestForm, модуль, в котором описана и реализована форма, пусть называется «TestFormUnit.pas».

На форме необходимо расположить объект TagsListView - экземпляр компоненты типа TListView, этот объект будет использоваться в качестве таблицы для отображения данных. Тип отображения для TagsListView – ViewStyle устанавливается в vsReport (табличное отображение). В объекте TagsListView создаются 4 колонки: «Наименование», «Описание», «Значение», «Штамп времени». Колонка «Наименование» предназначена для отображения списка имен тегов. Колонка «Описание» предназначена для отображения списка строк описаний тегов. Колонка «Значение» предназначена для отображения измеренных данных тегов. Если измеренные данные тега – это одно значение, то будет отображаться одно значение. Если измеренные данные тега – это массив, то будет производиться вычисление и отображение среднего из массива данных. Колонка «Штамп времени» предназначена для отображения времени, когда измеренные данные были получены.

Содержимое таблицы необходимо периодически обновлять. Обновление таблицы, то есть получение измеренных данных из тегов, обработку и отображение проще производить по таймеру. На форме TTestForm необходимо расположить объект DataUpdateTimer – экземпляр компоненты TTimer. Объект настраивается на интервал 500 миллисекунд (то есть обновление будет производиться 2 раза в секунду). Создадим отдельный метод, в котором будет производиться получение данных из тегов, обработка и отображение в таблице – это метод формы DataUpdate. Метод обработчик события от таймера DataUpdateTimerTimer будет только вызывать метод формы DataUpdate.

Очевидно, что форма TTestForm должна один раз создаваться plug-in` ом, когда создается сам plug-in и удаляться соответственно, когда удаляется plug-in. Plug-in должен хранить ссылку на объект формы. Описание класса plug-in`а дополняется следующими строками:

```
TTestPlugin = class(TInterfacedObject, IRecorderPlugin)
…
protected
        FTestForm: TTestForm;
public
        constructor Create;
        destructor Destroy; override;
…
end;
```

В данном участке кода опущены приведенные ранее (см. пп. 3.1.2.) методы и показаны только те методы и поле, которые добавились.

В конструкторе plug-in`а производится создание формы, а в деструкторе - удаление.

```
constructor TTestPlugin.Create;
begin
        FTestForm:= TTestForm.Create( nil);
end;
destructor TTestPlugin.Destroy;
begin
        FreeAndNil( FTestForm);
end;
```

3.2.4 Получение и хранение списка тегов

Доступ plug-in`а к измеренным данным возможен через объекты ядра ПО «Recorder» - теги. То есть plug-in должен получить и хранить список объектов тегов. Для получения списка тегов используется объект Recorder. Ссылку на интерфейс IRecorder, plug-in получает в методе IRecorderPlugin._Create(), эту ссылку необходимо сохранить.

Для получения списка тегов следует использовать методы: IRecorder.GetTagsCount, IRecorder.GetTagByIndex.

Наиболее удобный способ хранения списка тегов в plug-in`е – это динамический массив. Метод получения списка тегов, и формирования динамического массива со списком тегов, приведен ниже.

```
procedure TTestPlugin.GetDynTagsArray(var Tags: DynTagsArray);
var
    Count: integer;
    i: integer;
begin
    //получить кол-во тегов
    Count:= FIRecorder.GetTagsCount;
    //установить соответствующий размер динамического массива
    SetLength(Tags, Count);
    //последовательно получить ссылки на интерфейсы тегов
    for i:= 0 to Count-1 do
    //использование преобразования типа обосновано в описании
        Tags[i]:= ITag(FIRecorder.GetTagByIndex(i));
end;
```

Тип DynTagsArray – это динамический массив ссылок на объекты тегов, его описание приведено ниже.

3.2.5 Реакция на изменение настройки ПО «Recorder»

В процессе запуска и работы ПО «Recorder» сменяет ряд состояний. В моменты изменения настройки ПО «Recorder» находится в состоянии конфигурирования. В процессе настройки могут быть добавлены или удалены устройства, созданы или удалены теги, загружены или выгружены plug-in`ы. Так как после изменения настройки список тегов

может меняться, то каждый plug-in должен обновлять собственный внутренний список ссылок на теги, после каждого изменения настройки ПО «Recorder». В процессе настройки доступ к тегам запрещен!

Для того, чтобы plug-in мог «знать» о начале и завершении режима настройки (конфигурирования) ПО «Recorder» рассылает всем plug-in`ам сообщения. ПО «Recorder» вызывает метод IRecorderPlugin.Notify, с кодом сообщения:

PN_ENTERRCCONFIG – начало режима настройки,

PN_LEAVERCCONFIG – режим настройки завершен.

Очевидно, что, получив сообщение о входе в режим настройки, plug-in должен перестать использовать теги, если он раньше это делал. Получив сообщение о выходе из режима настройки, plug-in должен обновить внутренний список ссылок на теги и продолжить свою штатную работу. Реализация метода TTestPlugin.Notify тестового plug-in`а приведена ниже.

Так как инициация процесса получения обработки и отображения данных происходит в форме TTestForm, по таймеру, и измеренные данные необходимы только для отображения, то целесообразно хранить список ссылок на теги в классе формы. Однако, учитывая ограничения, описанные выше, форма должна останавливать процесс чтения данных на время изменения настройки, и список тегов должен обновляться, по завершению настройки.

Таким образом, необходимо создать два метода формы TTestForm.BeginConfigure и TTestForm.EndConfigure. Эти методы будет вызывать объект plug-in`а, «сообщая» форме о начале и завершении процесса настройки. TTestForm.BeginConfigure вызывается, когда plug-in получает оповещение о начале режима настройки. По завершении режима настройки plug-in запрашивает список тегов (возможно измененный) у Recorder`а и вызывает метод формы TTestForm.EndConfigure, список тегов передается форме.

В методе TTestForm.BeginConfigure форма должна остановить таймер обновления. В методе TTestForm.EndConfigure форма получает обновленный список тегов, сохраняет его во внутренних полях и запускает таймер обновления.

В методе TTestForm.EndConfigure вызывается специальный метод TTestForm.PrepareTagsView, он необходим для следующего:

1. для того, чтобы установить правильное, в соответствии с количеством тегов, количество строк таблицы с данными;
2. для обновления в таблице первой колонки с именами тегов.

```
function  TTestForm.BeginConfigure: boolean;
begin
    DataUpdateTimer.Enabled := false; //остановить таймер обновления
    SetLength(FTags, 0); //очистить список внутренний тегов
    Result := true;
end;
```

```
procedure TTestForm.EndConfigure( const Tags: DynTagsArray);
var i: integer;
begin
    //обновление внутреннего списка тегов
    SetLength(FTags, Length(Tags));
    for i := Low(Tags) to High(Tags) do
    begin
        FTags[i] := Tags[i];
    end;
    //обновление списка отображающего теги (имена, описания и данные)
    PrepareTagsView;
    //активизация таймера для отображения данных
    DataUpdateTimer.Enabled := true;
end;
```

Метод обработки сообщений (событий) от ПО «Recorder» TTestPlugin.Notify.

```
function TTestPlugin.Notify(const dwCommand: DWORD;
                            const dwData: DWORD): boolean; stdcall;
var
    Tags: DynTagsArray;
begin
    case dwCommand of
        PN_ENTERRCCONFIG: begin // перешел в режим настройки
            {В то время пока производится изменение настройки,
            теги недоступны (потому, что они могут удаляться)}
            //сообщить форме о начале изменения настройки
            TTestForm(FTestForm).BeginConfigure;
            Result:= true;
        end;
        PN_LEAVERCCONFIG: begin // вышел из режима настройки
            {Настройка завершена, и необходимо обновить список тегов}
            GetDynTagsArray(Tags); {получить массив тегов}
            {Оповестить форму о завершении изменения конфигурации и
            передать ей новый список ссылок на интерфейсы тегов}
            TTestForm(FTestForm).EndConfigure(Tags);
            Result:= true;
        end;
    else
        Result:= false; //прочие сообщения не обрабатываем
    end;
end;
```

3.2.6 Получение данных

В методе TTestForm.DataUpdate – это метод получения и отображения измеренных данных, выполняется обработка списка тегов в цикле, для каждого тега выполняется за-

прос измеренных данных и отображение данных в объекте TagsListView. Код метода приведен ниже.

```
procedure TTestForm.DataUpdate;
var
    i: integer;
    li: TListItem;
    StrData: string;
    StrTimeStamp: string;
begin
    for i:= Low(FTags) to High(FTags) do //цикл по всему списку тегов
    begin
        //проверка корректности таблицы TagsListView и
        //корректности ссылки на тег
        if (TagsListView.Items.Count > i) and Assigned(FTags[i]) then
        begin
          //получение ссылки на элемент таблицы
          li:= TagsListView.Items[i];
          //получение строкового представления измеренных
          //данных тега и штампа времени
          StrData:= GetDataThroughVector(FTags[i], StrTimeStamp);

          //установка данных в таблицу для отображения
          if li.SubItems.Count >= 2 then
                li.SubItems[1] := StrData;
          if li.SubItems.Count >= 3 then
                li.SubItems[2] := StrTimeStamp;
        end;
    end;
end;
```

По каждому тегу измеренные данные запрашиваются отдельно, получение данных у тега производится в методе GetDataThroughVector.

Метод GetDataThroughVector выполняет чтение измеренных данных тега, при необходимости вычисление среднего, преобразование данных в строку, получение временного штампа.

Для получения данных у тега используется блочный доступ и соответственно интерфейс блочного доступа IBlockAccess. Интерфейс блочного доступа реализует объект тега. В методе GetDataThroughVector интерфейс блочного доступа каждый раз запрашивается у тега, более эффективно, конечно же было бы, получить один раз и запомнить интерфейс блочного доступа, вместе с получением интерфейса тега ITag.

Перед тем как начать работу с данными необходимо блокировать буфер измеренных данных, далее получить размер блока, номер последнего блока с измеренными данными, подготовить внутренний буфер для чтения блока данных и произвести чтение блока во внутренний буфер. Размер блока может быть разным, он равен произведению частоты дискретизации тега и периода обновления данных ПО «Recorder». Для блоков, чей размер больше одного вычисляется среднее значение, результат преобразуется в строку. Временной штамп получения блока с измеренными данными получается при помощи метода GetTimeStamp. Метод GetTimeStamp использует ссылку на интерфейс блочного доступа и номер последнего блока, результат – строка с временным штампом. После получения данных необходимо буфер с измеренными данными разблокировать.

```
function TTestForm.GetDataThroughVector(Tag: ITag;
                                var StrTimeStamp: string): string;
var
    IBlock: IBlockAccess;   //интерфейс блочного доступа к данным
    bl_size: integer;       //размер одного блока данных
    bl_count: integer;      //кол-во блоков данных
    data: array of double;  //динамический массив для чтения блока
    i : integer;
    aod: double;            //вычисляемое средее арифметическое
begin
    //получить по ссылке на тег, ссылку на
    //интерфейс блочного доступа
    Tag.QueryInterface( IBlockAccess, IBlock);
    //блокировать буфер с измеренными данными, это обязательное
    //действие; так как буфер постоянно пополняется измеренными
    //данными – необходимо остановить изменение вектора на
    //время отображения данных.
    IBlock.LockVector;
    try
        //получить размер блока
        bl_size:=  IBlock.GetBlocksSize;
        //получить кол-во блоков
        bl_count:= IBlock.GetBlocksCount;
        //проверка
        if (bl_count <> 0) and (bl_size <> 0) then
        begin
            {при необходимости....}
            if (Length(data) <> bl_size) then
                SetLength(data, bl_size); {изменение размера массива данных}
            //чтение последнего блока измеренных данных
            IBlock.GetVectorR8( (pointer(data))^, bl_count - 1, bl_size, true);

            //вычисление среднего арифметического
            aod:= 0;
            for i:= 0 to bl_size - 1 do
                aod := aod + data[i];
            aod:= aod / bl_size;

            //формирование строки со значением
            Result := FloatToStr( aod);
            //получить временной штамп текущих данных
            StrTimeStamp:= GetTimeStamp(IBlock, bl_count - 1);
        end
        else
            Result:= '';
    finally
        //разблокировать вектор
        IBlock.UnlockVector;
    end;
end;
```

В данном тестовом plug-in`е предполагается, что аппаратная конфигурация допускает получение времени СЕВ (временные штампы для каждого блока измеренных данных).

```
function TTestForm.GetTimeStamp( IBlock: IBlockAccess;
                        const nblock: integer): string;
type
    FTR = record
        case integer of
            0: (itime: int64);
            1: (ftime: FILETIME);
        end;
var
    CTS: double;
    temp_time: FTR;
    systime: SYSTEMTIME; //преобразовать время в необходимый формат
begin
    CTS := IBlock.GetBlockUTSTime( nblock);
    try
        CTS := CTS * 1000;
        temp_time.itime := Round( CTS);
        temp_time.itime := temp_time.itime * 10000;
        FileTimeToSystemTime( temp_time.ftime, systime);
        //и в строку для отображения в таблице
        Result:= TimeToStr (SystemTimeToDateTime( systime));
    except
    end;
end;
```

Метод TTestForm.GetTimeStamp запрашивает по средствам интерфейса блочного доступа временной штамм СЕВ для последнего блока и преобразует временной штамп в тип SYSTEMTIME, а из типа SYSTEMTIME в строковое представление. Метод в качестве результата возвращает строковое представление временного штампа СЕВ.

Для успешной компиляции модуля с кодом формы TTestForm необходимо добавить использование интерфейсных файлов и описание типа, как указано на примере ниже.

```
uses
    …
    tags, blaccess;

type
    {Тип динамического массива ссылок на теги }
    DynTagsArray = array of ITag;
```

В приведенном выше, участке кода описан тип динамического массива ссылок на теги, этот тип необходим для описания переменной списка тегов в классе формы.

3.2.7 Отображение формы с данными

В пп. 3.2.3 описано, как следует создавать объект формы, однако, кроме создания, форму необходимо еще и отобразить. Отображение формы следует производить в методе TTestPlugin.Execute. Этот метод вызывается ПО «Recorder» для запуска штанного режима работы plug-in`a. Пример кода метода TTestPlugin.Execute приведен ниже.

```
function TTestPlugin.Execute: boolean;
begin
    FTestForm.Show(); // отображение формы
    Result:= true;
end;
```

ПО «Recorder» может вызывать методы plug-in`а IRecorderPlugin.Notify и IRecorderPlugin.Edit для предоставления возможности plug-in`у отображать собственные окна настройки.

Метод IRecorderPlugin.Edit вызывается, когда оператор нажимает кнопку «Свойства» в окне настройки plug-in`ов ПО «Recorder».

Метод IRecorderPlugin.Notify с кодом сообщения PN_SHOWINFO вызывается, когда оператор нажимает кнопку «Инфо» в окне настройки plug-in`ов ПО «Recorder».

Так как тестовый plug-in имеет только одно окно – окно для отображения данных, то именно это окно будет всегда отображаться.

```
function TTestPlugin.Edit: boolean;
begin
    FTestForm.Show();// отображение формы
    Result:= true;
end;

function TTestPlugin.Notify(const dwCommand: DWORD; const dwData:
                                        DWORD): boolean;
…
begin
    case dwCommand of
    …
    …
        PN_SHOWINFO: begin
            FTestForm.Show();    // отображение формы
            Result:= true;
        end;
    else
        Result:= false; //прочие сообщения не обрабатываем
    end;
end;
```

Выше приведена только часть кода метода TTestPlugin.Notify, та часть, которая касается обработки сообщения PN_SHOWINFO.

Специализированный plug-in может активизировать отдельное окно для отображения оператору описательной информации о модуле plug-in`а.

3.2.8 Строка описания. Возврат plug-in`ом строки описания

Необходимо отметить, что после того, как объект plug-in`а создан ПО «Recorder» пытается получить описание plug-in`а не у библиотеки, а у самого объекта.

Для получения имени plug-in`а используется метод IRecorderPlugin.GetName и метод IRecorderPlugin.GetProperty.

```
function TTestPlugin.GetName: LPCSTR;
begin
    {Из глобальной структуры описания plug-in`a}
    Result:= LPCSTR(GPluginInfo.Name);
end;

function TTestPlugin.GetProperty(const dwPropertyID: DWORD;
                                 var Value: OleVariant): boolean;
begin
    case dwPropertyID of
        {Получить строку описания plug-in`a}
        PLGPROP_INFOSTRING: begin
            {Из глобальной структуры описания plug-in`a}
            Value := GPluginInfo.Dsc;
            Result:= true;
        end;
    else
        Result:= false;
    end;
end;
```

Последние дополнения к коду тестового plug-in`a позволяют увидеть строку имени и строку описания plug-in`a в окне настройки plug-in`ов ПО «Recorder».

3.2.9 Стандартные «оценки», получение обработанных измеренных данных

ПО «Recorder» может выполнять автоматически оценку измеренных данных при помощи ряда стандартных алгоритмов.

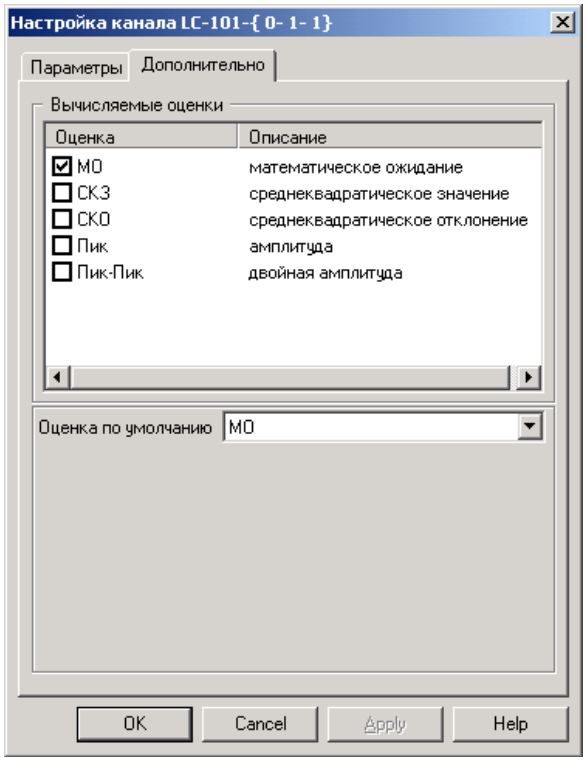

Рисунок 1

Список активных алгоритмов, по которым выполняется обработка измеренных данных, указан в свойствах каждого тега.

На рисунке 1 изображено окно свойств тега с именем «LC-101 – {0-1-1}». Для этого тега ПО «Recorder» автоматически вычисляет оценку «МО». Оценка вычисляется сразу при получении измеренных данных из драйвера измерительного устройства.

Получить оценку можно несколькими способами:

1. Получить значение свойства тега, код свойства TAGPROP_ESTIMATE. Для получения свойства используется метод ITag.GetProperty. Метод возвращает значение «оценки по умолчанию».
2. Получить значение оценки при помощи метода ITag.GetEstimate, параметром метода указывается тип оценки, значение который необходимо получить.

Легко изменить plug-in для того, чтобы он не производил обработку данных сам, а использовал оценки, которые рассчитаны ПО «Recorder». Для этого необходимо следующее:

1. создать новый метод формы TTestForm.GetDataThroughAOD, который параметром получает ссылку на тег, производит получение оценки измеренных данных для тега и возвращает значение оценки в виде строки;
2. изменить код метода формы TTestForm.DataUpdate, чтобы в нем вызывался не метод TTestForm.GetDataThroughVector, а метод TTestForm.GetDataThroughAOD.

Для того, чтобы оставить возможность использования и кода метода TTestForm.GetDataThroughVector, и кода метода TTestForm.GetDataThroughAOD вызов методов можно оформить в соответствии с условиями компиляции. Ниже приведен пример реализации метода TTestForm.DataUpdate.

```
procedure TTestForm.DataUpdate;
var
    i: integer;
    li: TListItem;
    StrData: string;
    StrTimeStamp: string;
begin
    //цикл по всему списку тегов
    for i:= Low(FTags) to High(FTags) do
    begin
        //проверка корректности таблицы TagsListView и
        //корректности ссылки на тег
        if (TagsListView.Items.Count > i) and Assigned(FTags[i]) then
        begin
            //получение ссылки на элемент таблицы
            li:= TagsListView.Items[i];
{$IFDEF DATATHROUGHVECTOR}
            //получение строкового представления измеренных
            //данных тега и штампа времени
            StrData:= GetDataThroughVector(FTags[i], StrTimeStamp);
{$ELSE}
            StrData:= GetDataThroughAOD( FTags[i]);
            StrTimeStamp := TimeToStr(Now);
{$ENDIF}
            //установка данных в таблицу для отображения
            if li.SubItems.Count >= 2 then
                li.SubItems[1] := StrData;
            if li.SubItems.Count >= 3 then
                li.SubItems[2] := StrTimeStamp;
        end;
    end;
end;
```

В том случае если в программе определен символ «DATATHROUGHVECTOR», то будет использоваться блочный способ получения измеренных данных, иначе будет использоваться оценка.

Ниже приведен пример кода получения оценки.

```
function TTestForm.GetDataThroughAOD(Tag: ITag): string;
var vData: OleVariant;
begin
    //получение среднего арифметического как одного из свойств тега
    if Tag.GetProperty(TAGPROP_ESTIMATE, vData) then
        Result:= vData
    else
        Result:= '';
end;
```

3.2.10 Запуск и останов режима измерения ПО «Recorder»

Запуск и останов режима измерения выполняется передачей объекту Recorder управляющих сообщений RCN_VIEW и RCN_STOP соответственно. Передача управляющих сообщений выполняется методом IRecorder.Notify.

Так как ссылку на интерфейс IRecorder хранит объект plug-in`a, то целесообразно в объекте plug-in`a реализовать два метода: метод «запуска» TTestPlugin.StartMeasure и метод останова TTestPlugin.StopMeasure.

```
function TTestPlugin.StartMeasure: boolean;
begin
    {Для запуска необходимо передать сообщение Recorder`y}
    Result:= FIRecorder.Notify( RCN_VIEW, 0);
end;

function TTestPlugin.StopMeasure: boolean;
begin
    {Для останова необходимо передать сообщение Recorder`y}
    Result:= FIRecorder.Notify( RCN_STOP, 0);
end;
```

Очевидно, что запуск и останов режима измерения должен выполняться по запросу оператора, то есть на форме plug-in`a необходимо расположить кнопки SBtnStop и SBtnView. По нажатию кнопки SBtnStop должен вызываться метод формы TTestForm.SBtnStopClick, по нажатию кнопки SBtnView должен вызываться метод формы TTestForm.SBtnViewClick. В свою очередь TTestForm.SBtnStopClick должен вызывать метод plug-in`a TTestPlugin.StopMeasure, а TTestForm.SBtnViewClick – метод TTestPlugin.StartMeasure.

Для того, чтобы получить доступ к объекту plug-in`a необходимо, чтобы форма хранила ссылку на объект plug-in`a. То есть и форма должна «знать» о plug-in`e и plug-in должен «знать» о форме. Во избежание рекурсивного обращения модулей друг к другу модуля «PluginClass.pas», и модуля «TestFormUnit.pas», необходимо выполнить следующее:

1. в секции implementation модуля «TestFormUnit.pas» должно быть объявлено об использовании модуля «PluginClass.pas»;
2. форма должна хранить, ссылку на plug-in в переменной типа TObject, и при использовании ссылки делать необходимое преобразование типа;

3. при создании формы plug-in должен передавать форме ссылку на себя, для этого следует переопределить конструктор формы.

Участки кода, иллюстрирующий конструктор формы, описание и реализацию

```
TTestForm = class(TForm)
public
…
…
    constructor Create( const Plugin: TObject); reintroduce;
private
    FPlugin: TObject;
…
…
end;
```

```
constructor TTestForm.Create( const Plugin: TObject);
begin
    inherited Create ( nil); //вызов метода (конструктора) предка
    FPlugin:= Plugin; //сохранение ссылки на объект plug-in`a
end;
```

Код конструктора plug-in`a.

```
constructor TTestPlugin.Create;
begin
    FTestForm:= TTestForm.Create( self );
end;
```

Следует обратить внимание, что код конструктора изменился, ранее при создании формы plug-in передавал аргументом нулевую ссылку, теперь передает ссылку на себя!

Ниже приведены методы обработки нажатия кнопок запуска и останова режима измерения

```
procedure TTestForm.SBtnStopClick(Sender: TObject);
begin
  TTestPlugin(FPlugin).StopMeasure;
end;

procedure TTestForm.SBtnViewClick(Sender: TObject);
begin
  TTestPlugin(FPlugin).StartMeasure;
end;
```

3.2.11 Компиляция и отладка

Процесс создания plug-in`а завершен. Полученный код должен компилироваться. Полная распечатка программного кода приведена в приложении.

3.3 Специализированные формуляры отображения ПО «Recorder»

ПО «Recorder» содержит несколько различных формуляров для отображения измеренных данных: табличный формуляр, формуляр с одной осциллограммой, двумя и т.д. Модули plug-in`ы имеют возможность создавать собственные формы, регистрировать их в качестве формуляров и тогда ПО «Recorder» будет отображать эти формуляры так же как встроенные. Формуляр plug-in`а появится в списке доступных формуляров, и ПО «Recorder» будет отображать этот формуляр в собственном окне.

3.3.1 Постановка задачи

В качестве примера необходимо разработать тестовый plug-in, который реализует следующие функции:

1. plug-in периодически получает измеренные данные всех тегов и отображает в специализированном окне - формуляре;
2. в окне plug-in`а отображается в табличном виде список имен всех тегов, строки описания всех тегов, строковые представления измеренных данных, значение штампа времени для измеренных данных;
3. период получения и отображения измеренных данных должен быть не более 1с.
4. Формуляр plug-in`а должен быть доступен (отображаться) из главного окна ПО «Recorder».

По функциям данный plug-in похож на plug-in, описанный в пп. 3.2. Используем в качестве основы код, описанный в пп.3.2. В данный plug-in необходимо лишь добавить код поддержки и регистрации формуляра.

3.3.2 Формуляр для отображения данных

Для того, чтобы ПО «Recorder» могло работать с формуляром plug-in`а, этот формуляр необходимо зарегистрировать. В модуле plug-in`а должен быть реализован интерфейс IVForm. В ПО «Recorder» должна быть передана ссылка на IVForm при помощи метода IRecorder::RegisterIForm.

Интерфейс IVForm может быть реализован отдельным объектом, а может быть реализован объектом plug-in`а. Для данного примера пусть IVForm реализуется объектом plug-in`а. В описании класса добавляется ряд строк, участок кода приведен ниже.

Следует обратить внимание на метод GetName. Метод с таким именем есть и в интерфейсе IRecorderPlugin и в интерфейсе IVForm. Для того, чтобы у интерфейса IVForm была собственная реализация этого метода необходимо описать «псевдоним» IVForm_GetName. То есть, в описании класса явно указано, что реализация метода IVForm.GetName выполняется при помощи метода с именем IVForm_GetName. Далее в описании класса упомянут уже сам метод IVForm_GetName. Аналогично в классе plug-in`а описаны методы IVForm.Close, IVForm.Edit, IVForm.Notify. Подробности правил разрешения имен методов смотрите в руководстве Borland® Delphi™.

```
TTestPlugin = class(TInterfacedObject, IRecorderPlugin, IVForm)
…
public //IVForm
    //Получить имя формы
    function IVForm.GetName = IVForm_GetName;
    //Инициализация формы
    function Init(pParent: IRecorder; hParent: HWND;
                        lParam: longint): boolean; stdcall;
    //Получить HWND формы
    function GetHWnd: HWND; stdcall;
    //Вызывается при закрытии формы
    function IVForm.Close = IVForm_Close;
    function Prepare: boolean; stdcall;
    function Update: boolean; stdcall;
    //Перерисовка формы
    function Repaint: boolean; stdcall;
    //Привязка к тегам рекордера
    function LinkTags( var pTagsList: TagsArray;
                        var nTagsCount: ULONG): boolean; stdcall;
    //Активизация формы
    function Activate: boolean; stdcall;
    //Деактивизация формы
    function Deactivate: boolean; stdcall;
    //Вызов окна редактирования
    function IVForm.Edit = IVForm_Edit;
    //События, уведомления, команды
    function IVForm.Notify = IVForm_Notify;
protected
    //Получить имя формы
    function IVForm_GetName: LPCSTR; stdcall;
    //Вызывается при закрытии формы
    function IVForm_Close: boolean; stdcall;
    //Вызов окна редактирования
    function IVForm_Edit: boolean; stdcall;
    //События, уведомления, команды
    function IVForm_Notify(const dwCommand: DWORD;
                const dwData: DWORD): boolean; stdcall;
…
end;
```

В качестве формуляра для отображения данных будет использоваться форма (см. пп. 3.2.3). предыдущего plug-in`a. Свойства формы необходимо изменить таким образом, чтобы форма осталась без оконной рамки – свойству формы «BorderStyle» установить значение «bsNone». Отображение оконной формы необходимо выполнять только по командам от ПО «Recorder».

Метод IVForm.GetName должен возвращать адрес строки имени формуляра, имя должно отличаться от имен дрегих формуляров. В данной ситуации имя формуляра может соответствовать имени plug-in`a. То есть метод IVForm.GetName возвращает адрес строки имени в структуре GPluginInfo.

Метод IVForm.Init. Этот метод ПО «Recorder» вызывает для инициализации формуляра и для передачи формуляру числового идентификатора главного окна. Класс plug-

in`a должен настроить свой собственный формуляр так, чтобы главное окно ПО «Recorder» являлось для него родительским. Код, выполняющий это, приведен ниже.

```
function TTestPlugin.Init(pParent: IRecorder; hParent: HWND;
                          lParam: longint): boolean;
begin
        FTestForm.ParentWindow := hParent;
        Result:= true;
end;
```

В этом коде FTestForm – это экземпляр формы – формуляр отображения измеренных данных.

Метод IVForm.GetHWnd должен возвращать идентификатор окна формуляра. Идентификатор окна формуляра – идентификатор окна формы FTestForm.

Метод IVForm.Close вызывается ПО «Recorder» перед удалением формуляра (перед закрытием plug-in`a), в этом методе можно выполнять код по освобожению ресусов, если таковые были выделены в коде инициализации фомуляра.

Методы IVForm.Update, IVForm.Edit и IVForm.LinkTags зарезервированы и в данный момент не используются.

Методы IVForm.Prepare и IVForm.Repaint периодически вызываются ПО «Recorder». Период вызова соответствует периоду обновления, указанному в настройке. Эти методы предназначены для того, чтобы plug-in и формуляр соответственно, имели возможность подготовить и обновить данные, которые формуляр отображает. В данном примере отображение производится по таймеру, поэтому тела методов IVForm.Prepare и IVForm.Repaint пусты.

Метод IVForm.Activate ПО «Recorder» использует, когда необходимо, чтобы формуляр стал видимым. Метод IVForm.Deactivate ПО «Recorder» использует, когда необходимо, чтобы формуляр перестал быть видимым. В методе IVForm.Activate необходимо показать форму FTestForm (метод формы Show), а в методе IVForm.Deactivate – скрыть форму FTestForm (метод формы Hide).

Метод IVForm.Notify. При помощи этого метода ПО «Recorder» передает в формуляр команды и оповещает о событиях. Формуляр должен обрабатывать команду с кодом VSN_RESIZE – команду на изменение размеров окна формуляра. Новые координаты окна формуляра можно получить через аргумент метода IVForm.Notify. Пример кода метода IVForm.Notify приведен ниже.

```
function TTestPlugin.IVForm_Notify(const dwCommand: DWORD;
                                   const dwData: DWORD): boolean;
var {Временная переменная для интерпретации параметра dwData}
    pRect: ^TRect absolute dwData; {как ссылки на структуру TRect}
begin
    case dwCommand of {Ветвление по коду команды (сообщения)...}
        VSN_RESIZE: begin {команда на изменение размеров окна}
            {Установить новые размеры окна формы}
            FTestForm.SetBounds( pRect.Left, pRect.Top,
                        pRect.Right - pRect.Left, pRect.Bottom - pRect.Top);
            Result:= true;
        end;
            else  Result:= false;
    end;
end;
```

3.3.3 Получение, анализ и отображение данных

Методы получения измеренных данных описаны в пп. 3.2.6.

3.3.4 Компиляция и отладка

Процесс создания plug-in`а завершен. Данное описание содержит лишь основные участки кода plug-in`а. Полная распечатка программного кода приведена в приложении. Полученный код должен компилироваться.

3.4 Plug-in real-time обработки измеренных данных

Plug-in, описанный в пп 3.2, получает и отображает измеренные данные 2 раза в секунду. Очевидно, что такой plug-in может получать не весь объем измеренных данных, измеренные данные в теги могут поступать чаще, чем 2 раза в секунду (к примеру, в том случае если период обновления ПО «Recorder» меньше 0.5с). К тому же, обновление данных в тегах и события от таймера не синхронизированы. То есть, при обработке измеренных данных по таймеру может возникнуть ситуация, когда один и тот же блок будет обработан дважды.

Ниже описан способ real-time обработки полного объема измеренных данных.

3.4.1 Постановка задачи

Требуется разработать тестовый plug-in`, который будет выполнять следующие функции:

1. анализ полного потока измеренных данных для тега, вычисление среднего арифметического данных каждого блока; использовать первый тег из списка тегов;
2. сохранение вычисленных в пп.1 значений в файл.

Основная задача разработки plug-in`а – это продемонстрировать способ получения полного потока измеренных данных.

В качестве основы для разработки нового plug-in`а можно использовать код, написанный в пп. 3.1.

3.4.2 Тег, для которого производится обработка данных

Тестовый plug-in должен получать измеренные данные у одного тега, первого в списке. Это значит, что plug-in должен хранить ссылку на тег, точнее на интерфейс тега. Для получения измеренных данных plug-in использует интерфейс тега IBlockAccess, именно на этот интерфейс и будет храниться ссылка. Получить ссылку на тег, plug-in может только у объекта Recorder, поэтому необходимо хранить ссылку на объект Recorder. Класс plug-in`а должен обновлять ссылку на тег после каждого изменения настройки ПО «Recorder».

Ниже приведен участок кода описания класса TTestPlugin. Этот участок содержит описание переменной для хранения ссылки на объект Recorder и на интерфейс рабочего тега.

```
TTestPlugin = class(TInterfacedObject, IRecorderPlugin)
…
protected
        {Ссылка на интерфейс для управления Recorder`ом}
        FIRecorder: IRecorder;
        {Ссылка на интерфейс для получения данных тега}
        FDataTag:   IBlockAccess;
…
end;
```

Ниже приведен код методов TTestPlugin.EndConfigure, TTestPlugin.RefreshTagReference. Метод EndConfigure будет вызываться при завершении настройки ПО «Recorder». В методе RefreshTagReference производится получение ссылки на первый тег и получение у этого тега интерфейса IBlockAccess.

```
procedure TTestPlugin.RefreshTagReference;
var  FTag: ITag;
begin
 if FIRecorder.GetTagsCount > 0 then //получить ссылки на интерфейс тега
 begin
   //использование преобразования типа обосновано в описании
   FTag:= ITag(FIRecorder.GetTagByIndex(0));
   //Получение у тега интерфейса на блочный доступ к данным
   if (FAILED(FTag.QueryInterface( IBlockAccess, FDataTag))) then
     FDataTag := nil;
 end;
end;

//Метод подготовки формы после того, как конфигурация была изменена
procedure TTestPlugin.EndConfigure;
begin
 RefreshTagReference; //обновление ссылки на тег
end;
```

3.4.3 Подготовка к измерению. Файл с результирующими данными

В процессе измерения, когда будет необходимо прочесть данные у тега, понадобится буфер, буфер в который будет читаться блок данных. Память под этот буфер следует выделить перед измерением, а освободить можно после завершения измерения.

Класс plug-in`а должен сохранять результат своей работы в файл. Для работы с файлом удобно использовать класс файлового потока TFileStream. Перед началом измерения необходимо подготовить файл – создать объект файлового потока, сформировать имя файла и открыть файл.

Для подготовки plug-in`а к измерению создадим отдельный метод PrepareToStart, в котором будет выполняться выделение памяти и открытие файла. Перед началом измерения ПО «Recorder» рассылает plug-in`ам сообщения с кодом PN_BEFORE_RCSTART, метод PrepareToStart следует вызывать при получении этого сообщения.

```
procedure TTestPlugin.PrepareToStart;
var
    name : string;
    Size: integer;
begin
    try
        //Формирование имени файла, в который будут
        //записаны результирующие данные.
        //...получение рабочей директории...
        name := ExtractFileDir( Application.ExeName);
        if name <> '' then
                name := name + '\';
        //...добаление имени файла...
        name := name + FILENAMETEMLATE;
        //...создание файлового потока...
        FDataFile:= TFileStream.Create(name , fmCreate);
    except
        FDataFile := nil;
    end;

    if Assigned( FDataTag) then
    begin
        FBlockCount:= 0;
        //получение размера блока для буфера
        Size:= FDataTag.GetBlocksSize;
        //выделение буфера для чтения данных
        SetLength( FDataBuffer, Size);
    end;
end;
```

3.4.4 Получение и обработка сообщений обновления данных

Как описано в спецификации интерфейсов ПО «Recorder» получает измеренные данные от драйверов порциями (блоками) и сохраняет для каждого тега в отдельном буфере. Для того, чтобы plug-in`ы могли «знать» о получении новых данных, ПО «Recorder» генерирует всем plug-in`ам сообщение. Сообщение обновления данных – это сообщение с кодом PN_UPDATEDATA, для каждого plug-in`а вызывается метод IRecorderPlugin.Notify.

Для получения измеренных данных из буферов тегов, создадим у класса plug-in`а отдельный метод UpdateProcess. Этот метод будут вызываться при получении сообщения обновления данных. Ниже приведен участок кода метода TTestPlugin.Notify.

```
function TTestPlugin.Notify(const dwCommand: DWORD;
                                    const dwData: DWORD): boolean;
begin
    case dwCommand of
        PN_ENTERRCCONFIG: begin // перешел в режим настройки
            BeginConfigure;  //обработать начало изменения настройки
            Result:= true;
        end;
        PN_LEAVERCCONFIG: begin // вышел из режима настройки
            EndConfigure;  //обработать изменение настройки
            Result:= true;
        end;
        PN_UPDATEDATA: begin //команда обновления данных в тегах
            UpdateProcess; //обработать полученные (новые) данные
            Result:= true;
        end;
        PN_BEFORE_RCSTART: begin // подготовка к измерению
            PrepareToStart; //выполнить подготовку к измерению
            Result:= true;
        end;
        PN_RCSTOP: begin     //оповещение о завершении измерения
            CompleteAfterStop; //обработать - измерение завершено
            Result:= true;
        end;
        else
            Result:= false; //прочие сообщения не обрабатываем
    end;
end;
```

Обработка обновленных данных выполняется в методе UpdateProcess.

ПО «Recorder» «сообщает» plug-in`ам о получении каждого нового блока. Блоки во все теги поступают не синхронно, момент получения блоков тегом зависит от настройки тега, от его частоты дискретизации. То есть, может возникнуть ситуация, когда ПО «Recorder» «сообщает» о получении нового блока, но не для того, тега, с которым работает plug-in. Таким образом, plug-in должен «уметь» определять изменились ли данные конкретного тега, обновились ли они.

Для каждого тега ПО «Recorder» хранит полное количество полученных им блоков, от начала измерения. Классу plug-in`а необходимо хранить полное количество обработанных plug-in`ом блоков, от начала измерения. Вычисляя разницу между количеством полученных ПО «Recorder» и количеством обработанных plug-in`ом можно определить наличие блоков данных, которые plug-in должен еще обработать. Очевидно, что эти блоки будут последними в буфере тега.

Пусть plug-in хранит кол-во обработанных им блоков в поле FBlockCount. Метод получения блоков данных UpdateProcess выглядит следующим образом:

```
procedure TTestPlugin.UpdateProcess;
var
    ReadyCount: integer; //счетчик обработанных Recorder`ом блоков
    Count: integer;      //счетчик блоков в буфере тега
    Size: integer;       //размер блока
    i: integer;
begin
    if Assigned(FDataTag) then
    begin
        FDataTag.LockVector; //Блокирование вектора с данными
        try
            //получение обработанных Recorder`ом блоков
            ReadyCount:= FDataTag.GetReadyBlocksCount;
            //получения кол-ва блоков в буфере тега
            Count:= FDataTag.GetBlocksCount;
            //получение размера блока
            Size:= FDataTag.GetBlocksSize;
            //Определение наличия новых данных в буфере тега
            if (ReadyCount <> FBlockCount) and (Count > 0 )then
            begin
                //отладочная проверка размера блока
                if Size <> Length( FDataBuffer) then
                        SetLength( FDataBuffer, Size);
                //в цикле чтение новых блоков и обработка их
                for i := 0 to ReadyCount - FBlockCount - 1 do
                begin
                    //чтение
                    if SUCCEEDED(FDataTag.GetVectorR8(
                                pointer(FDataBuffer)^,
                                Count - 1 - i, Size, true)) then
                        //обработка
                        DataProcess( pointer(FDataBuffer)^, Size);
                end;
            end;
            //запомнить кол-во обработанных plug-in`ом блоков
            FBlockCount:= ReadyCount;
        finally
            //разблокирование буфера
            FDataTag.UnlockVector
        end;
    end;
end;
```

В этом методе выполняется получение блоков данных у тега, а обработка производится при помощи функции DataProcess.

3.4.5 Обработка и сохранение данных

В методе DataProcess выполняется вычисление среднего и запись результата в файл.

```
procedure TTestPlugin.DataProcess( var Data; const Size: integer );
type
    DoubleArray = array[word] of double;
    PDoubleArray = ^DoubleArray;
var
    i: integer;
    PtrDoubleData: DoubleArray absolute Data;
    Summ: double;
begin
    //вычисление среднего
    Summ:= 0;
    for i:= 0 to Size - 1 do
        Summ:= PtrDoubleData[i];
    if Size <> 0 then
        Summ := Summ / Size;
    //сохранение в файл
    try
        FDataFile.Write( Summ, sizeof(Summ));
    except
    end;
end;
```

3.4.6 Многопоточное приложение. Потоки обработки данных

ПО «Recorder» является многопоточным приложением. С plug-in`ами одновременно могут работать два потока:

1. поток чтения данных из устройств, назовем его «Рабочий поток»;
2. поток управления plug-in`ами, назовем его «Оконный поток».

Рассмотрим ситуацию, когда plug-in не создает собственных потоков.

Какая часть кода выполняется, в каком потоке, описано ниже:

1. вызовы всех методов plug-in`а ПО «Recorder» осуществляет в Оконном потоке, (за одним исключением);
2. сообщение обновления данных все plug-in`ы получают в Рабочем потоке, это происходит только в процессе измерения (режим измерения);
3. если plug-in создает форму (любое окно), то все события этого окна обрабатываются в Оконном потоке.

Таким образом, необходимо иметь в виду, что доступ к plug-in`у в процессе измерения может осуществляться одновременно двумя потоками, особенно это актуально, если plug-in создает собственную форму для отображения измеренных данных и обрабатывает сообщение обновления данных. Для защиты класса plug-in`а и его ресурсов от одновременного использованияч несколькими потоками, следует использовать системные объекты синхронизации потоков.

Plug-in, разработанный в этом разделе, не содержит данных, которые могли бы использоваться одновременно обоими потоками, Рабочим потоком и Оконным потоком. По этой причине в коде plug-in`а не используются системные объекты синхронизации потоков.

3.4.7 Компиляция и отладка

Процесс создания plug-in`а завершен. Полученный код должен компилироваться. Полная распечатка программного кода приведена в приложении.

3.5 Plug-in, меняющий настройку ПО «Recorder»

3.5.1 Постановка задачи

В качестве примера необходимо разработать тестовый plug-in, который выполняет следующие функции:

1. позволяет просматривать значения параметров ПО «Recorder», периода обновления, времени хранения, состояния и т.д.;
2. позволяет получать список групп тегов, просматривать настройки групп, создавать и удалять группы тегов;
3. позволяет получать список тегов, просматривать параметры тегов, позволяет создавать и удалять виртуальные теги и теги, связанные с физическими каналами;
4. позволяет получать список доступных физических каналов, использовать данный список для создания тегов;
5. позволяет получать и менять параметры градуировочных/калибровочных функций тегов;
6. позволяет производить экспорт и импорт файлов настройки.

В качестве основы для разработки нового plug-in`а можно использовать код, написанный в пп. 3.1.

Дальнейшая доработка библиотеки и самого класса plug-in`а должна производится в соответствии с описанными выше требованиями. Основными целями, при разработке plug-in`а являются: реализация необходимой функциональности, простота кода, которая позволит показать принципы использования интерфейсов программирования ПО «Recorder».

3.5.2 Управление ПО «Recorder» из plug-in`а.

Объект plug-in`а при инициализации получает ссылку на интерфейс IRecorder объекта Recorder – объекта ядра ПО «Recorder». При помощи интерфейса IRecorder plug-in может получить доступ к списку тегов, как это описано в пп.3.2.4, либо получить доступ к списку групп тегов, либо управлять ПО «Recorder».

Объект plug-in`а имеет возможность выполнять следующее:

1. менять настройку ПО «Recorder»: добавлять/удалять группы тегов, добавлять/удалять теги, менять параметры работы программы, такие как период обновления и т.п.;
2. менять режим работы ПО «Recorder»: запускать измерение, запускать измерение с записью измеренных данных, останавливать измерение.

Очевидно, что объект plug-in`а должен сохранить ссылку на интерфейс IRecorder. Ниже приведен код метода инициализации тестового plug-in`а, в котором производится сохранение ссылки.

```
//Инициализация объекта
function TTestPlugin._Create(pOwner: IRecorder): boolean;
begin
    FIRecorder:= pOwner; // сохранение ссылки на интерфейс Recorder`а
    …
    Result:= true;
end;
```

3.5.3 Создание формы для управления ПО «Recorder»

Для предоставления оператору возможности просмотра и изменения настройки ПО «Recorder», необходимо создать в проекте новую форму. Пусть новая форма называется TTestForm, модуль, в котором описана и реализована форма, пусть называется «TestFormUnit.pas».

Очевидно, что форма TTestForm должна один раз создаваться plug-in`ом, когда создается сам plug-in и удаляться соответственно, когда удаляется plug-in. Plug-in должен хранить ссылку на объект формы. Описание класса plug-in`а дополняется следующими строками:

```
TTestPlugin = class(TInterfacedObject, IRecorderPlugin)
…
protected
        FTestForm: TTestForm;
public
        constructor Create;
        destructor Destroy; override;
…
end;
```

В данном участке кода опущены приведенные ранее (см. пп. 3.1.2.) методы и показаны только те методы и поле, которые были добавлены.

В конструкторе plug-in`а производится создание формы, а в деструкторе - удаление.

```
constructor TTestPlugin.Create;
begin
    FTestForm:= TTestForm.Create( Self);
end;
destructor TTestPlugin.Destroy;
begin
    FreeAndNil( FTestForm);
end;
```

Инициатором изменения настройки является оператор, через элементы управления формы TTestForm, он производит необходимые действия настройки. Для простоты кода plug-in`а, работа с объектом Recorder будет производиться непосредственно из формы, для этого форме необходима ссылка на интерфейс IRecorder. В форме будет храниться ссылка на объект plug-in`а, с его помощью, форма будет получить ссылку на интерфейс IRecorder.

Для избежания циклического использования модулей ссылка на объект plug-in`а описывается как ссылка на класс TObject, а в коде методов формы производится приведение её (то есть ссылки) к типу TTestPlugin.

```
TTestForm = class(TForm)
    …
    …
public
    //Конструктор формы переопределен для того, чтобы форму
    //можно было инициализировать ссылкой на объект plug-in`a
    constructor Create( const Plugin: TObject); reintroduce;
private
    {ссылка на объект plug-in`a}
    {ссылка типа TObject, для того, чтобы избежать}
    {рекурсивных использований модулей}
    FPlugin: TObject;
    …
    …
end;
```

Ссылку на объект plug-in`a форма будет получать в конструкторе, для этого конструктор формы переопределяется.

```
//Конструктор формы переопределен для того, чтобы форму
//можно было инициализировать ссылкой на объект plug-in`a
constructor TTestForm.Create( const Plugin: TObject);
begin
    inherited Create( nil);
    FPlugin:= Plugin; //сохранить ссылку на объект plug-in`a
end;
```

Программу просмотра и изменения настройки ПО «Recorder» можно разбить на несколько отдельных секций:

1. секция параметров ПО «Recorder», значение периода обновления, флаги состояния и пр.;
2. секция групп, просмотр, создание, удаление;
3. секция тегов, просмотр, создание, удаление;
4. секция физических каналов, просмотр;
5. секция ГФ/КФ, просмотр, создание ГФ/КФ с тестовыми значениями.

Расположим на форме объект PageControl, типа TPageControl, эта компонента позволяет создать многостраничное диалоговое окно. Создадим для каждой секции отдельную страницу: TSRecorder, TSGroups, TSTags, TSPChans, TSClbs. Доработка каждой страницы будет описана ниже. Расположим на форме две кнопки: SBtnImportConfig и SBtnExportConfig для реализации функций импорта и экспорта файлов настройки соответственно.

3.5.4 Изменение настройки ПО «Recorder» из plug-in`a

В ПО «Recorder» может быть запущено несколько plug-in`ов, в том числе, к примеру, и plug-in`ы, которые позволяют менять настройку удаленно, с других компьютеров.

Очевидно, что если несколько plug-in`ов одновременно попытаются изменить настройку, это может привести к ошибкам. Для того, чтобы упорядочить процесс изменения настройки, чтобы не дать возможности менять настройку двум plug-in`ам одновременно в ПО «Recorder» предусмотрен специальный режим – режим настройки.

Plug-in должен инициировать режим настройки, произвести необходимые изменения и завершить режим настройки. Если один plug-in перевел ПО «Recorder» в режим настройки, то другому, при попытке перевести в режим настройки, будет отказано.

Для перевода ПО «Recorder» в режим настройки в интерфейсе IRecorder определен метод EnterConfigMode, параметром метода передается ссылка на объект plug-in`a. Эта ссылка используется для того, чтобы объект Recorder «знал», какой plug-in в данный момент производит изменение настройки. Для выхода из режима настройки в интерфейсе IRecorder определен метод LeaveConfigMode.

Так как все изменения настройки, тестовым plug-in`ом, производятся через форму, то следует в форме определить два сервисных метода BeginConfigure и EndConfigure, которые выполняют вход и выход из режима настройки соответственно. Метод BeginConfigure возвращает признак успешности перехода в режим настройки. Если BeginConfigure вернул false, значит настройка ПО «Recorder» производится другим plug-in`ом, либо из управляющих диалогов самого ПО «Recorder». В этом случае следует отказать оператору в изменении настройки.

```
{Сервисный метод, который должен быть вызван для}
{перехода в режим настройки}
function TTestForm.BeginConfigure: boolean;
begin
    //Нельзя менять настройку если режим настройки не включен
    Result:=
        SUCCEEDED(TTestPlugin(FPlugin).FIRecorder.
                    EnterConfigMode( TTestPlugin(FPlugin), 0));
end;
{Сервисный метод, который должен быть вызван для}
{выхода из режима настройки}
procedure TTestForm.EndConfigure;
begin
    //Изменения, внесенные в настройку ПО «Recorder», вступят
    //в силу только после выхода из режима настройки
    TTestPlugin(FPlugin).FIRecorder.
                    LeaveConfigMode( TTestPlugin(FPlugin), 0)
end;
```

3.5.5 Период обновления, флаги состояния

Пусть первая страница формы TSRecorder, будет отображать: список установленных флагов состояний и таблицу со значениями параметров ПО «Recorder».

Для отображения списка флагов состояния на форме следует расположить объект ListBoxRecStates, типа TListBox. Для каждого флага следует сформировать строку, описывающую флаг, список ListBoxRecStates заполнить строками описания флагов. Для получения списка флагов на форме следует расположить кнопку BtnMrGetState. В методе обработчике нажатия этой кнопки будет производиться получение слова состояния, расшиф-

ровка его и заполнение списка ListBoxRecStates. Пример реализации подобного метода приведен ниже.

```
procedure TTestForm.BtnMrGetStateClick(Sender: TObject);

type //Структура для описания одной строки таблицы соответствия...
    TStateString = record
        Mask:       longword;    //маска на слово состояния
        State:      longword;    //состояние и ...
        Text:       string;      //соответствующая ему строка
    end;

const
    //Объявления констант упрщающих представление кода программы
    RS_ALLRESET = RS_NEEDLINKSREFRESH +
                              RS_NEEDSOFTWARERESET;
    RS_NHWR    = RS_NEEDHARDWARERESET;

    StateStringLen = 14; //размер таблицы соответствия
type
    TStateStringTable = array[0..StateStringLen-1] of TStateString;

const
    //Массив соответствия...
    stlinks: TStateStringTable = (
        {-Маска--------------------Состояние---------Описание------------}
        ( Mask: RS_STOP;  State: RS_STOP;     Text: 'Остановлен'),
        ( Mask: RS_VIEW;  State: RS_VIEW;     Text: 'Просмотр'),
        …
        …
    );
var
    State: integer;
    i: integer;
begin
    //Очистка списка отображающего перечень флагов состояния
    ListBoxRecStates.Clear;
    //получение кода состояния
    State:= TTestPlugin( FPlugin).FIRecorder.GetState(RS_FULLMASK);
    //в цикле проверка филагов и просмотр таблицы соответствия,
    //добавление строк описания флагов состояний в общий список
    for i := 0 to StateStringLen - 1 do
    begin
        if (State and stlinks[i].Mask) = stlinks[i].State then
                ListBoxRecStates.Items.Add( stlinks[i].Text);
    end;
end;
```

Для того, чтобы по флагу получить строку описания, в методе объявлена таблица соответствия флага – строке. Таблица соответствия – это массив, один элемент массива предназначен для описания одного флага. Для каждого флага указывается маска, которую необходимо наложить на слово состояния, чтобы получить значение флага, собственно само значение флага и соответствующая строка. Элемент таблицы соответствия описыва-

ется типом TStateString. Массив описывается типом TStateStringTable, таблица – сам константный массив – это переменная stlinks. В приведенном выше участке кода, содержится не весь константный массив соответствия, а только его часть.

Для получения состояния ПО «Recorder» вызывается метод IRecorder.GetState, параметром метода можно указать маску, для флагов состояния, в данном примере указана маска всех флагов. Слово состояния запрашивается один раз, потом производится его расшифровка.

Для отображения перечня параметров ПО «Recorder» следует на странице TSRecorder расположить объект SGMrInfo, типа TStringGrid. Перечень параметров:

1. код состояния, в цифровом виде;
2. базовый каталог, наименование каталога, в котором находится ПО «Recorder»;
3. каталог с данными, имя каталога в который сохраняются файлы с измеренными данными;
4. файл настройки №1, имя файла настройки;
5. каталог с файлами ГФ/КФ, имя каталога, в котором расположены файлы с параметрами ГФ/КФ – база ГФ/КФ;
6. файл настройки №2, имя файла настройки;
7. период обновления;
8. время отображения, временной промежуток для которого ПО «Recorder» хранит в памяти и отображает измеренные данные.

Для предоставления оператору возможности просмотра значений параметров на странице следует расположить кнопку BtnMrGetInfo. В методе обработчике нажатия кнопки будет выполняться последовательное получение значений параметров и отображение их. Для получения значений параметров в методе определен ряд временных переменных.

```
//Сбор информации о Recorder`е и отображение её в табличном виде
procedure TTestForm.BtnMrGetInfoClick(Sender: TObject);
var
    pmr: IRecorder; //ссылка на объект Recorder
    v: OleVariant;    //переменная для получения значений свойств
    len: integer;     //переменная для получения размеров строк
    tmp: string;      //переменная для получения строк
begin
    …
    //получение ссылки на объект Recorder
    pmr:= TTestPlugin( FPlugin).FIRecorder;
    …
    …
end;
```

В частности переменная pmr служит для хранения ссылки на объект Recorder.
Получение кода состояния осуществляется следующим образом:

```
tmp := '0x' + IntToHex( pmr.GetState(RS_FULLMASK), 8);
```

Получение базового каталога, каталога с данными, и имени файла настройки осуществляется соответственно при помощи методов GetRcBasePath, GetSignalFolderName, GetProjectName.

Для получения имени второго файла конфигурации, следует получить значение переменной окружения: Имя переменной окружения «\Recorder\System\CfgName». Для того, чтобы получить значение переменной окружения, следует вызвать метод GetEnvironmentString. В приведенном ниже участке кода этот метод вызывается дважды. Первый раз для того, чтобы получить размер строки имени файла, далее для временной переменной tmp устанавливается требуемый размер, метод вызывается второй раз и в подготовленную строку tmp копирует имя файла.

```
if pmr.GetEnvironmentString( '\Recorder\System\CfgName', nil, len) then
begin
    //выделение буфера строки под чтение имени файла настройки
    SetLEngth(tmp, len);
    //чтение строки имени
    pmr.GetEnvironmentString( '\Recorder\System\CfgName',
                                    PChar(tmp), len);
end
else
    tmp:= '';
```

Имя каталога ГФ/КФ можно получить, запросив, значение свойства с идентификатором RCPROP_CALIBRDBNAME.

```
if pmr.GetProperty( RCPROP_CALIBRDBNAME, v) = S_OK then
    tmp:= v
else
    tmp := '';
```

Значение периода обновления и времени отображения можно получить как значения свойств с идентификаторами RCPROP_REFRESHPERIOD, RCPROP_VIEWTIME, аналогично описанному выше.

Для того, чтобы изменить, к примеру, период обновления следует установить новое значение свойства с идентификатором RCPROP_REFRESHPERIOD. Новое значение свойства устанавливается при помощи метода SetProperty. Для изменения свойств, следует предварительно перейти в режим настройки.

3.5.6 Параметры физических каналов

В зависимости от конфигурации измерительной аппаратуры ПО «Recorder» может содержать различный список физических каналов. В зависимости от настройки ПО «Recorder» для некоторых каналов могут быть созданы теги, а для некоторых нет. Список доступных физических каналов, это перечень тех каналов, для которых еще не были созданы теги. Для физического канала можно получить несколько характеристик – это имя

аппаратного измерительного устройства, строка физического адреса и уникальный числовой идентификатор. Как только для физического канала создается тег, для этого канала становится возможным получение измеренных данных.

Для отображения перечня и характеристик физических каналов используется страница формы TSPChans. На странице следует расположить объект SGPhChans типа TStringGrid и кнопку BtnGetPhChans. Объект SGPhChans используется для табличного отображения списка физических каналов. По нажатию кнопки BtnGetPhChans следует производить получение списка физических каналов и формирование таблицы.

Запрос списка доступных физических каналов и отображение можно разделить на два отдельных метода для упрощения кода, соответственно метод получения каналов и метод отображения. Пусть специализированный метод GetPhChansDsc реализует получение списка физических каналов, формирует три динамических массива: массив числовых идентификаторов каналов, массив строк адресов каналов и массив строк имен устройств.

В приведенном ниже участке кода метода GetPhChansDsc, описывается ряд временных переменных и производится получение количества доступных физических адресов. Далее устанавливаются размеры результирующих динамических массивов.

```
function TTestForm.GetPhChansDsc( var ChanIDs: DynCardinals;
                                  var Addresses: DynStrings;
                                  var Devices: DynStrings): boolean;
var
    buffer: array [byte] of char; {Буфер для получения строк}
    buflen: integer;   {Переменная для хранения размера буфера строк}
    prec: IRecorder;
    Count: integer;
    i: integer;
begin
    try
        prec:= TTestPlugin(FPlugin).FIRecorder; {ссылка на Recorder}
        Count:= prec.dvchGetAvailableChansCount; //кол-во каналов

        //Установить размеры результирующих динамических массивов
        SetLength(ChanIDs, Count);    //идентификаторы
        SetLength(Addresses, Count);  //строки адресов
        SetLength(Devices, Count);    //строки имен устройств
```

В следующем участке кода метода GetPhChansDsc выполняется получение списка числовых идентификаторов физических каналов, и в цикле, по значениям идентификаторов, производится получение характеристик физических каналов.

```
        if Count > 0 then //если кол-во каналов больше 0, то...
        begin
          //Заполнение массива идентификаторов каналов
          if FAILED(prec.dvchGetChansList( ChanIDs[0])) then
                                        raise Exception.Create('');
          //цикл по всем идентификаторам каналов
          for i:= 0 to Count - 1 do
          begin
              buflen:= sizeof(buffer) - 1;
              //получение строки адреса физического канала
              if FAILED( prec.dvchGetAddress(ChanIDs[i],
                                              @buffer, buflen)) then
                                        raise Exception.Create('');
              //установка строки в динамический массив
              Addresses[i]:= PChar(@buffer);
              buflen:= sizeof(buffer) - 1;
              //получение строки имени устройства
              if FAILED( prec.dvchGetDeviceName(ChanIDs[i],
                                        @buffer, buflen)) then
                                        raise Exception.Create('');
              //установка строки в динамический массив
              Devices[i]:= PChar(@buffer);
          end;
      end;
```

Метод IRecorder.dvchGetChansList выполняет заполнение динамического массива значениями числовых идентификаторов каналов. Методы IRecorder.dvchGetAddress и IRecorder.dvchGetDeviceName служат для получения строки адреса канала и строки имени устройства по идентификатору соответственно.

Ниже приведен последний участок кода метода GetPhChansDsc, в котором, в случае ошибок, очищаются результирующие динамические массивы.

```
        //Если ошибка произошла с функциями получения
        //параметров физических каналов.
      except
          SetLength( ChanIDs, 0);
          SetLength( Addresses, 0);
          SetLength( Devices, 0);
          Result:= false;
      end;
  end;
```

Метод формы BtnGetPhChansClick, является обработчиком нажатия кнопки BtnGetPhChans, этот метод должен отображать список каналов в таблице SGPhChans, получение списка можно выполнить при помощи метода GetPhChansDsc.

```
procedure TTestForm.BtnGetPhChansClick(Sender: TObject);
var
    ChanIDs:        DynCardinals;{Массив идентификаторов каналов}
    Addresses:      DynStrings; {Массив строк адресов каналов}
    Devices:        DynStrings;   {Массив строк имен устройств каналов}
    Count:    integer;
    i:        integer;
begin
    //получение списка идентификаторов и т.и. доступных каналов
    if GetPhChansDsc( ChanIDs, Addresses, Devices) then
    begin
        //получить кол-во доступных физических каналов
        Count := Length( ChanIDs);
        //установить размер (кол-во) строк таблицы
        SGPhChans.RowCount := Count + 2;
        //Вывести в таблицу параметры доступных физических каналов
        for i:= 0 to Count - 1 do
        begin
            SGPhChans.Cells[0, i + 1] := Devices[i];   {имя устройства}
            SGPhChans.Cells[1, i + 1] := Addresses[i]; {строка адреса}
        end;
    end
    …
end;
```

В объекте SGPhChans для каждого канала создается отдельная строка, в строе отображаются: имя устройства и адрес канала.

3.5.7 Управление тегами

Для отображения списка тегов на форме расположена страница TSTags. Чтобы показать список тегов и различные характеристики тегов в табличном виде, следует создать на странице объект SGTags, типа TStringGrid. Для обновления содержимого таблицы, для создания тега, для создания виртуального тега, для удаления тега, для переименования тега расположим на странице кнопки BtnGetTags, BtnTagAdd, BtnTagAddVirt, BtnTagDel и BtnTagRename соответственно.

Метод формы BtnGetTagsClick будет служить для обработки нажатий на кнопку обновления таблицы тегов. Формирование таблицы тегов можно разбить на два метода: метод получения списка тегов и метод отображения тегов (всех параметров тегов) в таблице. Пример кода метода BtnGetTagsClick приведен ниже.

```
procedure TTestForm.BtnGetTagsClick(Sender: TObject);
var
    prec: IRecorder;
    ptag: ITag;
    i: integer;
    Count: integer;
begin
    SGTags.RowCount := 2; //предварительная очистка таблицы тегов
    SGTags.Rows[1].Clear;
    prec:= TTestPlugin(FPlugin).FIRecorder; //ссылка на Recorder
    Count:= prec.GetTagsCount; //получение общего кол-ва тегов
    for i:= 0 to Count - 1 do
    begin //в цикле получение ссылки на тег по индексу тега
        //получение ссылки на тег
        pointer(ptag):= prec.GetTagByIndex (i);
        //добавление строки с параметрами тега в таблицу
        AddTagToList(ptag);
        ptag := nil; //освобождение ссылки на тег
    end;
    //проверить корректность выбранной строки ( тега) в таблице
    CheckTagSelection( SGTags.Row);
end;
```

В этом методе используются функции интерфейса IRecorder.GetTagsCount и IRecorder.GetTagByIndex, для получения количества тегов и для получения ссылки на тег по номеру соответственно. Преобразование типа при получении ссылки на тег обусловлено соображениями, описанными в пп. 2.1.1.

Для формирования таблицы, отображения характеристик тегов используется метода формы AddTagToList. В этом методе производится добавление строки в таблицу тегов, запрос характеристик тега и отображение. В таблице тегов выводятся следующие характеристики тегов:

1. имя тега;
2. строка описания тега;
3. имя группы, которой принадлежит тег;
4. строка размерности данных тега (размерность измеряемой физической величины);
5. строка физического адреса, это строка адреса физического канала;
6. код типа связи с физическим каналом, данный код говорит о том, является ли тег виртуальным или он в данный момент связан с физическим каналом;
7. права доступа – чтение, запись;
8. частота дискретизации;
9. код типа данных, возможные значения соответствует значениям кодов типа Variant;
10. минимум;
11. максимум.

Характеристики №№ 2, 4, 5, 7, 9, 10, 11 можно получить через свойства тега, идентификаторы свойств TAGPROP_DESCRIBE, TAGPROP_UNITS, TAGPROP_HARDWAREADRESS, TAGPROP_TYPE, TAGPROP_DATATYPE, TAGPROP_MINVALUE, TAGPROP_MAXVALUE соответственно.

Пример кода получения свойства тега (строка размерности измеряемой величины):

```
//получить строку размерности
if not ptag.GetProperty( TAGPROP_UNITS, v) then
        raise Exception.Create('');
//установка полученного значения в таблицу
SGTags.Cells[ 3, Index] := v;
```

Характеристики №№ 1, 6, 8 можно получить при помощи функций интерфейса ITag: GetName, GetLinkState, GetFreq соответственно.

Для получения имени группы, которой принадлежит тег, выполняются следующие действия:

1. при помощи функции ITag.GetGroup, запрашивается ссылка на объект группы, которой принадлежит тег;
2. при помощи функции ITagGroup.GetName запрашивается имя группы.

Значения характеристик №№ 1, 2, 4, 8, 10, 11 можно изменить. Так же можно изменить группу, которой принадлежит тег, подробнее это описано в пп. 3.5.9. Для переименования тега и для изменения частоты дискретизации используются методы ITag.SetName и ITag.SetFreq соответственно. По вызову функции ITag.SetFreq для тега устанавливается такое значение частоты дискретизации, которое допустимо для физического канала и которое наиболее близко к требуемому. Значения характеристик №№ 2, 4, 10, 11 можно установить через соответствующие свойства. Изменять значения характеристик тега можно только в режиме настройки.

Для создания нового тега используется метод IRecorder.CreateTag. Параметрами этому методу необходимо указать новое имя тега, тип тега (в данной ситуации это LS_HARDWARE) и значение числового идентификатора физического канала, с которым будет связан тег. Методы получения идентификаторов физических каналов описаны в пп. 3.5.6. Метод IRecorder.CreateTag, в случае успешного выполнения возвращает ссылку на новый созданный тег. Если в процессе создания тега произошла ошибка, то возвращается nil. Возможными ошибками при создании тега являются: не уникальное имя тега и ошибочный числовой идентификатор физического канала.

Создание нового тега в тестовом plug-in`е выполняется в методе формы TTestForm.BtnTagAddClick, этот метод является обработчиком нажатия кнопки BtnTagAdd. В этом методе выполняется:

1. получение массива числовых идентификаторов и строковых адресов доступных физических каналов, при помощи метода GetPhChansDsc;
2. выполняется создание и отображение специальной диалоговой формы FormPhChans, эта форма позволяет оператору выбрать требуемый физический канал;
3. производится переход к режиму настройки;
4. выполняется создание тега – вызов метода IRecorder.CreateTag;
5. добавление тега в таблицу тегов;
6. выполняется выход из режима настройки.

Пример кода приведен ниже, в данном участке кода опущен ряд несущественных деталей.

```
procedure TTestForm.BtnTagAddClick(Sender: TObject);
…
begin
    …
    {Получение ссылки на объект Recorder}
    prec:= TTestPlugin(FPlugin).FIRecorder;
    //получение списка идентификаторов и строк
    //адресов доступных физических каналов
    if GetPhChansDsc( ChanIDs, Addresses, Devices) and
                    (Length( ChanIDs) > 0) then
    begin

        //Активизация формы со списокм физических каналов
        if FormPhChans.Execute( PhChans, i) and (i >= 0) and
                            (i < PhChans.Count) then
        begin
            //переход к режиму настройки
            if BeginConfigure then
            begin
                try
                  pointer(ptag) := prec.CreateTag( PChar(PhChans.Strings[i]),
                                                LS_HARDWARE, ChanIds[i]);
                  //если тег создан успешно и получена ссылка на него, то...
                  if Assigned(ptag) then
                        //добавить тег в таблицу
                        AddTagToList( ptag);
                        …
                finally//необходимо выйти из режима настройки
                  EndConfigure;
                end;
            end;
        end
    end;
end
```

После успешного создания тега, его можно добавить к таблице тегов, для этого используется метод формы AddTagToList.

Обработчик нажатия кнопки BtnTagDel – метод BtnTagDelClick выполняет удаление тега, выбранного в таблице тегов. Удаление должно производится следующим образом: выполняется переход к режиму настройки, поиск тега по выбранному в таблице имени, удаление тега, выход из режима настройки. Участок кода метода BtnTagDelClick приведен ниже.

```
procedure TTestForm.BtnTagDelClick(Sender: TObject);
var
    prec: IRecorder;
    ptag: ITag;
begin
    //Проерка корректности выбора строки таблицы тегов...
    if (SGTags.Row > 0) and (SGTags.Row < (SGTags.RowCount - 1)) then
    …
    //переход к режиму настройки, и проверка перехода
    if BeginConfigure then
    begin
        try
            //получение ссылки на Recorder
            prec:= TTestPlugin(FPlugin).FIRecorder;
            //получение ссылки на тег по имени
            pointer(ptag):= prec.GetTagByName( PChar(
                                    SGTags.Cells[0, SGTags.Row]));

            //Если получена корректная ссылка на тег, то...
            if Assigned( ptag) then
            if prec.CloseTag(ptag) then //по ссылке тег удаляется
            begin
                ptag:= nil;
                //удаление строки из таблицы
                DelRowWithUpShift( SGTags, SGTags.Row);
            end;
        finally//в завершении - выйти из режима настройки
            EndConfigure;
        end;
    end;
end;
```

Обработчик нажатия кнопки BtnTagRename – метод BtnTagRenameClick выполняет переименование тега, выбранного в таблице тегов. Переименование производится следующим образом: отображается диалоговая форма для ввода нового имени тега, выполняется переход к режиму настройки, поиск тега по выбранному в таблице имени, установка нового имени тегу, выход из режима настройки, коррекция таблицы тегов.

3.5.8 Управление виртуальными тегами

Процесс создания виртуального тега должен выполняется при помощи метода IRecorder.CreateTag, параметры следующие: имя нового тега, типа тега (в данной ситуации LS_VIRTUAL), значение последнего параметра не существенно, это может быть любая переменная.

Алгоритм создания нового виртуального тега (для, разрабатываемого в данном разделе, plug-in`a) должен быть следующим:

1. создание и вызов формы для предоставления оператору возможности ввода именно нового тега;
2. переход к режиму настройки;
3. создание тега, с требуемым именем;

4. добавление тега в таблицу тегов;
5. выход из режима настройки;

Создание нового виртуального тега выполняется по нажатию кнопки BtnTagAddVirt. Метод формы TTestForm.BtnTagAddVirtClick - обработчик нажатия кнопки BtnTagAddVirt, выполняет изложенный выше алгоритм создания виртуального тега.

Для добавления информации о теге в таблицу используется метод формы AddTagToList.

Созданный таким образом виртуальный тег, можно использовать для формирования plug-in`ом собственных тестовых или расчетных данных. Данные виртуального тега, как и любого другого, могут отображаться и сохраняться в файлы.

3.5.9 Управление группами тегов

Для отображения списка групп тегов на форме расположена страница TSGroups. Для табличного представления списка групп на странице необходимо расположить объект SGGroups типа TStringGrid. Так же на странице следует расположить кнопки BtnGetGroups, BtnGrpAdd, BtnGrpDel, BtnGrpRename для обновления содержимого таблицы, создания новой группы, удаления группы и переименования группы соответственно.

Обновление таблицы тегов выполняется по нажатию кнопки BtnGetGroups в методе формы BtnGetGroupsClick. Для группы тегов можно получить и отобразить значения следующих параметров:

1. строка имени группы;
2. строка описания группы;
3. признак активности группы, признак того, что измеренные данные тегов группы сохраняются в файлы;
4. признак сохранения измеренных данных в отдельный каталог;
5. строка имени каталога, в который сохраняются измеренные данные.

В объекте SGGroups для каждой группы создается отдельная строка, в строке находятся значения описанных выше параметров.

Получение списка групп осуществляется при помощи методов интерфейса IRecorder. Метод IRecorder.GetGroupsCount позволяет получить общее количество групп, метод IRecorder.GetGroupByIndex позволяет получить ссылку на объект группы по номеру группы. Ниже приведен участок кода метода BtnGetGroupsClick.

```
procedure TTestForm.BtnGetGroupsClick(Sender: TObject);
    …
begin
    …
    //Получение ссылки на объект Recorder`a (у plug-in`a)
    pmr := TTestPlugin(FPlugin).FIRecorder;
    …
    count:= pmr.GetGroupsCount();//получить кол-во групп
    for i:= 0 to count - 1 do //в цикле по всем группам...
    begin
    //получить интерфейс группы, по номеру
        pointer(pgrp) := pmr.GetGroupByIndex( i);
        AddGroupToList( pgrp); //добавить в таблицу и отобразить...
        pgrp:= nil;
    end;
end;
```

В методе BtnGetGroupsClick производится получение ссылок на все группы и при помощи метода формы AddGroupToList выполняется добавление информации о группе в таблицу групп. В метод AddGroupToList передается ссылка на группу. Ниже приведен участок кода метода AddGroupToList, в котором выполняется получение характеристик группы и отображение в таблице на форме.

```
function TTestForm.AddGroupToList( pgrp: ITagsGroup): boolean;
var
    buffer: array [byte] of char;
    Index: integer;
    v: OleVariant;
begin
    …
    //получение имени группы
    if FAILED(pgrp.GetName( @buffer)) then raise Exception.Create('');
    SGGroups.Cells[0, Index]:= PChar(@buffer);
    //получение строки описания группы
    if FAILED(pgrp.GetProperty( TGRPROP_DESCRIBE, v)) then
            raise Exception.Create('');
    SGGroups.Cells[1, Index]:= v;
    //получение признака активности группы
    if pgrp.GetRecEnabled() then
            SGGroups.Cells[2, Index]:= 'Активна'
    else
            SGGroups.Cells[2, Index]:= 'Не активна';

    //получение признака необходимости сохранения измеренных
    //данных всех тегов группы в отдельный (для группы) каталог
    if FAILED(pgrp.GetProperty( TGRPROP_WRITETOPERSONALFRAME, v))
                then raise Exception.Create('');
    if v then
            SGGroups.Cells[3, Index]:= 'Да'
    else
            SGGroups.Cells[3, Index]:= 'Нет';

    //получение строки имени каталога, для измеренных данных
    if FAILED(pgrp.GetProperty( TGRPROP_PERSONALFRAMENAME, v))
                then raise Exception.Create('');
    SGGroups.Cells[4, Index]:= v;

      …
end;
```

Получение имени группы осуществляется в символьный буфер buffer, размер которого должен быть не менее 256 символов. Строка описания, имя каталога для измеренных данных, признак использования отдельного каталога для измеренных данных можно получить как свойства. Признак активности группы можно получить при помощи метода ITagsGroup.GetRecEnabled.

3.5.10 Чтение и сохранение конфигурации ПО «Recorder»

ПО «Recorder» хранит собственную настройку в группе файлов. Интерфейс программирования ПО «Recorder» позволяет производить импорт и экспорт файлов. В интерфейсе IRecorder описаны два метода: ImportSettings и ExportSettings для импорта и экспорта соответственно. Параметрами метода служат:

1. имя каталога, в который экспортируются или из которого импортируются файлы конфигурации, строка имени каталога должна заканчиваться слешем;
2. флаги импорта-экспорта.

Процесс импорта и экспорта инициируются оператором, поэтому на форме следует расположить две кнопки SBtnImportConfig и SBtnExportConfig. По нажатию кнопки SBtnImportConfig вызывается метод формы SBtnImportConfigClick, в котором выполняется импорт. По нажатию кнопки SBtnExportConfig вызывается метод формы SBtnExportConfigClick, в котором выполняется экспорт. Код методов приведен ниже.

```
procedure TTestForm.SBtnImportConfigClick(Sender: TObject);
    var pmr: IRecorder;
begin
    pmr:= TTestPlugin( FPlugin).FIRecorder; //Сылка на объект Recorder`a
    //выполнение и проверка рез-та импорта
    if SUCCEEDED( pmr.ImportSettings( 'c:\temp\', 0)) then
        Application.MessageBox( 'Импорт завершен успешно!',
                        'Внимание!', MB_OK + MB_ICONINFORMATION)
    else
        Application.MessageBox( 'Ошибка импорта файлов настройки!',
                            'Ошибка!', MB_OK + MB_ICONERROR)
end;

procedure TTestForm.SBtnExportConfigClick(Sender: TObject);
    var pmr: IRecorder;
begin
    pmr:= TTestPlugin( FPlugin).FIRecorder; //Ссылка на объект Recorder`a
    //выполнение и проверка рез-та экспорта
    if SUCCEEDED( pmr.ExportSettings( 'c:\temp\', 0)) then
        Application.MessageBox( 'Экспорт завершен успешно!',
                        'Внимание!', MB_OK + MB_ICONINFORMATION)
    else
        Application.MessageBox( 'Ошибка экспорта файлов настройки!',
                        'Ошибка!', MB_OK + MB_ICONERROR)
end;
```

3.5.11 Просмотр и изменение параметров КФ/ГФ

ПО «Recorder» использует четыре типа калибровочных (градуировочных) функций, у каждой функции свой перечень параметров:

1. Функция «Масштабный коэффициент»
   $f(x) = a \bullet x$, всего один параметр- коэффициент a.
2. «Линейная» функция
   $f(x) = a \bullet x + b$, два коэффициента a и b.

3. Функция «Таблица линейной интерполяции»
   функция производит линейную интерполяцию по таблице значений, параметрами является массив пар значений *x* и *f(x)*.
4. Функция «Полином n-ой степени»
   $f(x) = \sum_{i=0}^{n} a_i \bullet x^i$, параметрами является массив коэффициентов *a*

Один тег ПО «Recorder» может иметь одну калибровочную (моудльную) функцию и одну градуировочную (канальную) функцию одного из перечисленных выше типов. Градуировочная функция тега может отсутствовать вовсе, калибровочная присутствует всегда. У тега есть признак активности калибровочной функции. Если признак установлен, то калибровочная функция используется, иначе не используется.

Для отображения параметров калибровочной и градуировочной функций, на форме расположим страницу TSClbs. На странице TSClbs расположим компоненту ComboBoxTags типа TComboBox, для того, чтобы можно было выбрать итег, калибровочную или градуировочную функцию которого необходимо отобразить. Расположим на странице TSClbs компоненту TabControlClb типа TTabControl. Компонента TabControlClb будет служить для работы с кнопками переключения страниц, отображающих калибровочную и градуировочную функции. И калибровочная и градуировочная функции тега имеют один список параметров. Для отображения функций можно создать отдельный фрейм - FrameClbShow. На фрейме FrameClbShow расположим компоненты:

1. компоненту ComboBoxClbType типа TComboBox, она будет служить для показа типа функции («Масштабный коэффициент», «Линейная» и т.п.).
2. компоненту CheckBoxUse типа TCheckBox, она будет служить для отображения признака активности функции (актуально только для калибровочной функции)

Так как у каждого типа функции есть свой перечень параметров, то целесообразно создать для каждого типа функции отдельный фрейм. На фрейме FrameClbShow отображать фрейм той функции, которая установлена в теге.

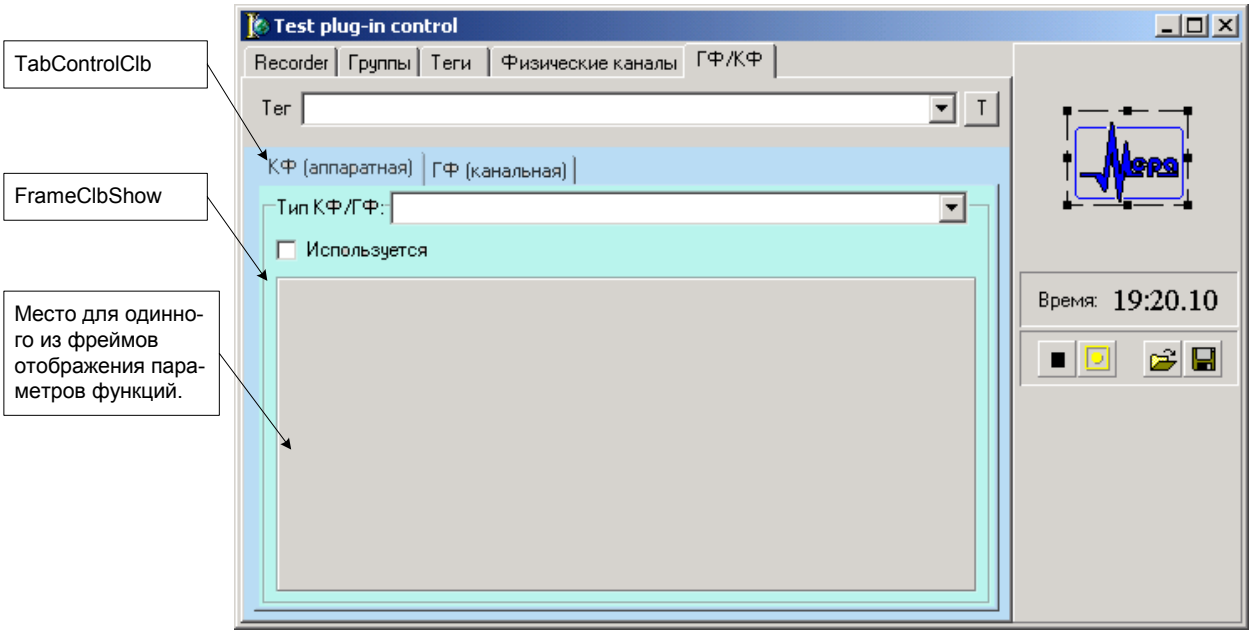

Рисунок 2. Вид страницы для отображения параметров калибровочной и градуировочной функций

Создадим следующие фреймы: FrameClbScale, FrameClbLine, FrameClbInt, FrameClbPol для функций: «Масштабный коэффициент», «Линейная», «Таблица линейной интерполяции», «Полином n-ой степени» соответственно.

Фрейм FrameClbScale будет отображать значение масштабного коэффициента. Фрейм FrameClbLine будет отображать два коэффициента. Фрейм FrameClbInt будет отображать таблицу пар значений. Фрейм FrameClbPol будет отображать массив коэффициентов. Для одинакового обращения к фреймам пусть все они реализуют метод UpdateClb. В этом методе фрейм получает ссылку на интерфейс градуировочной или калибровочной функции и должен отобразить её параметры.

Фрейм FrameClbShow будет реализовывать функцию ShowCalibration, в которой получит ссылку на тег и признак того, какая функция используется – калибровочная или градуировочная. В методе ShowCalibration будет запрашиваться у тега ссылка на объект калибровочной или градуировочной функции, будет определяться её тип и в зависимости от типа будет отображен необходимый фрейм, фрейму будет дана команда на показ параметров функции.

Создадим в главной форме метод, по которому будет выполняться отображение калибровочной и градуировочной функции для, выбранного в данный момент, тега. Это метод ShowCurrentCalibration.

```
function TTestForm.ShowCurrentCalibration: boolean;
var
    Index : integer;
    prec: IRecorder;
    ptag: ITag;
begin
    //получить индекс, выбраного в данный момент, тега
    Index := ComboBoxTags.ItemIndex;
    //если индекс на корректен, то...
    if (Index >= 0) and ( Index < ComboBoxTags.Items.Count) then
    begin
        //получить ссылку на Recorder
        prec:= TTestPlugin(FPlugin).FIRecorder;
        //получить ссылку на тег по имени
        pointer(ptag) := prec.GetTagByName(
        PChar(ComboBoxTags.Items[Index]));
        //отобразить КФ/ГФ по ссылке на тег, для этого вызывается фрейм
        //отображения, параметром ему передается ссылка на тег и
        //признак КФ (аппаратной функции), в зависимости от того,
        //какую страницу выбрал оператор (КФ или ГФ)
        Result:= FrameClbShow.ShowCalibration ( ptag,
                                        TabControlClb.TabIndex = 0);
    end
    else
        //Отображение пустого окна
        Result:= FrameClbShow.ShowCalibration ( nil,
                                    TabControlClb.TabIndex = 0);
end;
```

В этом методе выполняется получение выбранного, в данный момент, оператором тега и отображение калибровочной либо градуировочной функции тега. Главная форма обращается к фрейму FrameClbShow и методу ShowCalibration для показа КФ/ГФ.

Ниже приведен участок кода, в котором описываются класс фрейма TFrameClbShow. Полями класса являются специализированные фреймы для отображения параметров КФ/ГФ.

```
TFrameClbShow = class(TFrame)
…
…
private
    FClbScl : TFrameClbScale; {фрейм для отображения "Масштаба"}
    FClbLin : TFrameClbLine;  {фрейм для отображения "Линейная"}
    FClbInt : TFrameClbInt;   {фрейм для отображения "Таблица …"}
    FClbPol : TFrameClbPol;   {фрейм для отображения "Полином"}
…
…
end;
```

Участок кода метода FrameClbShow.ShowCalibration, приведен ниже.

```
function TFrameClbShow.ShowCalibration( ptag: ITag; const DevClb:
                                                    boolean): boolean;
var
    punk: IUnknown;
    pclb: ITransformer; //переменная для получения ссылки на КФ/ГФ
    v: OleVariant; // переменная для получения значений свойств
    bres: boolean;
begin
    try
        //если ссылка на тег не определена, то...нечго отображать
        if not Assigned(ptag) then raise Exception.Create('');
        if DevClb then //если КФ, то...
                //..получить ссылку КФ (аппаратную функцию)
                bres:= ptag.GetProperty( TAGPROP_DEVTARE, v)
        else//..получить ссылку ГФ (канальную функцию)
                bres:= ptag.GetProperty( TAGPROP_TARE, v);
        //если ошибка при получении КФ/ГФ,
        //либо объект не определен, то... нечго отображать
        if (not bres) then raise Exception.Create('');
           //получение из интерфейса IUnknown необходимого ITransformer
           punk:= v;
        if FAILED( punk.QueryInterface( ITransformer, pclb))  then
                raise Exception.Create('');

        //проанализировать и устновить признак использования
        if DevClb then //для КФ определен признак - свойство у тега
           CheckBoxUse.Checked :=
                ptag.GetProperty( TAGPROP_DEVSIGNALTYPE, v) and
                                    (v <> 0)
        else //для ГФ - всегда используется
           CheckBoxUse.Checked := true;
```

Первый участок кода выполняет получение у тега ссылки на калибровочную либо градуировочную функцию. Ссылка на функцию запрашивается как свойство. Далее у функции, если она калибровочная, запрашивается признак активности.

Во втором участке кода производится получение кода типа функции и ветвление по этому коду.

```
        //установить отображение строки типа КФ/ГФ
        ComboBoxClbType.ItemIndex := TTToIndex(pclb.GetTXType);

        case pclb.GetTXType of //ветвление по типу КФ/ГФ
            TXT_NULL: begin //активизация соответствующего фрйема
                SetCurrentEditor( nil );
                Enabled := false;
            end;
            TXT_SCALE: begin
                SetCurrentEditor( FClbScl);
                FClbScl.UpdateClb( pclb);
                Enabled := true;
            end;
            TXT_LINE: begin
                SetCurrentEditor( FClbLin );
                FClbLin.UpdateClb( pclb);
                Enabled := true;
            end;
            TXT_INTEPOLATETABLE: begin
                SetCurrentEditor( FClbInt);
                FClbInt.UpdateClb( pclb);
                Enabled := true;
            end;
            TXT_POLYNOME: begin
                SetCurrentEditor( FClbPol);
                FClbPol.UpdateClb( pclb);
                Enabled := true;
            end;
            else //неизвестный тип КФ/ГФ - фрейм деактивировать
            begin
                SetCurrentEditor( nil );
                Enabled := false;
            end;
        end;

        except
            //Ошибка работы с объектом тега или КФ/ГФ, либо
            //объект КФ/ГФ отсутствует - Фрейм деактивируется,
            //ничего не отображается
            SetCurrentEditor( nil );
            ComboBoxClbType.ItemIndex := -1;
            Enabled := false;
        end;
        Result := true;
    end;
```

Для каждого типа активизируется собственный специализированный фрейм. Метод SetCurrentEditor отображает фрейм. Метод UpdateClb, соответствующего фрейма, выпол-

няет отображение параметров функции. При возникновении ошибок ни один из специализированных фреймов не отображается.

Для получения и изменения параметров КФ/ГФ используются интерфейсы: IScale, ILinear, IInterpolate, IPolynomial.

В качестве примера приведем код установки тегу функции «Таблица линейной интерполяции», с тестовыми значениями параметров. Установка будет выполняться по нажатию кнопки на главной форме, кнопки BtnSetTestFunc. В методе главной формы, обработчике нажатия кнопки BtnSetTestFunc следует получить ссылку на, выбранный в данный момент, тег передать этот тег во фрйем FrameClbShow. Фрейм FrameClbShow при помощи специализированного фрейма TFrameClbInt создаст новый объект КФ/ГФ с тестовыми значениями параметров и установит функцию в тег.

Для этого в специализированном фрейме TFrameClbInt реализуем функцию SetTestFunc.

```
function TFrameClbInt.SetTestFunc( var tgdt: ITransformer): boolean;
var pint: IInterpolate;
begin
    try
        //Создание объекта ГФ/КФ
                if FAILED( CoCreateInstance(CLSID_InterpolateTransformer,
                                nil, CLSCTX_ALL, IInterpolate, pint)) then
                 raise Exception.Create('');

        //Установка трех узлов (элементов) "Таблицы…"
        if FAILED( pint.SetNode( 2, 10, -1)) then raise Exception.Create('');
        if FAILED( pint.SetNode( 4, 100, -1)) then raise Exception.Create('');
        if FAILED( pint.SetNode( 8, 1000, -1)) then raise Exception.Create('');

        //получение общего интерфейса
        if FAILED( pint.QueryInterface( ITransformer, tgdt)) then
                        raise Exception.Create('');

        Result := true;
    except
        tgdt := nil;
        Result := false;
    end;
end;
```

В методе SetTestFunc выполняется создание COM объекта КФ/ГФ, инициализация функции тестовыми значениями, производится получение, общего интереса ITransformer. В результате метод SetTestFunc параметром возвращает ссылку на вновь созданный объект КФ/ГФ. Далее следует функцию установить тегу.

Установку функции тегу можно выполнять в методе фрейма FrameClbShow. Только FrameClbShow имеет доступ к специализированному фрейму TFrameClbInt и может вызвать его метод SetTestFunc. Реализуем у фрейма FrameClbShow метод SetTestFunc. Код метода приведен ниже. Параметрами этого метода являются: ссылка на тег, для которого необходимо установить КФ/ГФ и признак, указывающий какая из функций используется, КФ либо ГФ.

```
function TFrameClbShow.SetTestFunc ( ptag: ITag;
                                     const DevClb: boolean): boolean;
var
    pclb: ITransformer;
    v: OleVariant;
begin
    try
        //если ссылка на тег не определена, то... установить ничего нельзя
        if not Assigned(ptag) then raise Exception.Create('');

        //В качестве тестовой устанавливается ГФ/КФ "Таблица…",
        //производится проверка корректности создания объекта ГФ/КФ.
        if (not FClbInt.SetTestFunc( pclb)) or
          (not Assigned(pclb)) then raise Exception.Create('');

        v := pclb; //подготовка параметра для установки ГФ/КФ.

        if DevClb then //если КФ, то...
           //..установить КФ (аппаратную функцию)
           Result:= ptag.SetProperty( TAGPROP_DEVTARE, v)
        else
           begin
              //...подготовка к установке ГФ...
              if not ptag.Notify(TN_ERASETAGTX,0) then
                     raise Exception.Create('');
              //..установить ГФ (канальную функцию)
              Result:= ptag.SetProperty( TAGPROP_TARE, v);
           end;

        //если ошибка при установке КФ/ГФ,
        if (not Result) then raise Exception.Create('');

        //Отображение установленной функции
        Result := ShowCalibration( ptag, DevClb);
    except
       Result := false;
    end;
end;
```

Установка тегу новой КФ/ГФ выполняется через изменений свойств. Если новая функция корректно установлена, то выполняется отображение её параметров.

Ниже приведен участок кода метода BtnSetTestFuncClick главной формы, который обрабатывает нажатие кнопки установки тестовой функции КФ/ГФ. В этом методе выполняется получение тега, выбранного в данный момент, вход в режим настройки, вызов метода SetTestFunc фрейма FrameClbShow, и выход из режима настройки.

```
procedure TTestForm.BtnSetTestFuncClick(Sender: TObject);
var
    Index : integer;
    prec: IRecorder;
    ptag: ITag;
begin
    //получить индекс, выбраного в данный момент, тега
    Index := ComboBoxTags.ItemIndex;
    //если индекс на корректен, то...
    if (Index >= 0) and ( Index < ComboBoxTags.Items.Count) then
    begin
        //получить ссылку на Recorder
        prec:= TTestPlugin(FPlugin).FIRecorder;
        //получить ссылку на тег по имени
        pointer(ptag) := prec.GetTagByName(
        PChar(ComboBoxTags.Items[Index]));

        //обязательный переход к режиму настройки
        if BeginConfigure then
        begin
            try
                //проверка результата установки новой КФ/ГФ
                if not FrameClbShow.SetTestFunc ( ptag,
                                TabControlClb.TabIndex = 0) …
            finally
                EndConfigure; //обязательный выход из режима настройки
            end;
        end
        else //не удалось перейти в режим настройки
        …
    end
end;
```

3.5.12 Компиляция и отладка

Процесс создания plug-in`а завершен. Полученный код должен компилироваться. Полная распечатка программного кода приведена в приложении.

3.6 Plug-in, генерирующий сигналы

Модуль plug-in`а имеет возможность получать и обрабатывать измеренные данные. Результаты обработки могут быть отображены оператору, записаны в файл, либо записаны в виртуальный тег. Запись данных в виртуальные теги может быть использована для моделирования процесса измерения, в ситуации, когда аппаратные измерительные устройства отсутствуют.

3.6.1 Постановка задачи

В качестве примера необходимо разработать тестовый plug-in, который выполняет следующие функции:

1. создание одного виртуального тега;
2. в режиме измерения генерирование гармонического сигнала (синусоида) и запись данных, этого сигнала, в виртуальный тег;

В качестве основы для разработки нового plug-in`а можно использовать код, написанный в пп. 3.1.

Дальнейшая доработка библиотеки и самого класса plug-in`а должна производится в соответствии с описанными выше требованиями.

3.6.2 Создание виртуального тега

Создание виртуального тега выполняется так, как описано в пп. 3.5.8.

```
function TTestPlugin.Execute: boolean;
var v: OleVariant;
begin
    if BeginConfigure then // если удалось перейти к настройке, то...
    begin
        try
          //добавление нового тега
          pointer(FTag):= FIRecorder.CreateTag( PChar(VT_Name),
                                                LS_VIRTUAL, FTag);
          if Assigned( FTag) then //если удалось создать, то...
          begin
            {Установка требуемой частоты дискретизации}
            FTag.SetFreq( VT_SF);
            //Установка типа данных тега
            v:= 0;
            v:= VarAsType(v, varDouble);
            FTag.SetProperty( TAGPROP_DATATYPE, v);

            //становка минимума и максимума
            v := VT_Amplitude * 1.1 / 2; //максимум от значения амплитуды
            v:= VarAsType(v, varDouble);
            FTag.SetProperty( TAGPROP_MINVALUE , v);

            v := - VT_Amplitude * 1.1 / 2; //минимум от значения амплитуды
            v:= VarAsType(v, varDouble);
            FTag.SetProperty( TAGPROP_MAXVALUE, v);
          end;
        finally
          EndConfigure; //проццесс изменения настройки завершен
        end;
    end
    else
        …//Обработка ошибки. Ошибка перехода в режим настройки!
    Result:= true;
end;
```

Plug-in может создавать свой рабочий тег непосредственно в методе Execute. Новый тег создается с именем, которое указывается константой VT_Name. После создания тега, устанавливается несколько его свойств:

1. Требуемая частота дискретизации тега. Эта та частота, с которой будут генерироваться значения для тега. Частоты указывается константой VT_SF.
2. Требуемый тип данных буфера тега.
3. Минимальное и максимальное значения на шкале графика отображения данных тега. Пусть в тег будет генерироваться гармонический сигнал с амплитудой VT_Amplitude. Тогда минимум и максимум рассчитывается на основе амплитуды.

3.6.3 Запись данных в теги

Запись данных в тег осуществляется блоками, при помощи метода тега ITag.PushData. Тип данных в блоке установлен при создании тега. Размер блока, количество элементов, зависит от периода обновления ПО «Recorder». Размер блока можно получить при помощи свойства TAGPROP_BUFFSIZE.

Участок кода записи данных в тег приведен ниже.

```
//Установить блок данных в буфер тега...
FTag.PushData( pointer(FDataBuffer)^, -1);
```

здесь FTag – ссылка на тег, FDataBuffer – динамический массив действительных чисел (тип double). Последний параметр функции – резерв, его значение должно быть -1.

3.6.4 Генерация данных

Наиболее простой способ генерации данных это генерация по таймеру. То есть plug-in должен создать таймер (TTimer), и реализовывать метод обработки событий таймера.

В методе обработки событий от таймера необходимо выполнить следующее:

1. Подсчитать точное время, прошедшее от предыдущего события от таймера. События от таймера являются наименее приоритетными в очереди оконных событий, поэтому их поступление может быть не точным (по времени).
2. В зависимости от частоты дискретизации, установленной тегу, и размера блока, следует подсчитать какое количество блоков необходимо записать в тег, в данный момент.
3. Подготовить необходимое количество блоков данных. Синусоида генерируется при помощи функции sin. В зависимости от частоты дискретизации, установленной тегу, и частоты генерируемого сигнала, номера значения в блоке рассчитывается значение угла для функции sin.

Пусть генерация данных производится в методе TTestPlugin.OnTimer. Этот метод должен вызываться по событиям таймера. Код метода приведен ниже.

```
procedure TTestPlugin.OnTimer(Sender: TObject);
var
    CurTime: DWORD; {Текущее время в миллисекундах }
    DeltaTime: integer; {Время прошедшее с последней операции записи}
    BufferSize: integer;{Размер блока данных}
    n, i: integer;      {Счестчики циклов, по блокам и по элементам блоков}
    BlocksCount: integer; {Кол-во блоков, которые необходимо записать}
begin
    {Текущее время в миллисекундах от запуска компьютера}
    CurTime:= GetTickCount();
    //Вычисляем время от последнего шага генерации данных.
    DeltaTime:= CurTime - FLastWriteTime;
    //Размер одного блока данных
    BufferSize:= Length( FDataBuffer);
    //По времени прошедшему от последнего шага генерации сигнала,
    //по частоте дискретизации и по размеру буфера определяем
    //кол-во блоков, которое необходимо передать в тег на данном шаге...
    BlocksCount:= Trunc(DeltaTime * VT_SF / 1000 / BufferSize);
    //Если кол-во блоков не нулевое, то  необходимо запомнить
    //время текущего шага записи как последнего
    if BlocksCount >= 1 then
        FLastWriteTime := CurTime; //Запоминаем время текущего шага
    for n:= 0 to BlocksCount - 1 do
    begin
        //В цикле заполняем каждый блок, каждое значение в блоке
        for i:= 0 to BufferSize - 1 do
        begin
            FPhi := FPhi + FAlpha; ////Расчитываем угол, для синуса, где…
            //FAlpha - это изменение угла от одного до другого значения
            //(генерируемых) данных в буфере, зависит от частоты
            //генерируемого сигнала и от частоты дискретизации тега.

            //Периодический сигнал синус, вычитание периода...
            if (FPhi >= 2 * Pi) then FPhi := FPhi - 2 * Pi;

            //Расчет значения генерируемых данных (по углу)
            FDataBuffer[i] := (VT_Amplitude / 2) * sin( FPhi);
        end;
        //Установить блок данных в буфер тега...
        FTag.PushData( pointer(FDataBuffer)^, -1);
    end;
end;
```

3.6.5 Инициализация, подготовка к генерации

Как было описано выше, для генерации данных используется таймер. Таймер – это компонента типа TTimer, экземпляр компоненты можно создать в конструкторе класса plug-in`a. Там же и проинициализировать его, установить период событий и метод, который будет вызваться по событиям таймера.

```
constructor TTestPlugin.Create;
begin
  FTag:= nil;
  FTimer:= TTimer.Create( nil);
  FTimer.OnTimer := OnTimer;
  FTimer.Interval := VT_UPDATE_PERIOD;
  FTimer.Enabled := false;
end;
```

Для генерации сигнала используется информация о размере буфера тега, которая зависит от настройки ПО «Recorder». По этой причине, после каждого изменения настройки необходимо получать новое значение размера буфера тега. Так же, для генерации сигнала используется внутренний временный буфер данных – динамический массив, размер которого равен размеру одного блока. Инициализацию динамического массива следует выполнять после изменения настройки ПО «Recorder».

Ниже приведен код метода класса plug-in`a, который будет вызываться по изменению настройки в ПО «Recorder».

```
procedure TTestPlugin.OnEndConfigure;
var
  BufferSize: integer;
  v: OleVariant;
begin
  //Получить размер одного блока из буфера тега.
  if Assigned( FTag) then
  begin
    //Получить у тега размер одного блока буфера
    FTag.GetProperty( TAGPROP_BUFFSIZE, v);
    BufferSize:= v;
    //Выделить блок памяти для формирования данных.
    SetLength( FDataBuffer, BufferSize);
  end;

  //Расчет приращения угла в одной точке измерения
  //Угол в радианах, зависит от частоты сигнала и
  //частоты дискретизации тега...
  FAlpha := 2*Pi*VT_SignalFrequency / VT_SF;

  FPhi:= 0;
end;
```

3.6.6 Компиляция и отладка

Выше были описаны несколько основных методов класса plug-in`a. Не были упомянуты методы:

1. TTestPlugin.Notify – в этом методе получение сообщений и вызов методов их обработки. Обрабатываются сообщения PN_ENTERRCCONFIG, PN_LEAVERCCONFIG, PN_RCSTART, PN_RCSTOP.

2. TTestPlugin.BeginConfigure, TTestPlugin.EndConfigure – эти методы служат для перехода в режим настройки и выхода из режима настройки.
3. TTestPlugin.OnStart, TTestPlugin.OnStop – эти методы обрабатывают события перехода в режим измерения и выхода из режима измерения. При переходе в режим измерения активизируется таймер генерации сигнала. При выходе из режима измерения таймер деактивируется.

Полная распечатка кода plug-in`а приведена в приложении. Этот код должен компилироваться.

4 Разработка plug-in`а при помощи Borland® C++ Builder™

4.1 Минимальный код plug-in`а.

Как описано в спецификации интерфейса PluginAPI модуль plug-in`а – это динамически линкуемая библиотека, которая экспортирует несколько функций и реализует класс plug-in`а.

4.1.1 Создание и настройка проекта

Для разработки plug-in`а необходимо:

1. создать проект динамически линкуемой библиотеки (dll). Новый проект создается при помощи Wizard`а, файлы, созданного проекта, необходимо сохранить; пусть тестовый plug-in имеет имя «ctest»; файл проекта соответственно имя «ctest.bpf», а файл с исходным кодом библиотеки: «ctest.cpp».
2. указать в качестве директории поиска («Search path») путь к интерфейсным файлам.

4.1.2 Описание класса plug-in`а

ПО «Recorder» работает с объектом plug-in`а. Библиотека plug-in`а должна описывать соответствующий класс и реализовывать его методы. В методах класса plug-in`а может быть реализована любая функциональность необходимая для решения поставленных задач. Класс plug-in`а может производить обработку, сохранение, отображение измеренных данных.

Таким образом, необходимо создать класс, пусть это будет класс с именем TTestPlugin. Данное имя не является обязательным, имя класса может быть любым (ограничением являются только ограничения на имена идентификаторов в C++Builder™).

Для описания нового класса создадим в проекте новый файл заголовка «TTestPlugin.h» и новый файл «TTestPlugin.cpp».

Класс TTestPlugin должен реализовать интерфейс IRecorderPlugin, по средствам этого интерфейса ПО «Recorder» взаимодействует с объектом plug-in`а. Так как IRecorderPlugin COM интерфейс, и он наследует IUnknown, то класс TTestPlugin должен реализовывать и интерфейс IUnknown.

Участок кода с описанием класса TTestPlugin приведен ниже. Класс plug-in`а должен реализовать каждый метод интерфейса IRecorderPlugin. На данном этапе разработки функциональность plug-in`а не известна, поэтому тела методов можно оставить пустыми. Для штатного запуска plug-in`а необходимо, чтобы методы create(), config(), execute(), canclose() возвращали в качестве результата «true», это будет означать соответственно, что инициализация, настройка, запуск рабочего режима выполнены успешно, и plug-in не препятствует закрытию приложения.

```
class TTestPlugin: public IRecorderPlugin
{
public:
    __fastcall TTestPlugin(void);
public: //IRecorderPlugin:
    //Создание плагина
    virtual bool STDMETHODCALLTYPE create(IRecorder* a_pOwner=NULL);
    //Конфигурирование
    virtual bool STDMETHODCALLTYPE config();
    // Вызов окна настройки
    virtual bool STDMETHODCALLTYPE edit();

    // Запуск
    virtual bool STDMETHODCALLTYPE execute();
    // Приостановка работы
    virtual bool STDMETHODCALLTYPE suspend();
    // Возобновление работы
    virtual bool STDMETHODCALLTYPE resume();
    // Уведомление о внешних событиях
    virtual bool STDMETHODCALLTYPE notify(DWORD a_dwCommand,
                                          DWORD a_dwData=0);
    // Получение имени
    virtual LPCSTR STDMETHODCALLTYPE getname();
    // Получить свойство
    virtual bool   STDMETHODCALLTYPE getproperty(DWORD a_dwPropertyID,
                                                 VARIANT& a_Value);
    // Задать свойство
    virtual bool   STDMETHODCALLTYPE setproperty(DWORD a_dwPropertyID,
                                                 VARIANT a_Value);
    // Узнать можно ли завершить работу плагина
    virtual bool   STDMETHODCALLTYPE canclose();
    // Завершить работу плагина
    virtual bool   STDMETHODCALLTYPE close();

private: //Методы и поля для реализации IUnknown
    DWORD FRefCount;
public:
    virtual HRESULT STDMETHODCALLTYPE QueryInterface(
            /* [in] */ REFIID riid,
            /* [iid_is][out] */ void __RPC_FAR *__RPC_FAR *ppvObject);
    virtual ULONG STDMETHODCALLTYPE AddRef( void);
    virtual ULONG STDMETHODCALLTYPE Release( void);
};
```

Реализация интерфейса IUnknown. Метод QueryInterface должен возвращать ссылку на требуемый интерфейс plug-in`a. Данный plug-in поддерживает только интерфейсы: IRecorderPlugin и IUnknown. Участок кода с методом QueryInterface приведен ниже. Методы AddRef и Release выполняют только отладочный подсчет ссылок. Метод Release (в данной реализации plug-in`a) не должен удалять экземпляр объекта, причина такого поведения описана в пп. 4.1.3.

```
HRESULT STDMETHODCALLTYPE TTestPlugin::QueryInterface(
      /* [in] */ REFIID riid,
      /* [iid_is][out] */ void __RPC_FAR *__RPC_FAR *ppvObject)
{
    if (!ppvObject) return E_FAIL;
    *ppvObject = NULL;

    if (riid == IID_IUnknown)
    *ppvObject = this;
    else if (riid == IID_IRecorderPlugin)
    *ppvObject = static_cast<IRecorderPlugin*> (this);

    if (*ppvObject)
    {
        reinterpret_cast<IUnknown*>(*ppvObject)->AddRef();
        return S_OK;
    }
    else
        return E_NOINTERFACE;
}
ULONG STDMETHODCALLTYPE TTestPlugin::AddRef( void)
{
        return ++ FRefCount;
}
ULONG STDMETHODCALLTYPE TTestPlugin::Release( void)
{
        return --FRefCount;
}
```

4.1.3 Создание объекта plug-in` a

Наиболее простой способ создания экземпляра plug-in`a – это создание статического глобального объекта. Инициализация такого объекта будет производиться по загрузке библиотеки. Удаление по выгрузке библиотеки. В ПО «Recorder» всегда будет передаваться ссылка на данный объект.

Так как удаление объекта plug-in`a выполняется автоматически, то сам объект себя удалять не должен, это касается особенностей реализации метода Release(). В этом методе для отладки оставлен код подсчета ссылок.

Глобальную переменную объект класса plug-in`a следует определить в файле «ctest.cpp», пусть эта переменная имеет имя «GPlugin».

4.1.4 Информация о plug-in` e

ПО «Recorder» имеет возможность отобразить оператору информацию о библиотеке plug-in`a, это строка описания, строка наименования разработчика, номер версии. Для того, чтобы всю эту информацию отобразить ПО «Recorder» запрашивает ее у самого plug-in`a (см. [3] пп. A.1.1.5.).

Для хранения описательной информации о plug-in`e можно использовать структуру PLUGININFO. Следует создать глобальную переменную типа PLUGININFO и инициализировать её данными, как приведено в примере ниже.

```
PLUGININFO tstinfo = {
  "Builder C++ plug-in test",        //name[101]
  "The test plug-in for programmer guide.", //describe[201];
  "NPP \"Mera\"",                    //vendor[201];
  1,                          //version;
  0                           //subversion;
};
```

4.1.5 Описание экспортируемых функций

В библиотеке необходимо описать (экспортировать) и реализовать функции:

1. GetPluginType(),
2. CreatePluginClass(),
3. GetPluginDescription(),
4. DestroyPluginClass(),
5. GetPluginInfo(),

в соответствии со спецификацией (см. [3] пп. А.1.1.). Функции предназначены для того, чтобы ПО «Recorder» могло получить доступ к объекту plug-in`a, а также получить информацию описывающую объект plug-in`a. Реализацию функций можно поместить непосредственно в код проекта библиотеки, либо можно создать новый модуль исходного кода.

Экспортируемые функции могут быть описаны и реализованы в файле библиотеки «ctest.cpp». Для описания функций следует использовать три директивы: **extern "C"**, **_export** и **_cdecl**. Директива **extern "C"** устанавливает необходимый формат имен функций. Директива **_export** описывает функции как экспортируемые. Директива **_cdecl** описывает необходимое соглашение о вызове функций.

Функция GetPluginType(). Эта функция должна возвращать тип plug-in`a, константу PLUGIN_CLASS.

Функция CreatePluginClass(). Эта функция должна возвращать ссылку на объект plug-in`a, то есть адрес переменной GPlugin.

Функция GetPluginDescription(). Эта функция должна возвращать ссылку на строку описания plug-in`a, то есть адрес переменной tstinfo.describe.

Функция DestroyPluginClass(). Так как объект plug-in`a удаляется автоматически, то тело этой функции пусто.

Функция GetPluginInfo(). Этьо функция получения расширенной информации о модуле plug-in`a. Информацию о plug-in`е следует взять из переменной tstinfo. Пример реализации данной функции приведен ниже.

```
extern "C" {
    void _export _cdecl GetPluginInfo(LPPLUGININFO lpPluginInfo)
    {
        if(lpPluginInfo)
        {
            //В структуру (параметр) копируются строки описания...
            strcpy(lpPluginInfo->name,     tstinfo.name);
            strcpy(lpPluginInfo->describe, tstinfo.describe);
            strcpy(lpPluginInfo->vendor,   tstinfo.vendor);
            //...и номера версий...
            lpPluginInfo->version=     tstinfo.version;
            lpPluginInfo->subversion=  tstinfo.subversion;
        }
    }
}
```

4.1.6 Компиляция и отладка

На данном этапе создан проект с минимальным необходимым кодом для запуска plug-in`a. Данный код является «скелетом», на основе которого можно создать любой по функциональности plug-in. Проект следует скомпилировать.

Для запуска plug-in`a под отладчиком IDE Borland® C++Builder™ необходимо:

1. указать полный путь и имя файла plug-in`a в файле настройки ПО «Recorder». Файл настройки (по умолчанию) «recorder.cfg» находится в корневой директории ПО «Recorder».
2. указать в IDE Borland® C++ Builder™ в параметрах запуска в качестве «Host Application» имя (и соответственно полный путь) программы «Recorder».

Для того чтобы убедиться, что ПО «Recorder» «вызывает» plug-in можно запустить режим отладки, предварительно установив несколько отладочных точек останова, к примеру, в теле методов: TTestPlugin::create(), TTestPlugin::execute(). После запуска режима отладки IDE Borland® C++ Builder™ укажет, что выполнение программы остановилось в теле метода TTestPlugin::create(), а за тем и в теле метода TTestPlugin::execute().

Если останов в отладочных точках останова не произошел, то необходимо убедиться в правильной настройке IDE Borland® C++Builder™ и корректности настройки ПО «Recorder», проверить содержимое файла «recorder.cfg» и убедиться, что имя библиотеки plug-in`a указано правильно.

5 Разработка plug-in`a при помощи Microsoft® Visual C++6.0

5.1 Минимальный код plug-in`a.

5.1.1 Создание проекта

5.1.2 Описание экспортируемых функций

5.1.3 Описание класса plug-in`a

5.1.4 Отладка

6 Список используемых документов

1. «Программный интерфейс DevAPI. Руководство программиста».
2. «ИВК MIC. Программа управления комплексом MIC Руководство пользователя».
3. «Программное обеспечение сбора измерительных данных «Recorder». Руководство программиста».
4. «Программный интерфейс настройки ПО «Recorder». Руководство программиста».
5. «Разработка ПО ОС ИИС для автоматизации испытаний».

7 Список используемых сокращений

COM – Component Object Model
ПО – программное обеспечение
СЕВ – система единого времени

A Приложение. Распечатка кода программы plug-in`a, разработанного при помощи Borland® Delphi

A.1 Plug-in с функциями обработки и отображения данных

A.1.1 Модуль проекта библиотеки «test2.pas»

```
{-------------------------------------------------------------------------}
{ Проект (Модуль) реслизации plug-in`a для измерительного ПО Recorder }
{ Компилятор: Borland Delphi 6.0                                          }
{ НПП "ООО Мера" 2004г.                                                   }
{-------------------------------------------------------------------------}

library test2;

uses
  Windows,
  SysUtils,
  Classes,
  blaccess in '..\interfaces\blaccess.pas',
  journal in '..\interfaces\journal.pas',
  modules in '..\interfaces\modules.pas',
  plugin in '..\interfaces\plugin.pas',
  recorder in '..\interfaces\recorder.pas',
  signal in '..\interfaces\signal.pas',
  tags in '..\interfaces\tags.pas',
  transf in '..\interfaces\transf.pas',
  transformers in '..\interfaces\transformers.pas',
  waitwnd in '..\interfaces\waitwnd.pas',
  PluginClass in 'PluginClass.pas',
  TestFormUnit in 'TestFormUnit.pas' {TestForm};

{$R *.res}

{-------------------------------------------------------------Экспортируемые функции}
{Ниже описаны и реализованы пять функций, которые должна экспортировать}
{библиотека plug-in`a для Recorder`a}

{Получение типа объекта, реализованного в билиотеке}
function GetPluginType: integer; cdecl;
begin
  Result:= PLUGIN_CLASS; {В данном случае, это тип объекта plug-in`a}
end;
{Функция создания экземпляра plug-in`a.}
{Функция объявлена таким образом, как будто она возвращает не интерфейс,
а указатель. Причина такого объявления обоснована в документе описания.}
function CreatePluginClass: pointer{IRecorderPlugin}; cdecl;
begin
  {Данный plug-in поддерживает создание всего одного экземпляра,
   поэтому поэтому просто возвращается ссылка на глобальный объект.}
  {Сам объект не создается здаесь! Объект создается при загрузке библиотеки.}
  Result := pointer(GPluginInstance);
end;
{Функция удаления plug-in`a.}
function DestroyPluginClass(piPlg: IRecorderPlugin): integer; cdecl;
begin
   {Так как объект один и глобальный, то он здесь не удаляется.}
   {Объект plug-in`a удаляет себя сам (как любой COM объект),
    если счетчик ссылок его интерфейсов станет равен нулю.}
   Result:= 0;
end;
{Функция получения строки описания plug-in`a}
function GetPluginDescription: LPCSTR; cdecl;
```

```
begin
   {Описание извлекается из глобальной описательной структуры}
   Result:= LPCSTR(GPluginInfo.Dsc);
end;
{Функция получения полного описания plug-in`a}
procedure GetPluginInfo(var lpPluginInfo: PLUGININFO); cdecl;
begin
   {Описание извлекается из глобальной описательной структуры}
   {Копирование строк производится именно функциями StrCopy()
    из-за того, что структура описания plug-in`a описана
    в языке C++}
  StrCopy( @lpPluginInfo.name, LPCSTR(GPluginInfo.Name));
  StrCopy( @lpPluginInfo.describe, LPCSTR(GPluginInfo.Dsc));
  StrCopy( @lpPluginInfo.vendor, LPCSTR(GPluginInfo.Vendor));
  lpPluginInfo.version:= GPluginInfo.Version;
  lpPluginInfo.subversion:= GPluginInfo.SubVertion;
end;

{---------------------Объявление экспортируемых функций-----------------------}
{Объявление строковых имен экспортируемых функций}
exports GetPluginType        name 'GetPluginType';
exports CreatePluginClass    name 'CreatePluginClass';
exports DestroyPluginClass   name 'DestroyPluginClass';
exports GetPluginDescription name 'GetPluginDescription';
exports GetPluginInfo        name 'GetPluginInfo';

end.
```

A.1.2 Модуль «PluginClass.pas»

```
{-------------------------------------------------------------------------}
{ Проект (Модуль) реслизации plug-in`a для измерительного ПО Recorder     }
{ Модуль класса тестового plug-in`a, описание и реализация                }
{ Компилятор: Borland Delphi 6.0                                          }
{ НПП "ООО Мера" 2004г.                                                   }
{-------------------------------------------------------------------------}

{В данном модуле описан и реализован класс TTestPlugin тестового plug-in`a.}

unit PluginClass;
interface
uses
   Windows,
   recorder, tags, plugin,
   TestFormUnit;

type
   {Класс тестового plug-in`a. Наследует классу TInterfacedObject,
    при этом наследуется реализация IUnknown и механизм подсчета
    ссылок соответсвенно. Класс реализует интерфейс IRecorderPlugin -
    интерфейс plug-in`a. Для отображения данных класс использует
    специальную форму (поле FTestForm).}
   TTestPlugin = class(TInterfacedObject, IRecorderPlugin)
   public
      {Конструктор}
      constructor Create;
      {Деструктор}
      destructor Destroy; override;

      {Метод запуска и останова измерения}
      function StartMeasure: boolean;
      function StopMeasure: boolean;
   public    //IRecorderPlugin
      // Создание плагина
      function _Create(pOwner: IRecorder): boolean; stdcall;
      // Конфигурирование
      function Config: boolean; stdcall;
      // Вызов окна настройки
      function Edit: boolean; stdcall;
```

```
      // Запуск - активизация работы plug-in`a
      function Execute: boolean; stdcall;
      // Приостановка работы
      function Suspend: boolean; stdcall;
      // Возобновление работы
      function Resume: boolean; stdcall;
      // Уведомление о внешних событиях
      function Notify(const dwCommand: DWORD;
                      const dwData: DWORD): boolean; stdcall;
      // Получение имени
      function GetName: LPCSTR; stdcall;
      // Получить свойство
      function GetProperty(const dwPropertyID: DWORD;
                           var Value: OleVariant): boolean; stdcall;
      // Задать свойство
      function SetProperty(const dwPropertyID: DWORD;
                           {const} Value: OleVariant): boolean; stdcall;

      // Узнать можно ли завершить работу плагина
      function CanClose: boolean; stdcall;
      // Завершить работу плагина
      function Close:  boolean; stdcall;
   protected
      {Ссылка на форму plug-in`a. Во избежание рекурсивных ссылок
       для работы с формой используется понижающее приведение типа.
       Переменная описана как TForm, а используется везде с приведением
       типа к TTestForm}
      FTestForm: TTestForm;
      {Ссылка на интерфейс для управления Recorder`ом}
      FIRecorder: IRecorder;

      {Метод получения массива ссылок на интерфейсы тегов Recorder`a}
      procedure GetDynTagsArray(var Tags: DynTagsArray);
   end;


var
  {В библиотеке создается всего один глобальный экземпляр plug-in`a, и }
  {соответсвенно есть переменная ссылка на интерфейс этого plug-in`a.}
  {Экземпляр plug-in`a создается по загрузке библиотеки, и должен удаляться,}
  {когда счетчик ссылок станет равен нулю, это должно происходить перед}
  {выгрузкой библиотеки (если подсчет ссылок организоан правильно).    }
  GPluginInstance: IRecorderPlugin = nil;

type
   {Тип для хранения информации о plug-in`e}
   {Этот тип удобнее использовать в Delphi, чем PLUGININFO}
  TInternalPluginInfo = record
    Name:           string; //наименование plug-in`a
    Dsc:            string;//строка описания
    Vendor:  string;//строка наименования фирмы
    Version: integer;//номер версии
    SubVertion:     integer;//номер подверсии
  end;

const
   {Глобальная переменная для храения описания plug-in`a.}
  GPluginInfo: TInternalPluginInfo = (
    Name:           'Delphi Тест';
    Dsc:            'Тестирование';
    Vendor:  'ООО НПП Мера';
    Version: 0;
    SubVertion:     1;
  );

implementation
uses
  SysUtils;

{Конструктор} {Создание формы тестового plug-in`a}
constructor TTestPlugin.Create;
```

```
begin
  FTestForm:= TTestForm.Create( self );
end;
{Деструктор} {Удаление формы тестового plug-in`a}
destructor TTestPlugin.Destroy;
begin
  FreeAndNil( FTestForm);
end;

//IRecorderPlugin
// Создание плагина
function TTestPlugin._Create(pOwner: IRecorder): boolean;
begin
  FIRecorder:= pOwner; // сохранение полученной ссылки на интерфейс Recorder`a
  FTestForm.Show();    // отображение формы
  Result:= true;
end;

// Конфигурирование
function TTestPlugin.Config: boolean;
begin
  Result:= true;
end;
// Вызов окна настройки
function TTestPlugin.Edit: boolean;
begin
  FTestForm.Show();    // отображение формы
  Result:= true;
end;
// Запуск - активизация работы plug-in`a
function TTestPlugin.Execute: boolean;
begin
  FTestForm.Show();    // отображение формы
  Result:= true;
end;
// Приостановка работы
function TTestPlugin.Suspend: boolean;
begin
  Result:= true;
end;
// Возобновление работы
function TTestPlugin.Resume: boolean;
begin
  Result:= true;
end;
// Уведомление о внешних событиях
function TTestPlugin.Notify(const dwCommand: DWORD; const dwData: DWORD): boolean;
var
  Tags: DynTagsArray;
begin
  case dwCommand of
    PN_ENTERRCCONFIG: begin // перешел в режим настройки
      {В то время пока производится изменение настройки,
      теги недоступны (потому, что они могут удаляться)}
      //сообщить фрме о начале изменения настройки
      FTestForm.BeginConfigure;
      Result:= true;
    end;
    PN_LEAVERCCONFIG: begin // вышел из режима настройки
      {Настройка завершена, и необходимо обновить список тегов}
      GetDynTagsArray(Tags); {получить массив тегов}
      {Оповестить форму о завершении изменения конфигурации и
      передать ей новый список ссылок на интерфейсы тегов}
      FTestForm.EndConfigure(Tags);
      Result:= true;
    end;
    PN_SHOWINFO: begin  //команда для отображения собственной формы
      FTestForm.Show();    // отображение формы
      Result:= true;
    end;
    else
```

```
      Result:= false; //прочие сообщения не обрабатываем
  end;
end;

// Получение имени
function TTestPlugin.GetName: LPCSTR;
begin
   Result:= LPCSTR(GPluginInfo.Name); {Из глобальной структуры описания plug-in`a}
end;

// Получить свойство
function TTestPlugin.GetProperty(const dwPropertyID: DWORD;
                  var Value: OleVariant): boolean;
begin
  case dwPropertyID of
     PLGPROP_INFOSTRING: begin    {Получить строку описания plug-in`a}
        Value := GPluginInfo.Dsc; {Из глобальной структуры описания plug-in`a}
        Result:= true;
     end;
     else
        Result:= false;
  end;
end;
// Задать свойство
function TTestPlugin.SetProperty(const dwPropertyID: DWORD;
                  {const} Value: OleVariant): boolean;
begin
  Result:= true;
end;

// Узнать можно ли завершить работу плагина
function TTestPlugin.CanClose: boolean;
begin
  Result:= true;
end;
// Завершить работу плагина
function TTestPlugin.Close:  boolean;
begin
  Result:= true;
end;

{Метод получения массива ссылок на интерфейсы тегов Recorder`a}
{Список тегов запрашивается у Recorder`a, через ссылку на его интерфейс.}
procedure TTestPlugin.GetDynTagsArray(var Tags: DynTagsArray);
var
  Count: integer;
  i: integer;
begin
  //получить кол-во тегов
  Count:= FIRecorder.GetTagsCount;
  //установить соответствующий размер динамического массива
  SetLength(Tags, Count);
  //последовательно получить ссылки на интерфейсы тегов
  for i:= 0 to Count-1 do
    //использование преобразования типа обосновано в описании
    Tags[i]:= ITag(FIRecorder.GetTagByIndex(i));
end;

{Методы запуска и останова измерения}
function TTestPlugin.StartMeasure: boolean;
begin
  {Для запуска необходимо передать сообщение Recorder`y}
  Result:= FIRecorder.Notify( RCN_VIEW, 0);
end;
function TTestPlugin.StopMeasure: boolean;
begin
  {Для останова необходимо передать сообщение Recorder`y}
  Result:= FIRecorder.Notify( RCN_STOP, 0);
end;

initialization
```

```
   {Код создания одного экземпляра объетка plug-in`a, объект создается и
   в глобальной переменной сохраняется ссылка на интерфейс объекта. Объект
   удаляется автоматически, когда глобальная ссылка освобождается, то есть
   когда библиотека plug-in`a выгружается.}
   GPluginInstance:= TTestPlugin.Create;
end.
```

A.1.3 Модуль «TestFormUnit.pas»

```
{-----------------------------------------------------------------------}
{ Проект (Модуль) реслизации plug-in`a для измерительного ПО Recorder }
{ Модуль описания класса формы тестового plug-in`a                      }
{ Компилятор: Borland Delphi 6.0                                        }
{ НПП "ООО Мера" 2004г.                                                 }
{-----------------------------------------------------------------------}

unit TestFormUnit;

interface

uses
  Windows, Messages, SysUtils, Variants,
  Classes, Graphics, Controls, Forms,
  Dialogs, ComCtrls, ExtCtrls, Buttons,
  ImgList, StdCtrls,
  tags, blaccess;

{ В plug-in`e, точнее в форме используется два различных метода получения измеренных }
{ данных. Первый метод - это получение вектора (последнего блока) и вычисление сред- }
{ него. Второй метод - это получение сразу среднего.                                  }
{ Символ DATATHROUGHVECTOR позволяет управлять выбором используемого метода. Если    }
{ символ определен, то используется метод первый, если символ не определен, то       }
{ используется метод второй.                                                         }

{$DEFINE DATATHROUGHVECTOR}

type
   {Тип динамического массива интерфейсов на тег}
   DynTagsArray = array of ITag;

type
  //Класс формы тестового Plug-in`a. Основные выполняемые действия:
  //отображение списака имен и списка строк описаний тегов, отображение
  //и периодическое обнволение измеряемых значений для тегов.
  //Обновление измеряемых значений производится по событиям таймера DataUpdateTimer.
  //Форма хранит список тегов.
  TTestForm = class(TForm)
    TagsListView: TListView;
    DataUpdateTimer: TTimer;
    ControlPanel: TPanel;
    Bevel2: TBevel;
    Bevel1: TBevel;
    LogoImage: TImage;
    TimeLabel: TLabel;
    SBtnStop: TSpeedButton;
    SBtnView: TSpeedButton;
    Label1: TLabel;
    TagsImageList: TImageList;
    TimeTimer: TTimer;
    procedure DataUpdateTimerTimer(Sender: TObject);
    procedure TimeTimerTimer(Sender: TObject);
    procedure SBtnStopClick(Sender: TObject);
    procedure SBtnViewClick(Sender: TObject);
    procedure FormCreate(Sender: TObject);
  public
    //Метод подготовки формы к изменению конфигурации
    function  BeginConfigure: boolean;
    //Метод подготовки формы после того, как конфигурация была изменена
    procedure EndConfigure( const Tags: DynTagsArray);
```

```
    {Метод подготовки формы после изменения списка тегов}
    procedure PrepareTagsView;

    //Конструктор формы переопределен для того, чтобы форму
    //можно было инициализировать ссылкой на объект plug-in`a
    constructor Create( const Plugin: TObject); reintroduce;
  private
    {Динамический массив используемых в данный момент тегов}
    FTags: DynTagsArray;
    {ссылка на объект plug-in`a}
    {ссылка типа TObject, для того, чтобы избежать рекурсивных использований модулей}
    FPlugin: TObject;

    {Метод обновления отображения измеренных данных}
    procedure DataUpdate;
{$IFDEF DATATHROUGHVECTOR}
    {Метод получения у тега измеренных данных в виде строки.}
    {Получение производится через вектор.}
    function GetDataThroughVector(Tag: ITag; var StrTimeStamp: string): string;
    {Получение временного штампа для блока данных с номером nblock,}
    {доступ к блоку данных осуществляется по средствам IBlock}
    function GetTimeStamp( IBlock: IBlockAccess; const nblock: integer): string;
{$ELSE}
    {Метод получения у тега измеренных данных в виде строки.}
    {Получение производится через среднее.}
    function GetDataThroughAOD(Tag: ITag): string;
{$ENDIF}
  end;

implementation
{$R *.dfm}
uses
   PluginClass;

//Конструктор формы переопределен для того, чтобы форму
//можно было инициализировать ссылкой на объект plug-in`a
constructor TTestForm.Create( const Plugin: TObject);
begin
  inherited Create( nil);
  FPlugin:= Plugin; //сохранить ссылку на объект plug-in`a
end;

//Метод подготовки формы к изменению конфигурации
function  TTestForm.BeginConfigure: boolean;
begin
  DataUpdateTimer.Enabled := false; // остановить таймер обновления
  SetLength(FTags, 0); //очистить список внутренний тегов
  Result:= true;
end;

//Метод подготовки формы после того, как конфигурация была изменена
procedure TTestForm.EndConfigure( const Tags: DynTagsArray);
var i: integer;
begin
  //обновление внутреннего списка тегов
  SetLength(FTags, Length(Tags));
  for i := Low(Tags) to High(Tags) do
  begin
    FTags[i] := Tags[i];
  end;
  //обновление списка отображающего теги (имена, описания и данные)
  PrepareTagsView;
  //активизация таймера для отображения данных
  DataUpdateTimer.Enabled := true;
end;

{Метод подготовки формы после изменения списка тегов}
procedure TTestForm.PrepareTagsView;
var
  i: integer;
  li: TListItem;
```

```
  vDsc: OleVariant;
begin
  //Список тегов на форме очистить
  TagsListView.Clear;
  //и по всему списку ссылок на теги...
  for i:= Low(FTags) to High(FTags) do
  begin
    li:= TagsListView.Items.Add; //добавить элемент в список отображения
    if ( Assigned(FTags[i])) then
    begin
       //имя тега
       li.Caption:= FTags[i].GetName;
       //строка описания тега
       if FTags[i].GetProperty( TAGPROP_DESCRIBE, vDsc) then
          li.SubItems.Add(vDsc)
       else
          li.SubItems.Add('');
       //и оставить место для значения данных тега
       li.SubItems.Add('');
       //и оставить место для значения временного штампа тега
       li.SubItems.Add('');
    end;
  end;
end;

//Обработчик событий таймера обновления данных
procedure TTestForm.DataUpdateTimerTimer(Sender: TObject);
begin
  DataUpdate;
end;

{Метод обновления отображения измеренных данных}
procedure TTestForm.DataUpdate;
var
  i: integer;
  li: TListItem;
  StrData: string;
  StrTimeStamp: string;
begin
  //цикл по всему списку тегов
  for i:= Low(FTags) to High(FTags) do
  begin
    //проверка корректности таблицы TagsListView и
    //корректности ссылки на тег
    if (TagsListView.Items.Count > i) and Assigned(FTags[i]) then
    begin
      //получение ссылки на элемент таблицы
      li:= TagsListView.Items[i];

{$IFDEF DATATHROUGHVECTOR} {получить блок из буфера}
      //получение строкового представления измеренных
      //данных тега и штампа времени
      StrData:= GetDataThroughVector(FTags[i], StrTimeStamp);

{$ELSE} {получить обработанные данные}
      StrData:= GetDataThroughAOD( FTags[i]);
      StrTimeStamp := TimeToStr(Now);

{$ENDIF}
      //установка данных в таблицу для отображения
      if li.SubItems.Count >= 2 then
        li.SubItems[1] := StrData;
      if li.SubItems.Count >= 3 then
        li.SubItems[2] := StrTimeStamp;
    end;
  end;
end;


{$IFDEF DATATHROUGHVECTOR}
{Метод получения у тега измеренных данных в виде строки.}
```

```
{Получение производится через буфер по блокам.}
function TTestForm.GetDataThroughVector(Tag: ITag;
                                        var StrTimeStamp: string): string;
var
  IBlock: IBlockAccess;     //интерфейс блочного доступа к данным
  bl_size: integer;  //размер одного блока данных
  bl_count: integer;        //кол-во блоков данных
  data: array of double; //динамический массив для чтения блока
  i : integer;
  aod: double;              //вычисляемое средее арифметическое
begin
  //получить по ссылке на тег, ссылку на
  //интерфейс блочного доступа
  Tag.QueryInterface( IBlockAccess, IBlock);
  //блокировать буфер с измеренными данными, это обязательное
  //действие; так как буфер постоянно пополняется измеренными
  //данными - необходимо остановить изменение вектора на
  //время отображения данных.
  IBlock.LockVector;
  try
    //получить размер блока
    bl_size:=  IBlock.GetBlocksSize;
    //получить кол-во блоков
    bl_count:= IBlock.GetBlocksCount;
    //проверка
    if (bl_count <> 0) and (bl_size <> 0) then
    begin
      {при необходимости....}
      if (Length(data) <> bl_size) then
        SetLength(data, bl_size); {изменение размера массива данных}
      //чтение последнего блока измеренных данных
      IBlock.GetVectorR8( (pointer(data))^, bl_count - 1, bl_size, true);

      //вычисление среднего арифметического
      aod:= 0;
      for i:= 0 to bl_size - 1 do
        aod := aod + data[i];
      aod:= aod / bl_size;

      //формирование строки со значением
      Result := FloatToStr( aod);
      //получить временной штам текущих данных
      StrTimeStamp:= GetTimeStamp(IBlock, bl_count - 1);
    end
    else
      Result:= '';
  finally
    //разблокировать вектор
    IBlock.UnlockVector;
  end;
end;
{$ENDIF}

{$IFDEF DATATHROUGHVECTOR}
{Получение временного штампа для блока данных с номером nblock,}
{доступ к блоку данных осуществляется по средствам IBlock}
function TTestForm.GetTimeStamp( IBlock: IBlockAccess;
                                 const nblock: integer): string;
type
  FTR = record
    case integer of
      0: (itime: int64);
      1: (ftime: FILETIME);
  end;
var
  CTS: double;
  temp_time: FTR;
  systime: SYSTEMTIME; //преобразовать время в необходимый формат
begin
  CTS := IBlock.GetBlockUTSTime( nblock);
  try
```

```
    CTS := CTS * 1000;
    temp_time.itime := Round( CTS);
    temp_time.itime := temp_time.itime * 10000;
    FileTimeToSystemTime( temp_time.ftime, systime);
    //и в строку для отображения в таблице
    Result:= TimeToStr (SystemTimeToDateTime( systime));
  except
  end;
end;
{$ENDIF}

{$IFNDEF DATATHROUGHVECTOR}
{Метод получения у тега измеренных данных в виде строки.}
{Получение производится через среднее.}
function TTestForm.GetDataThroughAOD(Tag: ITag): string;
var vData: OleVariant;
begin
  //Получение среднего арифметического как одного из свойств тега
  if Tag.GetProperty(TAGPROP_ESTIMATE, vData) then
     Result:= vData
  else
     Result:= '';
end;
{$ENDIF}

procedure TTestForm.TimeTimerTimer(Sender: TObject);
begin
  TimeLabel.Caption := TimeToStr( Now);
end;

procedure TTestForm.SBtnStopClick(Sender: TObject);
begin
  TTestPlugin(FPlugin).StopMeasure;
end;

procedure TTestForm.SBtnViewClick(Sender: TObject);
begin
  TTestPlugin(FPlugin).StartMeasure;
end;

procedure TTestForm.FormCreate(Sender: TObject);
begin
  LogoImage.Hint:= 'ООО "НПП Мера"'+#13#10+'т. 516-89-16'+#13#10+'web:
www.nppmera.ru';
end;

end.
```

A.2 Plug-in со специализированным формуляр отображения ПО «Recorder»

A.2.1 Модуль проекта библиотеки «test_ivf.pas»

```
{-------------------------------------------------------------------}
{ Проект (Модуль) реслизации plug-in`a для измерительного ПО Recorder }
{ Компилятор: Borland Delphi 6.0                                    }
{ НПП "ООО Мера" 2004г.                                             }
{-------------------------------------------------------------------}

library test_ivf;

uses
  Windows,
  SysUtils,
  Classes,
  blaccess in '..\interfaces\blaccess.pas',
  journal in '..\interfaces\journal.pas',
```

```
  modules in '..\interfaces\modules.pas',
  plugin in '..\interfaces\plugin.pas',
  recorder in '..\interfaces\recorder.pas',
  signal in '..\interfaces\signal.pas',
  tags in '..\interfaces\tags.pas',
  transf in '..\interfaces\transf.pas',
  transformers in '..\interfaces\transformers.pas',
  waitwnd in '..\interfaces\waitwnd.pas',
  PluginClass in 'PluginClass.pas',
  TestFormUnit in 'TestFormUnit.pas' {TestForm};

{$R *.res}

{-------------------------------------------------------------Экспортируемые функции}
{Ниже описаны и реализованы пять функций, которые должна экспортировать}
{библиотека plug-in`a для Recorder`a}

{Получение типа объекта, реализованного в билиотеке}
function GetPluginType: integer; cdecl;
begin
  Result:= PLUGIN_CLASS; {В данном случае, это тип объекта plug-in`a}
end;
{Функция создания экземпляра plug-in`a.}
{Функция объявлена таким образом, как будто она возвращает не интерфейс,
а указатель. Причина такого объявления обоснована в документе описания.}
function CreatePluginClass: pointer{IRecorderPlugin}; cdecl;
begin
  {Данный plug-in поддерживает создание всего одного экземпляра,
   поэтому поэтому просто возвращается ссылка на глобальный объект.}
  {Сам объект не создается здаесь! Объект создается при загрузке библиотеки.}
  Result := pointer(GPluginInstance);
end;
{Функция удаления plug-in`a.}
function DestroyPluginClass(piPlg: IRecorderPlugin): integer; cdecl;
begin
   {Так как объект один и глобальный, то он здесь не удаляется.}
   {Объект plug-in`a удаляет себя сам (как любой COM объект),
    если счетчик ссылок его интерфейсов станет равен нулю.}
   Result:= 0;
end;
{Функция получения строки описания plug-in`a}
function GetPluginDescription: LPCSTR; cdecl;
begin
   {Описание извлекается из глобальной описательной структуры}
   Result:= LPCSTR(GPluginInfo.Dsc);
end;
{Функция получения полного описания plug-in`a}
procedure GetPluginInfo(var lpPluginInfo: PLUGININFO); cdecl;
begin
   {Описание извлекается из глобальной описательной структуры}
   {Копирование строк производится именно функциями StrCopy()
    из-за того, что структура описания plug-in`a описана
    в языке C++}
  StrCopy( @lpPluginInfo.name, LPCSTR(GPluginInfo.Name));
  StrCopy( @lpPluginInfo.describe, LPCSTR(GPluginInfo.Dsc));
  StrCopy( @lpPluginInfo.vendor, LPCSTR(GPluginInfo.Vendor));
  lpPluginInfo.version:= GPluginInfo.Version;
  lpPluginInfo.subversion:= GPluginInfo.SubVertion;
end;

{---------------------Объявление экспортируемых функций-----------------------}
{Объявление строковых имен экспортируемых функций}
exports GetPluginType        name 'GetPluginType';
exports CreatePluginClass    name 'CreatePluginClass';
exports DestroyPluginClass   name 'DestroyPluginClass';
exports GetPluginDescription name 'GetPluginDescription';
exports GetPluginInfo        name 'GetPluginInfo';

end.
```

A.2.2 Модуль «PluginClass.pas»

```
{-----------------------------------------------------------------------}
{ Проект (Модуль) реслизации plug-in`a для измерительного ПО Recorder }
{ Модуль класса тестового plug-in`a, описание и реализация            }
{ Компилятор: Borland Delphi 6.0                                      }
{ НПП "ООО Мера" 2004г.                                               }
{-----------------------------------------------------------------------}

{В данном модуле описан и реализован класс TTestPlugin тестового plug-in`a.}

unit PluginClass;
interface
uses
   Windows,
   recorder, tags, plugin,
   TestFormUnit;

type
   {Класс тестового plug-in`a. Наследует классу TInterfacedObject,
    при этом наследуется реализация IUnknown и механизм подсчета
    ссылок соответсвенно. Класс реализует интерфейс IRecorderPlugin -
    интерфейс plug-in`a. Для отображения данных класс использует
    специальную форму (поле FTestForm).}
   TTestPlugin = class(TInterfacedObject, IRecorderPlugin, IVForm)
   public
      {Конструктор}
      constructor Create;
      {Деструктор}
      destructor Destroy; override;

   public    //IRecorderPlugin
      // Создание плагина
      function _Create(pOwner: IRecorder): boolean; stdcall;
      // Конфигурирование
      function Config: boolean; stdcall;
      // Вызов окна настройки
      function Edit: boolean; stdcall;
      // Запуск - активизация работы plug-in`a
      function Execute: boolean; stdcall;
      // Приостановка работы
      function Suspend: boolean; stdcall;
      // Возобновление работы
      function Resume: boolean; stdcall;
      // Уведомление о внешних событиях
      function Notify(const dwCommand: DWORD;
                   const dwData: DWORD): boolean; stdcall;
      // Получение имени
      function GetName: LPCSTR; stdcall;
      // Получить свойство
      function GetProperty(const dwPropertyID: DWORD;
                           var Value: OleVariant): boolean; stdcall;
      // Задать свойство
      function SetProperty(const dwPropertyID: DWORD;
                           {const} Value: OleVariant): boolean; stdcall;

      // Узнать можно ли завершить работу плагина
      function CanClose: boolean; stdcall;
      // Завершить работу плагина
      function Close:  boolean; stdcall;

   public //IVForm
      //Получить имя формы должно быть уникальным, используется при регистрации
      function IVForm.GetName = IVForm_GetName;
      //Инициализация формы
      function Init(pParent: IRecorder; hParent: HWND; lParam: longint): boolean;
stdcall;
      //Получить HWND формы
      function GetHWnd: HWND; stdcall;
      //Вызывается при закрытии формы
      function IVForm.Close = IVForm_Close;
```

```
      function Prepare: boolean; stdcall;
      function Update: boolean; stdcall;
      //Перерисовка формы
      function Repaint: boolean; stdcall;
      //Привязка к тегам рекордера
      function LinkTags( var pTagsList: TagsArray;
                         var nTagsCount: ULONG): boolean; stdcall;
      //Активизация формы
      function Activate: boolean; stdcall;
      //Деактивизация формы
      function Deactivate: boolean; stdcall;
      //Вызов окна редактирования
      function IVForm.Edit = IVForm_Edit;
      //События, уведомления, команды
      function IVForm.Notify = IVForm_Notify;
   protected
      //Получить имя формы должно быть уникальным, используется при регистрации
      function IVForm_GetName: LPCSTR; stdcall;
      //Вызывается при закрытии формы
      function IVForm_Close: boolean; stdcall;
      //Вызов окна редактирования
      function IVForm_Edit: boolean; stdcall;
      //События, уведомления, команды
      function IVForm_Notify(const dwCommand: DWORD; const dwData: DWORD): boolean;
stdcall;
   protected
      {Ссылка на форму plug-in`a. Во избежание рекурсивных ссылок
       для работы с формой используется понижающее приведение типа.
       Переменная описана как TForm, а используется везде с приведением
       типа к TTestForm}
      FTestForm: TTestForm;
      {Ссылка на интерфейс для управления Recorder`ом}
      FIRecorder: IRecorder;

      {Метод получения массива ссылок на интерфейсы тегов Recorder`a}
      procedure GetDynTagsArray(var Tags: DynTagsArray);
   end;


var
  {В библиотеке создается всего один глобальный экземпляр plug-in`a, и }
  {соответсвенно есть переменная ссылка на интерфейс этого plug-in`a.}
  {Экземпляр plug-in`a создается по загрузке библиотеки, и должен удаляться,}
  {когда счетчик ссылок станет равен нулю, это должно происходить перед}
  {выгрузкой библиотеки (если подсчет ссылок организоан правильно).    }
  GPluginInstance: IRecorderPlugin = nil;

type
   {Тип для хранения информации о plug-in`e}
   {Этот тип удобнее использовать в Delphi, чем PLUGININFO}
  TInternalPluginInfo = record
    Name:           string; //наименование plug-in`a
    Dsc:            string;//строка описания
    Vendor:  string;//строка наименования фирмы
    Version: integer;//номер версии
    SubVertion:     integer;//номер подверсии
  end;

const
   {Глобальная переменная для храения описания plug-in`a.}
  GPluginInfo: TInternalPluginInfo = (
    Name:           'Delphi Тест (Формуляр)';
    Dsc:            'Тестирование';
    Vendor:  'ООО НПП Мера';
    Version: 0;
    SubVertion:     1;
  );

implementation
uses
  SysUtils;
```

```
{Конструктор} {Создание формы тестового plug-in`a}
constructor TTestPlugin.Create;
begin
  FTestForm:= TTestForm.Create( nil );
end;
{Деструктор} {Удаление формы тестового plug-in`a}
destructor TTestPlugin.Destroy;
begin
  FreeAndNil( FTestForm);
end;

//IRecorderPlugin
// Создание плагина
function TTestPlugin._Create(pOwner: IRecorder): boolean;
begin
  FIRecorder:= pOwner; // сохранение полученной ссылки на интерфейс Recorder`a
  //FTestForm.Show();    // отображение формы
  Result:= true;
end;

// Конфигурирование
function TTestPlugin.Config: boolean;
begin
  Result:= true;
end;
// Вызов окна настройки
function TTestPlugin.Edit: boolean;
begin
//  FTestForm.Show();    // отображение формы
  Result:= true;
end;
// Запуск - активизация работы plug-in`a
function TTestPlugin.Execute: boolean;
begin
//  FTestForm.Show();    // отображение формы
  FIRecorder.RegisterIForm( Self, 0);
  Result:= true;
end;
// Приостановка работы
function TTestPlugin.Suspend: boolean;
begin
  Result:= true;
end;
// Возобновление работы
function TTestPlugin.Resume: boolean;
begin
  Result:= true;
end;
// Уведомление о внешних событиях
function TTestPlugin.Notify(const dwCommand: DWORD; const dwData: DWORD): boolean;
var
  Tags: DynTagsArray;
begin
  case dwCommand of
    PN_ENTERRCCONFIG: begin // перешел в режим настройки
      {В то время пока производится изменение настройки,
      теги недоступны (потому, что они могут удаляться)}
      //сообщить фрме о начале изменения настройки
      FTestForm.BeginConfigure;
      Result:= true;
    end;
    PN_LEAVERCCONFIG: begin // вышел из режима настройки
      {Настройка завершена, и необходимо обновить список тегов}
      GetDynTagsArray(Tags); {получить массив тегов}
      {Оповестить форму о завершении изменения конфигурации и
      передать ей новый список ссылок на интерфейсы тегов}
      FTestForm.EndConfigure(Tags);
      Result:= true;
    end;
    PN_SHOWINFO: begin  //команда для отображения собственной формы
```

```
      //FTestForm.Show();    // отображение формы
      Result:= true;
    end;
    else
      Result:= false; //прочие сообщения не обрабатываем
  end;
end;

// Получение имени
function TTestPlugin.GetName: LPCSTR;
begin
   Result:= LPCSTR(GPluginInfo.Name); {Из глобальной структуры описания plug-in`a}
end;

// Получить свойство
function TTestPlugin.GetProperty(const dwPropertyID: DWORD;
                  var Value: OleVariant): boolean;
begin
  case dwPropertyID of
     PLGPROP_INFOSTRING: begin    {Получить строку описания plug-in`a}
        Value := GPluginInfo.Dsc; {Из глобальной структуры описания plug-in`a}
        Result:= true;
     end;
     else
        Result:= false;
  end;
end;
// Задать свойство
function TTestPlugin.SetProperty(const dwPropertyID: DWORD;
                  {const} Value: OleVariant): boolean;
begin
  Result:= true;
end;

// Узнать можно ли завершить работу плагина
function TTestPlugin.CanClose: boolean;
begin
  Result:= true;
end;
// Завершить работу плагина
function TTestPlugin.Close:  boolean;
begin
  FIRecorder.UnregisterIForm( Self);
  Result:= true;
end;


//Инициализация формы
function TTestPlugin.Init(pParent: IRecorder; hParent: HWND;
                          lParam: longint): boolean;
begin
  FTestForm.ParentWindow := hParent;
  Result:= true;
end;
//Получить HWND формы
function TTestPlugin.GetHWnd: HWND;
begin
  Result:= FTestForm.Handle;
end;
function TTestPlugin.Prepare: boolean;
begin
  Result:= true;
end;
function TTestPlugin.Update: boolean;
begin
  Result:= true;
end;
//Перерисовка формы
function TTestPlugin.Repaint: boolean;
begin
  Result:= true;
```

```
end;
//Привязка к тегам рекордера
function TTestPlugin.LinkTags( var pTagsList: TagsArray;
                   var nTagsCount: ULONG): boolean;
begin
  Result:= true;
end;
//Активизация формы
function TTestPlugin.Activate: boolean;
begin
  FTestForm.Show;
  Result:= true;
end;
//Деактивизация формы
function TTestPlugin.Deactivate: boolean;
begin
  FTestForm.Hide;
  Result:= true;
end;

//Получить имя формы должно быть уникальным, используется при регистрации
function TTestPlugin.IVForm_GetName: LPCSTR;
begin
   Result:= LPCSTR(GPluginInfo.Name); {Из глобальной структуры описания plug-in`a}
end;
//Вызывается при закрытии формы
function TTestPlugin.IVForm_Close: boolean;
begin
  Result:= true;
end;
//Вызов окна редактирования
function TTestPlugin.IVForm_Edit: boolean;
begin
  Result:= false;
end;
//События, уведомления, команды
function TTestPlugin.IVForm_Notify(const dwCommand: DWORD;
                                   const dwData: DWORD): boolean;
var
  {Временная переменная для интерпретации параметра dwData}
  pRect: ^TRect absolute dwData; {как ссылки на структуру TRect}
begin
  {Ветвление по коду команды (сообщения)...}
  case dwCommand of
    VSN_RESIZE: begin {Получена команда на изменение размеров окна}
      {Установить новые размеры окна формы}
      FTestForm.SetBounds( pRect.Left, pRect.Top,
                           pRect.Right - pRect.Left,
                           pRect.Bottom - pRect.Top);
      Result:= true;
    end;
  else
    Result:= false;
  end;
end;


{Метод получения массива ссылок на интерфейсы тегов Recorder`a}
{Список тегов запрашивается у Recorder`a, через ссылку на его интерфейс.}
procedure TTestPlugin.GetDynTagsArray(var Tags: DynTagsArray);
var
  Count: integer;
  i: integer;
begin
  //получить кол-во тегов
  Count:= FIRecorder.GetTagsCount;
  //установить соответствующий размер динамического массива
  SetLength(Tags, Count);
  //последовательно получить ссылки на интерфейсы тегов
  for i:= 0 to Count-1 do
    //использование преобразования типа обосновано в описании
```

```
    Tags[i]:= ITag(FIRecorder.GetTagByIndex(i));
end;

initialization
   {Код создания одного экземпляра объетка plug-in`a, объект создается и
   в глобальной переменной сохраняется ссылка на интерфейс объекта. Объект
   удаляется автоматически, когда глобальная ссылка освобождается, то есть
   когда библиотека plug-in`a выгружается.}
   GPluginInstance:= TTestPlugin.Create;
end.
```

A.2.3 Модуль «TestFormUnit.pas»

```
{-----------------------------------------------------------------------}
{ Проект (Модуль) реслизации plug-in`a для измерительного ПО Recorder }
{ Модуль описания класса формы тестового plug-in`a                     }
{ Компилятор: Borland Delphi 6.0                                       }
{ НПП "ООО Мера" 2004г.                                                }
{-----------------------------------------------------------------------}

unit TestFormUnit;

interface

uses
  Windows, Messages, SysUtils, Variants,
  Classes, Graphics, Controls, Forms,
  Dialogs, ComCtrls, ExtCtrls, Buttons,
  ImgList, StdCtrls,
  tags, blaccess;

{ В plug-in`e, точнее в форме используется два различных метода получения измеренных }
{ данных. Первый метод - это получение вектора (последнего блока) и вычисление сред- }
{ него. Второй метод - это получение сразу среднего.                                  }
{ Символ DATATHROUGHVECTOR позволяет управлять выбором используемого метода. Если    }
{ символ определен, то используется метод первый, если символ не определен, то       }
{ используется метод второй.                                                         }

{$DEFINE DATATHROUGHVECTOR}

type
   {Тип динамического массива интерфейсов на тег}
   DynTagsArray = array of ITag;

type
  //Класс формы тестового Plug-in`a. Основные выполняемые действия:
  //отображение списака имен и списка строк описаний тегов, отображение
  //и периодическое обнволение измеряемых значений для тегов.
  //Обновление измеряемых значений производится по событиям таймера DataUpdateTimer.
  //Форма хранит список тегов.
  TTestForm = class(TForm)
    TagsListView: TListView;
    DataUpdateTimer: TTimer;
    ControlPanel: TPanel;
    Bevel1: TBevel;
    LogoImage: TImage;
    TimeLabel: TLabel;
    Label1: TLabel;
    TagsImageList: TImageList;
    TimeTimer: TTimer;
    procedure DataUpdateTimerTimer(Sender: TObject);
    procedure TimeTimerTimer(Sender: TObject);
    procedure FormCreate(Sender: TObject);
  public
    //Метод подготовки формы к изменению конфигурации
    function  BeginConfigure: boolean;
    //Метод подготовки формы после того, как конфигурация была изменена
    procedure EndConfigure( const Tags: DynTagsArray);

    {Метод подготовки формы после изменения списка тегов}
```

```
    procedure PrepareTagsView;

  private
    {Динамический массив используемых в данный момент тегов}
    FTags: DynTagsArray;

    {Метод обновления отображения измеренных данных}
    procedure DataUpdate;
{$IFDEF DATATHROUGHVECTOR}
    {Метод получения у тега измеренных данных в виде строки.}
    {Получение производится через вектор.}
    function GetDataThroughVector(Tag: ITag; var StrTimeStamp: string): string;
    {Получение временного штампа для блока данных с номером nblock,}
    {доступ к блоку данных осуществляется по средствам IBlock}
    function GetTimeStamp( IBlock: IBlockAccess; const nblock: integer): string;
{$ELSE}
    {Метод получения у тега измеренных данных в виде строки.}
    {Получение производится через среднее.}
    function GetDataThroughAOD(Tag: ITag): string;
{$ENDIF}
  end;

implementation
{$R *.dfm}

//Метод подготовки формы к изменению конфигурации
function  TTestForm.BeginConfigure: boolean;
begin
  DataUpdateTimer.Enabled := false; // остановить таймер обновления
  SetLength(FTags, 0); //очистить список внутренний тегов
  Result:= true;
end;

//Метод подготовки формы после того, как конфигурация была изменена
procedure TTestForm.EndConfigure( const Tags: DynTagsArray);
var i: integer;
begin
  //обновление внутреннего списка тегов
  SetLength(FTags, Length(Tags));
  for i := Low(Tags) to High(Tags) do
  begin
    FTags[i] := Tags[i];
  end;
  //обновление списка отображающего теги (имена, описания и данные)
  PrepareTagsView;
  //активизация таймера для отображения данных
  DataUpdateTimer.Enabled := true;
end;

{Метод подготовки формы после изменения списка тегов}
procedure TTestForm.PrepareTagsView;
var
  i: integer;
  li: TListItem;
  vDsc: OleVariant;
begin
  //Список тегов на форме очистить
  TagsListView.Clear;
  //и по всему списку ссылок на теги...
  for i:= Low(FTags) to High(FTags) do
  begin
    li:= TagsListView.Items.Add; //добавить элемент в список отображения
    if ( Assigned(FTags[i])) then
    begin
       //имя тега
       li.Caption:= FTags[i].GetName;
       //строка описания тега
       if FTags[i].GetProperty( TAGPROP_DESCRIBE, vDsc) then
          li.SubItems.Add(vDsc)
       else
          li.SubItems.Add('');
```

```
      //и оставить место для значения данных тега
      li.SubItems.Add('');
      //и оставить место для значения временного штампа тега
      li.SubItems.Add('');
    end;
  end;
end;

//Обработчик событий таймера обновления данных
procedure TTestForm.DataUpdateTimerTimer(Sender: TObject);
begin
  DataUpdate;
end;

{Метод обновления отображения измеренных данных}
procedure TTestForm.DataUpdate;
var
  i: integer;
  li: TListItem;
  StrData: string;
  StrTimeStamp: string;
begin
  //цикл по всему списку тегов
  for i:= Low(FTags) to High(FTags) do
  begin
    //проверка корректности таблицы TagsListView и
    //корректности ссылки на тег
    if (TagsListView.Items.Count > i) and Assigned(FTags[i]) then
    begin
      //получение ссылки на элемент таблицы
      li:= TagsListView.Items[i];

{$IFDEF DATATHROUGHVECTOR} {получить блок из буфера}
      //получение строкового представления измеренных
      //данных тега и штампа времени
      StrData:= GetDataThroughVector(FTags[i], StrTimeStamp);

{$ELSE} {получить обработанные данные}
      StrData:= GetDataThroughAOD( FTags[i]);
      StrTimeStamp := TimeToStr(Now);

{$ENDIF}
      //установка данных в таблицу для отображения
      if li.SubItems.Count >= 2 then
        li.SubItems[1] := StrData;
      if li.SubItems.Count >= 3 then
        li.SubItems[2] := StrTimeStamp;
    end;
  end;
end;


{$IFDEF DATATHROUGHVECTOR}
{Метод получения у тега измеренных данных в виде строки.}
{Получение производится через буфер по блокам.}
function TTestForm.GetDataThroughVector(Tag: ITag;
                                        var StrTimeStamp: string): string;
var
  IBlock: IBlockAccess;    //интерфейс блочного доступа к данным
  bl_size: integer;  //размер одного блока данных
  bl_count: integer;       //кол-во блоков данных
  data: array of double; //динамический массив для чтения блока
  i : integer;
  aod: double;             //вычисляемое средее арифметическое
begin
  //получить по ссылке на тег, ссылку на
  //интерфейс блочного доступа
  Tag.QueryInterface( IBlockAccess, IBlock);
  //блокировать буфер с измеренными данными, это обязательное
  //действие; так как буфер постоянно пополняется измеренными
  //данными - необходимо остановить изменение вектора на
```

```
  //время отображения данных.
  IBlock.LockVector;
  try
    //получить размер блока
    bl_size:=  IBlock.GetBlocksSize;
    //получить кол-во блоков
    bl_count:= IBlock.GetBlocksCount;
    //проверка
    if (bl_count <> 0) and (bl_size <> 0) then
    begin
      {при необходимости....}
      if (Length(data) <> bl_size) then
        SetLength(data, bl_size); {изменение размера массива данных}
      //чтение последнего блока измеренных данных
      IBlock.GetVectorR8( (pointer(data))^, bl_count - 1, bl_size, true);

      //вычисление среднего арифметического
      aod:= 0;
      for i:= 0 to bl_size - 1 do
        aod := aod + data[i];
      aod:= aod / bl_size;

      //формирование строки со значением
      Result := FloatToStr( aod);
      //получить временной штам текущих данных
      StrTimeStamp:= GetTimeStamp(IBlock, bl_count - 1);
    end
    else
      Result:= '';
  finally
    //разблокировать вектор
    IBlock.UnlockVector;
  end;
end;
{$ENDIF}

{$IFDEF DATATHROUGHVECTOR}
{Получение временного штампа для блока данных с номером nblock,}
{доступ к блоку данных осуществляется по средствам IBlock}
function TTestForm.GetTimeStamp( IBlock: IBlockAccess;
                                 const nblock: integer): string;
type
  FTR = record
    case integer of
      0: (itime: int64);
      1: (ftime: FILETIME);
  end;
var
  CTS: double;
  temp_time: FTR;
  systime: SYSTEMTIME; //преобразовать время в необходимый формат
begin
  CTS := IBlock.GetBlockUTSTime( nblock);
  try
    CTS := CTS * 1000;
    temp_time.itime := Round( CTS);
    temp_time.itime := temp_time.itime * 10000;
    FileTimeToSystemTime( temp_time.ftime, systime);
    //и в строку для отображения в таблице
    Result:= TimeToStr (SystemTimeToDateTime( systime));
  except
  end;
end;
{$ENDIF}

{$IFNDEF DATATHROUGHVECTOR}
{Метод получения у тега измеренных данных в виде строки.}
{Получение производится через среднее.}
function TTestForm.GetDataThroughAOD(Tag: ITag): string;
var vData: OleVariant;
begin
```

```
  //Получение среднего арифметического как одного из свойств тега
  if Tag.GetProperty(TAGPROP_ESTIMATE, vData) then
     Result:= vData
  else
     Result:= '';
end;
{$ENDIF}

procedure TTestForm.TimeTimerTimer(Sender: TObject);
begin
  TimeLabel.Caption := TimeToStr( Now);
end;

procedure TTestForm.FormCreate(Sender: TObject);
begin
  LogoImage.Hint:= 'ООО "НПП Мера"'+#13#10+'т. 516-89-16'+#13#10+'web:
www.nppmera.ru';
end;

end.
```

A.3 Plug-in real-time обработки измеренных данных

A.3.1 Модуль проекта библиотеки «test_rtm.pas»

```
{-------------------------------------------------------------------------}
{ Проект (Модуль) реслизации plug-in`a для измерительного ПО Recorder     }
{ Компилятор: Borland Delphi 6.0                                          }
{ НПП "ООО Мера" 2004г.                                                   }
{-------------------------------------------------------------------------}

library test_rtm;

uses
  Windows,
  SysUtils,
  Classes,
  PluginClass in 'PluginClass.pas',
  blaccess in '..\interfaces\blaccess.pas',
  journal in '..\interfaces\journal.pas',
  modules in '..\interfaces\modules.pas',
  plugin in '..\interfaces\plugin.pas',
  recorder in '..\interfaces\recorder.pas',
  signal in '..\interfaces\signal.pas',
  tags in '..\interfaces\tags.pas',
  transf in '..\interfaces\transf.pas',
  transformers in '..\interfaces\transformers.pas',
  waitwnd in '..\interfaces\waitwnd.pas';

{$R *.res}

{-----------------------------------------------------------------Экспортируемые функции}
{Ниже описаны и реализованы пять функций, которые должна экспортировать}
{библиотека plug-in`a для Recorder`a}

{Получение типа объекта, реализованного в билиотеке}
function GetPluginType: integer; cdecl;
begin
  Result:= PLUGIN_CLASS; {В данном случае, это тип объекта plug-in`a}
end;
{Функция создания экземпляра plug-in`a.}
{Функция объявлена таким образом, как будто она возвращает не интерфейс,
а указатель. Причина такого объявления обоснована в документе описания.}
function CreatePluginClass: pointer{IRecorderPlugin}; cdecl;
begin
  {Данный plug-in поддерживает создание всего одного экземпляра,
```

```
   поэтому поэтому просто возвращается ссылка на глобальный объект.}
  {Сам объект не создается здаесь! Объект создается при загрузке библиотеки.}
  Result := pointer(GPluginInstance);
end;
{Функция удаления plug-in`a.}
function DestroyPluginClass(piPlg: IRecorderPlugin): integer; cdecl;
begin
   {Так как объект один и глобальный, то он здесь не удаляется.}
   {Объект plug-in`a удаляет себя сам (как любой COM объект),
    если счетчик ссылок его интерфейсов станет равен нулю.}
   Result:= 0;
end;
{Функция получения строки описания plug-in`a}
function GetPluginDescription: LPCSTR; cdecl;
begin
   {Описание извлекается из глобальной описательной структуры}
   Result:= LPCSTR(GPluginInfo.Dsc);
end;
{Функция получения полного описания plug-in`a}
procedure GetPluginInfo(var lpPluginInfo: PLUGININFO); cdecl;
begin
   {Описание извлекается из глобальной описательной структуры}
   {Копирование строк производится именно функциями StrCopy()
    из-за того, что структура описания plug-in`a описана
    в языке C++}
  StrCopy( @lpPluginInfo.name, LPCSTR(GPluginInfo.Name));
  StrCopy( @lpPluginInfo.describe, LPCSTR(GPluginInfo.Dsc));
  StrCopy( @lpPluginInfo.vendor, LPCSTR(GPluginInfo.Vendor));
  lpPluginInfo.version:= GPluginInfo.Version;
  lpPluginInfo.subversion:= GPluginInfo.SubVertion;
end;

{----------------------Объявление экспортируемых функций-----------------------}
{Объявление строковых имен экспортируемых функций}
exports GetPluginType        name 'GetPluginType';
exports CreatePluginClass    name 'CreatePluginClass';
exports DestroyPluginClass   name 'DestroyPluginClass';
exports GetPluginDescription name 'GetPluginDescription';
exports GetPluginInfo        name 'GetPluginInfo';

end.
```

A.3.2 Модуль «PluginClass.pas»

```
{-------------------------------------------------------------------------}
{ Проект (Модуль) реслизации plug-in`a для измерительного ПО Recorder }
{ Модуль класса тестового plug-in`a, описание и реализация            }
{ Компилятор: Borland Delphi 6.0                                       }
{ НПП "ООО Мера" 2004г.                                                }
{-------------------------------------------------------------------------}

{В данном модуле описан и реализован класс TTestPlugin тестового plug-in`a.}

unit PluginClass;
interface
uses
   Windows,
   SyncObjs, Classes,
   recorder, tags, plugin, blaccess;

const
   FILENAMETEMLATE = 'data.dat';

type
   {Класс тестового plug-in`a. Наследует классу TInterfacedObject,
    при этом наследуется реализация IUnknown и механизм подсчета
    ссылок соответсвенно. Класс реализует интерфейс IRecorderPlugin -
    интерфейс plug-in`a. Для отображения данных класс использует
```

```
    специальную форму (поле FTestForm).}
   TTestPlugin = class(TInterfacedObject, IRecorderPlugin)
   public
      {Конструктор}
      constructor Create;
      {Деструктор}
      destructor Destroy; override;

   public    //IRecorderPlugin
      // Создание плагина
      function _Create(pOwner: IRecorder): boolean; stdcall;
      // Конфигурирование
      function Config: boolean; stdcall;
      // Вызов окна настройки
      function Edit: boolean; stdcall;
      // Запуск - активизация работы plug-in`a
      function Execute: boolean; stdcall;
      // Приостановка работы
      function Suspend: boolean; stdcall;
      // Возобновление работы
      function Resume: boolean; stdcall;
      // Уведомление о внешних событиях
      function Notify(const dwCommand: DWORD;
                   const dwData: DWORD): boolean; stdcall;
      // Получение имени
      function GetName: LPCSTR; stdcall;
      // Получить свойство
      function GetProperty(const dwPropertyID: DWORD;
                        var Value: OleVariant): boolean; stdcall;
      // Задать свойство
      function SetProperty(const dwPropertyID: DWORD;
                        {const} Value: OleVariant): boolean; stdcall;

      // Узнать можно ли завершить работу плагина
      function CanClose: boolean; stdcall;
      // Завершить работу плагина
      function Close:  boolean; stdcall;
   protected
      {Ссылка на интерфейс для управления Recorder`ом}
      FIRecorder: IRecorder;

      FDataTag:   IBlockAccess;
      FBlockCount: integer;
      FDataBuffer: array of double;

      FDataFile: TFileStream;

      procedure RefreshTagReference;

      //Метод подготовки к изменению конфигурации
      function  BeginConfigure: boolean;
      //Метод подготовки после того, как конфигурация была изменена
      procedure EndConfigure;

      procedure PrepareToStart;
      procedure CompleteAfterStop;

      procedure UpdateProcess;
      procedure DataProcess( var Data; const Size: integer );
   end;


var
  {В библиотеке создается всего один глобальный экземпляр plug-in`a, и }
  {соответсвенно есть переменная ссылка на интерфейс этого plug-in`a.}
  {Экземпляр plug-in`a создается по загрузке библиотеки, и должен удаляться,}
  {когда счетчик ссылок станет равен нулю, это должно происходить перед}
  {выгрузкой библиотеки (если подсчет ссылок организоан правильно).    }
  GPluginInstance: IRecorderPlugin = nil;

type
```

```
   {Тип для хранения информации о plug-in`e}
   {Этот тип удобнее использовать в Delphi, чем PLUGININFO}
  TInternalPluginInfo = record
    Name:           string; //наименование plug-in`a
    Dsc:            string;//строка описания
    Vendor:  string;//строка наименования фирмы
    Version: integer;//номер версии
    SubVertion:     integer;//номер подверсии
  end;

const
   {Глобальная переменная для храения описания plug-in`a.}
  GPluginInfo: TInternalPluginInfo = (
    Name:           'Delphi Тест';
    Dsc:            'Тестирование';
    Vendor:  'ООО НПП Мера';
    Version: 0;
    SubVertion:     1;
  );

implementation
uses
  SysUtils, Forms;

{Конструктор} {Создание формы тестового plug-in`a}
constructor TTestPlugin.Create;
begin
  FDataFile:= nil;
  FDataTag:= nil;
  FBlockCount:= 0;
end;
{Деструктор} {Удаление формы тестового plug-in`a}
destructor TTestPlugin.Destroy;
begin
  inherited Destroy;
end;

//IRecorderPlugin
// Создание плагина
function TTestPlugin._Create(pOwner: IRecorder): boolean;
begin
  FIRecorder:= pOwner; // сохранение полученной ссылки на интерфейс Recorder`a
  Result:= true;
end;

// Конфигурирование
function TTestPlugin.Config: boolean;
begin
  Result:= true;
end;
// Вызов окна настройки
function TTestPlugin.Edit: boolean;
begin
  Result:= true;
end;
// Запуск - активизация работы plug-in`a
function TTestPlugin.Execute: boolean;
begin
  Result:= true;
end;
// Приостановка работы
function TTestPlugin.Suspend: boolean;
begin
  Result:= true;
end;
// Возобновление работы
function TTestPlugin.Resume: boolean;
begin
  Result:= true;
end;
// Уведомление о внешних событиях
```

```
function TTestPlugin.Notify(const dwCommand: DWORD; const dwData: DWORD): boolean;
begin
  case dwCommand of
    PN_ENTERRCCONFIG: begin // перешел в режим настройки
      BeginConfigure;  //обработать начало изменения настройки
      Result:= true;
    end;
    PN_LEAVERCCONFIG: begin // вышел из режима настройки
      EndConfigure;  //обработать завершение изменения настройки
      Result:= true;
    end;
    PN_UPDATEDATA: begin //команда обновления данных в тегах
      UpdateProcess; //обработать полученные (новые) данные
      Result:= true;
    end;
    PN_BEFORE_RCSTART: begin //команда подготовки к запуску измерения
      PrepareToStart; //выполнить подготовку к измерению
      Result:= true;
    end;
    PN_RCSTOP: begin     //команда оповещающая о завершении измерения
      CompleteAfterStop; //обработать - измерение завершено
      Result:= true;
    end;
    else
      Result:= false; //прочие сообщения не обрабатываем
  end;
end;

// Получение имени
function TTestPlugin.GetName: LPCSTR;
begin
   Result:= LPCSTR(GPluginInfo.Name); {Из глобальной структуры описания plug-in`a}
end;

// Получить свойство
function TTestPlugin.GetProperty(const dwPropertyID: DWORD;
                  var Value: OleVariant): boolean;
begin
  case dwPropertyID of
     PLGPROP_INFOSTRING: begin    {Получить строку описания plug-in`a}
        Value := GPluginInfo.Dsc; {Из глобальной структуры описания plug-in`a}
        Result:= true;
     end;
     else
        Result:= false;
  end;
end;
// Задать свойство
function TTestPlugin.SetProperty(const dwPropertyID: DWORD;
                  {const} Value: OleVariant): boolean;
begin
  Result:= true;
end;

// Узнать можно ли завершить работу плагина
function TTestPlugin.CanClose: boolean;
begin
  Result:= true;
end;
// Завершить работу плагина
function TTestPlugin.Close:  boolean;
begin
  Result:= true;
end;

//Данный plug-in использует всего один, первый в списке тег.
//Обновление ссылки на этот (первый) тег.
procedure TTestPlugin.RefreshTagReference;
var
  FTag: ITag;
begin
```

```
  //получить ссылки на интерфейс тега
  if FIRecorder.GetTagsCount > 0 then
  begin
    //использование преобразования типа обосновано в описании
    FTag:= ITag(FIRecorder.GetTagByIndex(0));
    //Получение у тега интерфейса на блочный доступ к данным
    if (FAILED(FTag.QueryInterface( IBlockAccess, FDataTag))) then
       FDataTag := nil;
  end;
end;

//Метод подготовки формы к изменению конфигурации
function  TTestPlugin.BeginConfigure: boolean;
begin
  Result:= false;
end;

//Метод подготовки формы после того, как конфигурация была изменена
procedure TTestPlugin.EndConfigure;
begin
  RefreshTagReference; //обновление ссылки на тег
end;

//Метод подготовки к запуску процесса измерения, вызывается по
//сообщению от Recorder`a. В этом методе выполняется открытие файла
//для сохранения обработанных данных, а также подготовка блока памяти
//для получения измеренных данных из тега.
procedure TTestPlugin.PrepareToStart;
var
  name : string;
  Size: integer;
begin
  try
    //Формирование имени файла, в который будут
    //записаны результирующие данные.
    //...получение рабочей директории...
    name := ExtractFileDir( Application.ExeName);
    if name <> '' then
       name := name + '\';
    //...добаление имени файла...
    name := name + FILENAMETEMLATE;
    //...создание файлового потока...
    FDataFile:= TFileStream.Create(name , fmCreate);
  except
    FDataFile := nil;
  end;

  if Assigned( FDataTag) then
  begin
    FBlockCount:= 0;
    //получение размера блока для буфера
    Size:= FDataTag.GetBlocksSize;
    //выделение буфера для чтения данных
    SetLength( FDataBuffer, Size);
  end;
end;
procedure TTestPlugin.CompleteAfterStop;
begin
  if Assigned(FDataFile) then
     FreeAndNil(FDataFile);
end;

//Получение блоков данных из буфера тега, обработка блоков данных
procedure TTestPlugin.UpdateProcess;
var
  ReadyCount: integer; //счетчик обработанных Recorder`ом блоков
  Count: integer;      //счетчик блоков в буфере тега
  Size: integer;       //размер блока
  i: integer;
begin
  if Assigned(FDataTag) then
```

```
  begin
    FDataTag.LockVector; //Блокирование вектора с данными
    try
      //получение обработанных Recorder`ом блоков
      ReadyCount:= FDataTag.GetReadyBlocksCount;
      //получения кол-ва блоков в буфере тега
      Count:= FDataTag.GetBlocksCount;
      //получение размера блока
      Size:= FDataTag.GetBlocksSize;
      //Определение наличия новых данных в буфере тега
      if (ReadyCount <> FBlockCount) and (Count > 0 )then
      begin
        //отладочная проверка размера блока
        if Size <> Length( FDataBuffer) then
            SetLength( FDataBuffer, Size);
        //в цикле чтение новых блоков и обработка их
        for i := 0 to ReadyCount - FBlockCount - 1 do
        begin
          //чтение
          if SUCCEEDED(FDataTag.GetVectorR8( pointer(FDataBuffer)^,
                                             Count - 1 - i, Size, true)) then
              //обработка
              DataProcess( pointer(FDataBuffer)^, Size);
        end;
      end;
      //запомнить кол-во обработанных plug-in`ом блоков
      FBlockCount:= ReadyCount;
    finally
      //разблокирование буфера
      FDataTag.UnlockVector
    end;
  end;
end;

procedure TTestPlugin.DataProcess( var Data; const Size: integer );
type
  DoubleArray = array[word] of double;
  PDoubleArray = ^DoubleArray;
var
  i: integer;
  PtrDoubleData: DoubleArray absolute Data;
  Summ: double;
begin
  //вычисление среднего
  Summ:= 0;
  for i:= 0 to Size - 1 do
    Summ:= PtrDoubleData[i];
  if Size <> 0 then
    Summ := Summ / Size;
  //сохранение в файл
  try
    FDataFile.Write( Summ, sizeof(Summ));
  except
  end;
end;

initialization
   {Код создания одного экземпляра объетка plug-in`a, объект создается и
   в глобальной переменной сохраняется ссылка на интерфейс объекта. Объект
   удаляется автоматически, когда глобальная ссылка освобождается, то есть
   когда библиотека plug-in`a выгружается.}
   GPluginInstance:= TTestPlugin.Create;
end.
```

A.4 Plug-in, меняющий настройку ПО «Recorder»

A.4.1 Модуль проекта библиотеки «test_ctrl.pas»

```
{-------------------------------------------------------------------------}
{ Проект (Модуль) реслизации plug-in`a для измерительного ПО Recorder }
{ Компилятор: Borland Delphi 6.0                                          }
{ НПП "ООО Мера" 2004г.                                                   }
{-------------------------------------------------------------------------}

library test_ctrl;

uses
  Windows,
  SysUtils,
  Classes,
  blaccess in '..\interfaces\blaccess.pas',
  journal in '..\interfaces\journal.pas',
  modules in '..\interfaces\modules.pas',
  plugin in '..\interfaces\plugin.pas',
  recorder in '..\interfaces\recorder.pas',
  signal in '..\interfaces\signal.pas',
  tags in '..\interfaces\tags.pas',
  transf in '..\interfaces\transf.pas',
  transformers in '..\interfaces\transformers.pas',
  waitwnd in '..\interfaces\waitwnd.pas',
  PluginClass in 'PluginClass.pas',
  TestFormUnit in 'TestFormUnit.pas' {TestForm},
  RenameFormUnit in 'RenameFormUnit.pas' {FormRename},
  ChanFormUnit in 'ChanFormUnit.pas' {FormPhChans},
  ClbShowFrameUnit in 'ClbShowFrameUnit.pas' {FrameClbShow: TFrame},
  ClbScaleFrameUnit in 'ClbScaleFrameUnit.pas' {FrameClbScale: TFrame},
  ClbLineFrameUnit in 'ClbLineFrameUnit.pas' {FrameClbLine: TFrame},
  ClbIntFrameUnit in 'ClbIntFrameUnit.pas' {FrameClbInt: TFrame},
  ClbPolFrameUnit in 'ClbPolFrameUnit.pas' {FrameClbPol: TFrame};

{$R *.res}

{-------------------------------------------------------------Экспортируемые функции}
{Ниже описаны и реализованы пять функций, которые должна экспортировать}
{библиотека plug-in`a для Recorder`a}

{Получение типа объекта, реализованного в билиотеке}
function GetPluginType: integer; cdecl;
begin
  Result:= PLUGIN_CLASS; {В данном случае, это тип объекта plug-in`a}
end;
{Функция создания экземпляра plug-in`a.}
{Функция объявлена таким образом, как будто она возвращает не интерфейс,
а указатель. Причина такого объявления обоснована в документе описания.}
function CreatePluginClass: pointer{IRecorderPlugin}; cdecl;
begin
  {Данный plug-in поддерживает создание всего одного экземпляра,
   поэтому поэтому просто возвращается ссылка на глобальный объект.}
  {Сам объект не создается здаесь! Объект создается при загрузке библиотеки.}
  Result := pointer(GPluginInstance);
end;
{Функция удаления plug-in`a.}
function DestroyPluginClass(piPlg: IRecorderPlugin): integer; cdecl;
begin
   {Так как объект один и глобальный, то он здесь не удаляется.}
   {Объект plug-in`a удаляет себя сам (как любой COM объект),
    если счетчик ссылок его интерфейсов станет равен нулю.}
   Result:= 0;
end;
{Функция получения строки описания plug-in`a}
function GetPluginDescription: LPCSTR; cdecl;
begin
   {Описание извлекается из глобальной описательной структуры}
   Result:= LPCSTR(GPluginInfo.Dsc);
end;
{Функция получения полного описания plug-in`a}
procedure GetPluginInfo(var lpPluginInfo: PLUGININFO); cdecl;
begin
   {Описание извлекается из глобальной описательной структуры}
```

```
   {Копирование строк производится именно функциями StrCopy()
    из-за того, что структура описания plug-in`a описана
    в языке C++}
  StrCopy( @lpPluginInfo.name, LPCSTR(GPluginInfo.Name));
  StrCopy( @lpPluginInfo.describe, LPCSTR(GPluginInfo.Dsc));
  StrCopy( @lpPluginInfo.vendor, LPCSTR(GPluginInfo.Vendor));
  lpPluginInfo.version:= GPluginInfo.Version;
  lpPluginInfo.subversion:= GPluginInfo.SubVertion;
end;

{--------------------Объявление экспортируемых функций----------------------}
{Объявление строковых имен экспортируемых функций}
exports GetPluginType        name 'GetPluginType';
exports CreatePluginClass    name 'CreatePluginClass';
exports DestroyPluginClass   name 'DestroyPluginClass';
exports GetPluginDescription name 'GetPluginDescription';
exports GetPluginInfo        name 'GetPluginInfo';

end.
```

A.4.2 Модуль «PluginClass.pas»

```
{-----------------------------------------------------------------------}
{ Проект (Модуль) реслизации plug-in`a для измерительного ПО Recorder }
{ Модуль класса тестового plug-in`a, описание и реализация            }
{ Компилятор: Borland Delphi 6.0                                      }
{ НПП "ООО Мера" 2004г.                                               }
{-----------------------------------------------------------------------}

{В данном модуле описан и реализован класс TTestPlugin тестового plug-in`a.}

unit PluginClass;
interface
uses
   Windows,
   recorder, tags, plugin,
   TestFormUnit;

type
   {Класс тестового plug-in`a. Наследует классу TInterfacedObject,
    при этом наследуется реализация IUnknown и механизм подсчета
    ссылок соответсвенно. Класс реализует интерфейс IRecorderPlugin -
    интерфейс plug-in`a. Для отображения данных класс использует
    специальную форму (поле FTestForm).}
   TTestPlugin = class(TInterfacedObject, IRecorderPlugin)
   public
      {Конструктор}
      constructor Create;
      {Деструктор}
      destructor Destroy; override;

      {Метод запуска и останова измерения}
      function StartMeasure: boolean;
      function StopMeasure: boolean;
   public    //IRecorderPlugin
      // Создание плагина
      function _Create(pOwner: IRecorder): boolean; stdcall;
      // Конфигурирование
      function Config: boolean; stdcall;
      // Вызов окна настройки
      function Edit: boolean; stdcall;
      // Запуск - активизация работы plug-in`a
      function Execute: boolean; stdcall;
      // Приостановка работы
      function Suspend: boolean; stdcall;
      // Возобновление работы
      function Resume: boolean; stdcall;
      // Уведомление о внешних событиях
```

```
      function Notify(const dwCommand: DWORD;
                   const dwData: DWORD): boolean; stdcall;
      // Получение имени
      function GetName: LPCSTR; stdcall;
      // Получить свойство
      function GetProperty(const dwPropertyID: DWORD;
                        var Value: OleVariant): boolean; stdcall;
      // Задать свойство
      function SetProperty(const dwPropertyID: DWORD;
                        {const} Value: OleVariant): boolean; stdcall;

      // Узнать можно ли завершить работу плагина
      function CanClose: boolean; stdcall;
      // Завершить работу плагина
      function Close:  boolean; stdcall;
   protected
      {Ссылка на форму plug-in`a. Во избежание рекурсивных ссылок
       для работы с формой используется понижающее приведение типа.
       Переменная описана как TForm, а используется везде с приведением
       типа к TTestForm}
      FTestForm: TTestForm;

   public
      {Ссылка на интерфейс для управления Recorder`ом}
      FIRecorder: IRecorder;
   end;


var
  {В библиотеке создается всего один глобальный экземпляр plug-in`a, и }
  {соответсвенно есть переменная ссылка на интерфейс этого plug-in`a.}
  {Экземпляр plug-in`a создается по загрузке библиотеки, и должен удаляться,}
  {когда счетчик ссылок станет равен нулю, это должно происходить перед}
  {выгрузкой библиотеки (если подсчет ссылок организоан правильно).    }
  GPluginInstance: IRecorderPlugin = nil;

type
   {Тип для хранения информации о plug-in`e}
   {Этот тип удобнее использовать в Delphi, чем PLUGININFO}
  TInternalPluginInfo = record
    Name:           string; //наименование plug-in`a
    Dsc:            string;//строка описания
    Vendor:  string;//строка наименования фирмы
    Version: integer;//номер версии
    SubVertion:     integer;//номер подверсии
  end;

const
   {Глобальная переменная для храения описания plug-in`a.}
  GPluginInfo: TInternalPluginInfo = (
    Name:           'Delphi Тест управления';
    Dsc:            'Тестирование';
    Vendor:  'ООО НПП Мера';
    Version: 0;
    SubVertion:     1;
  );

implementation
uses
  SysUtils;

{Конструктор} {Создание формы тестового plug-in`a}
constructor TTestPlugin.Create;
begin
  FTestForm:= TTestForm.Create( self );
end;
{Деструктор} {Удаление формы тестового plug-in`a}
destructor TTestPlugin.Destroy;
begin
  FreeAndNil( FTestForm);
end;
```

```
//IRecorderPlugin
//Инициализация объекта plug-in`a.
function TTestPlugin._Create(pOwner: IRecorder): boolean;
begin
  FIRecorder:= pOwner; // сохранение полученной ссылки на интерфейс Recorder`a
  FTestForm.Show();    // отображение формы
  Result:= true;
end;

// Конфигурирование
function TTestPlugin.Config: boolean;
begin
  Result:= true;
end;
// Вызов окна настройки
function TTestPlugin.Edit: boolean;
begin
  FTestForm.Show();    // отображение формы
  Result:= true;
end;
// Запуск - активизация работы plug-in`a
function TTestPlugin.Execute: boolean;
begin
  FTestForm.Show();    // отображение формы
  Result:= true;
end;
// Приостановка работы
function TTestPlugin.Suspend: boolean;
begin
  Result:= true;
end;
// Возобновление работы
function TTestPlugin.Resume: boolean;
begin
  Result:= true;
end;
// Уведомление о внешних событиях
function TTestPlugin.Notify(const dwCommand: DWORD; const dwData: DWORD): boolean;
begin
  case dwCommand of
    PN_ENTERRCCONFIG: begin // перешел в режим настройки
      {В то время пока производится изменение настройки,
      теги недоступны (потому, что они могут удаляться)}
      //сообщить фрме о начале изменения настройки
      Result:= true;
    end;
    PN_LEAVERCCONFIG: begin // вышел из режима настройки
      {Оповестить форму о завершении изменения конфигурации и
      передать ей новый список ссылок на интерфейсы тегов}
      Result:= true;
    end;
    PN_SHOWINFO: begin  //команда для отображения собственной формы
      FTestForm.Show();    // отображение формы
      Result:= true;
    end;
    else
      Result:= false; //прочие сообщения не обрабатываем
  end;
end;

// Получение имени
function TTestPlugin.GetName: LPCSTR;
begin
   Result:= LPCSTR(GPluginInfo.Name); {Из глобальной структуры описания plug-in`a}
end;

// Получить свойство
function TTestPlugin.GetProperty(const dwPropertyID: DWORD;
                  var Value: OleVariant): boolean;
begin
```

```
  case dwPropertyID of
     PLGPROP_INFOSTRING: begin    {Получить строку описания plug-in`a}
        Value := GPluginInfo.Dsc; {Из глобальной структуры описания plug-in`a}
        Result:= true;
     end;
     else
        Result:= false;
  end;
end;
// Задать свойство
function TTestPlugin.SetProperty(const dwPropertyID: DWORD;
                  {const} Value: OleVariant): boolean;
begin
  Result:= true;
end;

// Узнать можно ли завершить работу плагина
function TTestPlugin.CanClose: boolean;
begin
  Result:= true;
end;
// Завершить работу плагина
function TTestPlugin.Close:  boolean;
begin
  Result:= true;
end;

{Методы запуска и останова измерения}
function TTestPlugin.StartMeasure: boolean;
begin
  {Для запуска необходимо передать сообщение Recorder`y}
  Result:= FIRecorder.Notify( RCN_VIEW, 0);
end;
function TTestPlugin.StopMeasure: boolean;
begin
  {Для останова необходимо передать сообщение Recorder`y}
  Result:= FIRecorder.Notify( RCN_STOP, 0);
end;

initialization
   {Код создания одного экземпляра объетка plug-in`a, объект создается и
   в глобальной переменной сохраняется ссылка на интерфейс объекта. Объект
   удаляется автоматически, когда глобальная ссылка освобождается, то есть
   когда библиотека plug-in`a выгружается.}
   GPluginInstance:= TTestPlugin.Create;
end.
```

A.4.3 Модуль «TestFormUnit.pas»

```
{---------------------------------------------------------------------}
{ Проект (Модуль) реслизации plug-in`a для измерительного ПО Recorder }
{ Модуль описания класса формы тестового plug-in`a                    }
{ Компилятор: Borland Delphi 6.0                                      }
{ НПП "ООО Мера" 2004г.                                               }
{---------------------------------------------------------------------}

unit TestFormUnit;

interface

uses
  Windows, Messages, SysUtils, Variants,
  Classes, Graphics, Controls, Forms,
  Dialogs, ComCtrls, ExtCtrls, Buttons,
  ImgList, StdCtrls,
  recorder, tags, blaccess, Grids, ClbShowFrameUnit;

type
```

```
  {Тип динамического массива целых, используется для
  работы с идентификаторами физических каналов}
  DynCardinals = array of Cardinal;

  {Тип динамического массива строк}
  DynStrings =   array of string;

type
  //Класс формы тестового Plug-in`a. Основные выполняемые действия:
  //отображение списака имен и списка строк описаний тегов, отображение
  //и периодическое обнволение измеряемых значений для тегов.
  //Обновление измеряемых значений производится по событиям таймера DataUpdateTimer.
  //Форма хранит список тегов.
  TTestForm = class(TForm)
    DataUpdateTimer: TTimer;
    ControlPanel: TPanel;
    Bevel2: TBevel;
    Bevel1: TBevel;
    LogoImage: TImage;
    TimeLabel: TLabel;
    SBtnStop: TSpeedButton;
    SBtnView: TSpeedButton;
    Label1: TLabel;
    PageControl: TPageControl;
    TSRecorder: TTabSheet;
    TSGroups: TTabSheet;
    TSTags: TTabSheet;
    TSPChans: TTabSheet;
    TSClbs: TTabSheet;
    SGGroups: TStringGrid;
    SGTags: TStringGrid;
    SGPhChans: TStringGrid;
    ComboBoxTags: TComboBox;
    Panel1: TPanel;
    Splitter1: TSplitter;
    Panel2: TPanel;
    SGMrInfo: TStringGrid;
    ListBoxRecStates: TListBox;
    EditUpdateTime: TEdit;
    EditVectorTime: TEdit;
    BtnMrGetInfo: TButton;
    BtnMrGetState: TButton;
    BtnMrSetInfo: TButton;
    BtnGetGroups: TButton;
    BtnGrpAdd: TButton;
    BtnGrpDel: TButton;
    BtnGrpRename: TButton;
    BtnGetTags: TButton;
    BtnTagAdd: TButton;
    BtnTagDel: TButton;
    BtnTagRename: TButton;
    BtnGetPhChans: TButton;
    Bevel3: TBevel;
    Label2: TLabel;
    Label3: TLabel;
    SBtnImportConfig: TSpeedButton;
    SBtnExportConfig: TSpeedButton;
    Label4: TLabel;
    TabControlClb: TTabControl;
    FrameClbShow: TFrameClbShow;
    BtnTagAddVirt: TButton;
    BtnSetTestFunc: TButton;
    procedure TimeTimerTimer(Sender: TObject);
    procedure SBtnStopClick(Sender: TObject);
    procedure SBtnViewClick(Sender: TObject);
    procedure FormCreate(Sender: TObject);
    procedure BtnMrGetInfoClick(Sender: TObject);
    procedure BtnMrGetStateClick(Sender: TObject);
    procedure BtnGetGroupsClick(Sender: TObject);
    procedure TSGroupsShow(Sender: TObject);
    procedure TSRecorderShow(Sender: TObject);
```

```
  procedure BtnMrSetInfoClick(Sender: TObject);
  procedure SGGroupsSelectCell(Sender: TObject; ACol, ARow: Integer;
    var CanSelect: Boolean);
  procedure BtnGrpAddClick(Sender: TObject);
  procedure BtnGrpDelClick(Sender: TObject);
  procedure BtnGrpRenameClick(Sender: TObject);
  procedure TSTagsShow(Sender: TObject);
  procedure BtnGetTagsClick(Sender: TObject);
  procedure SGTagsSelectCell(Sender: TObject; ACol, ARow: Integer;
    var CanSelect: Boolean);
  procedure BtnTagAddClick(Sender: TObject);
  procedure BtnTagRenameClick(Sender: TObject);
  procedure BtnTagDelClick(Sender: TObject);
  procedure BtnGetPhChansClick(Sender: TObject);
  procedure TSPChansShow(Sender: TObject);
  procedure TSClbsShow(Sender: TObject);
  procedure ComboBoxTagsChange(Sender: TObject);
  procedure TabControlClbChange(Sender: TObject);
  procedure ComboBoxTagsDropDown(Sender: TObject);
  procedure BtnTagAddVirtClick(Sender: TObject);
  procedure SBtnImportConfigClick(Sender: TObject);
  procedure SBtnExportConfigClick(Sender: TObject);
  procedure BtnSetTestFuncClick(Sender: TObject);
public
  //Конструктор формы переопределен для того, чтобы форму
  //можно было инициализировать ссылкой на объект plug-in`a
  constructor Create( const Plugin: TObject); reintroduce;
private
  {ссылка на объект plug-in`a}
  {ссылка типа TObject, для того, чтобы избежать рекурсивных использований модулей}
  FPlugin: TObject;

  {Сервисный метод, который должен быть вызван для перехода в режим настройки}
  function BeginConfigure: boolean;
  {Сервисный метод, который должен быть вызван для выхода из режим настройки}
  procedure EndConfigure;

  //Методы  - Страница управления группами
  //Добавление группы в таблицу (на форме) отображения параметров групп
  //Праматметром является ссылка на интерфейс группы, при помощи этогоинтерфейса
  //метод получает информацию о группе и добавляет строку в табблицу.
  function AddGroupToList( pgrp: ITagsGroup): boolean;
  //проверить корректность выбранной строки в таблице
  //групп (на форме), автоматическое разрешение или запрещение
  //кнопок удаления и переименования объекта группы
  procedure CheckGrpSelection( const row: integer);

  //Методы - Страница управления тегами
  //Добавление тега в таблицу (на форме) отображения параметров тега
  function AddTagToList( ptag: ITag): boolean;
  //проверить корректность выбранной строки в таблице
  //тегов (на форме), автоматическое разрешение или запрещение
  //кнопок удаления и переименования объекта тега
  procedure CheckTagSelection( const row: integer);

  //Функция преобразования кода типа данных Variant в строковое представление
  function VarTypeToString(const v: Variant): string;
  //Функция преобразования кода типа связи тега с физическим каналом в строку
  function LinkStateToString(const v: LINKSTATE): string;
  //Функция преобразования кода типа доступа к тегу в строку
  function AccessRightToString(const v: integer): string;

  //Методы - Страница просмотра списка физических каналов.
  //Получение трех массивов описывающих список физических адресов.
  //Данный метод производит получение списка "свободных" физических каналов,
  //для которых еще не созданы теги, и заполняет три динамических массива:
  //массив идентификаторов каналов, массив строк адресов, массив строк имен
  //измерительных устройств, которым принадлежат физические каналы.
  function GetPhChansDsc( var ChanIDs: DynCardinals; var Addresses: DynStrings;
                          var Devices: DynStrings): boolean;
```

```
    //Методы - Страница просмотра ГФ/КФ.
    //Метод по имени выбранного на форме тега производит получение
    //ссылки на тег и при помощи фрейма отображения ГФ/КФ производит
    //вывод параметров ГФ/КФ.
    function ShowCurrentCalibration: boolean;

    //Удаление строки из таблицы со сдвигом строк вверх.
    //Общий метод применимый к любому объекту  класса TStringGrid.
    procedure DelRowWithUpShift( SG: TStringGrid; const Row: integer);
  end;

implementation
{$R *.dfm}
uses
   PluginClass,    {Используется для доступа к классу plug-in`a}
   RenameFormUnit, {Спец. форма изменения имени объекта}
   ChanFormUnit;   {Спец. форма отображения и выбора имени канала из списка}

//Конструктор формы переопределен для того, чтобы форму
//можно было инициализировать ссылкой на объект plug-in`a
constructor TTestForm.Create( const Plugin: TObject);
begin
  inherited Create( nil);
  FPlugin:= Plugin; //сохранить ссылку на объект plug-in`a
end;

//Обработчик событий таймера обновления данных
procedure TTestForm.TimeTimerTimer(Sender: TObject);
begin
  TimeLabel.Caption := TimeToStr( Now);
end;

procedure TTestForm.SBtnStopClick(Sender: TObject);
begin
  TTestPlugin(FPlugin).StopMeasure;
end;

procedure TTestForm.SBtnViewClick(Sender: TObject);
begin
  TTestPlugin(FPlugin).StartMeasure;
end;

//Импортирование натсройки ПО "Recorder"
procedure TTestForm.SBtnImportConfigClick(Sender: TObject);
var
  pmr: IRecorder;
begin
  //Получение ссылки на объект Recorder`a
  pmr:= TTestPlugin( FPlugin).FIRecorder;
  //выполнение и проверка рез-та импорта
  if SUCCEEDED( pmr.ImportSettings( 'c:\temp\', 0)) then
    Application.MessageBox( 'Импорт файлов настройки завершен успешно!',
                            'Внимание!', MB_OK + MB_ICONINFORMATION)
  else
    Application.MessageBox( 'Ошибка импорта файлов настройки!',
                            'Ошибка!', MB_OK + MB_ICONERROR)
end;

//Экспортирование натсройки ПО "Recorder"
procedure TTestForm.SBtnExportConfigClick(Sender: TObject);
var
  pmr: IRecorder;
begin
  //Получение ссылки на объект Recorder`a
  pmr:= TTestPlugin( FPlugin).FIRecorder;
  //выполнение и проверка рез-та экспорта
  if SUCCEEDED( pmr.ExportSettings( 'c:\temp\', 0)) then
    Application.MessageBox( 'Экспорт файлов настройки завершен успешно!',
                            'Внимание!', MB_OK + MB_ICONINFORMATION)
  else
    Application.MessageBox( 'Ошибка экспорта файлов настройки!',
```

```
                              'Ошибка!', MB_OK + MB_ICONERROR)
end;

procedure TTestForm.FormCreate(Sender: TObject);
begin
  LogoImage.Hint:= 'ООО "НПП Мера"'+#13#10+'т. 516-89-16'+#13#10+'web:
www.nppmera.ru';
  //Активная страница на запуске программы
  PageControl.ActivePage := TSRecorder;
end;

{Сервисный метод, который должен быть вызван для перехода в режим настройки}
function TTestForm.BeginConfigure: boolean;
begin
  //Нельзя менять настройку если режим настройки не включен
  Result:=
    SUCCEEDED(TTestPlugin(FPlugin).FIRecorder.EnterConfigMode( TTestPlugin(FPlugin),
0));
end;
{Сервисный метод, который должен быть вызван для выхода из режим настройки}
procedure TTestForm.EndConfigure;
begin
  //Изменения внесенные в настройку ПО Recorder вступят
  //в силу только после выхода из режима настройки
  TTestPlugin(FPlugin).FIRecorder.LeaveConfigMode( TTestPlugin(FPlugin), 0)
end;

//Удаление строки из таблицы со сдвигом строк вверх
//Общий метод применимый к любому объекту  класса TStringGrid.
procedure TTestForm.DelRowWithUpShift( SG: TStringGrid; const Row: integer);
var
  i, j: integer;
begin
  //Предпологается, что последняя строка в TStringGrid всегда пуста и
  //не используется для отображения, из этого вытекают условия для
  //проверки корректности индекса удаляемой строки.
  if (SG.RowCount > 2) and (Row > 0) and (Row < (SG.RowCount - 1)) then
  begin
    //от удаляемой строки в цикле...
    for i:= Row to SG.RowCount - 2 do
      //...каждая колонка...
      for j := 0 to SG.ColCount - 1 do
        //...переписываются...
        SG.Cells[j, i] := SG.Cells[j, i + 1];
    //уменьшается общее кол-во строк
    SG.RowCount := SG.RowCount - 1;
  end;
end;

{-----------------------------------------------------------------------------------------}
{          Страница промотра и изменения (общих) свойств объекта Recorder`a          }
{-----------------------------------------------------------------------------------------}
//При отображении страницы просмотра свойств Recorder`a
//необходимо страницу инициализировать, очистить.
procedure TTestForm.TSRecorderShow(Sender: TObject);
begin
  //Очистка списка параметров
  SGMrInfo.RowCount:= 2;
  SGMrInfo.Cells[0, 0] := 'Наименование';
  SGMrInfo.Cells[1, 0] := 'Значение';
  SGMrInfo.Rows[1].Clear;

  //Очистка списка флагов состояний
  ListBoxRecStates.Clear;

  //Временные параметры
  EditUpdateTime.Text := '?';
  EditVectorTime.Text := '?';
end;

//Сбор информации о Recorder`e и отображение её в табличном виде
```

```
procedure TTestForm.BtnMrGetInfoClick(Sender: TObject);
var
  pmr: IRecorder;
  v: OleVariant;
  len: integer;
  tmp: string;
begin
  //Установка размера таблицы с параметрами ПО Recorder
  SGMrInfo.RowCount:= 9; //..8 строк и одна на заголовок

  //получение ссылки на объект Recorder
  pmr:= TTestPlugin( FPlugin).FIRecorder;

  //получение и отображение кода состояния
  SGMrInfo.Cells[0, 1] := 'Код состояния';
  SGMrInfo.Cells[1, 1] := '0x' + IntToHex( pmr.GetState(RS_FULLMASK), 8);

  SGMrInfo.Cells[0, 2] := 'Базовый каталог';
  SGMrInfo.Cells[1, 2] := pmr.GetRcBasePath;

  SGMrInfo.Cells[0, 3] := 'Каталог с данными';
  SGMrInfo.Cells[1, 3] := pmr.GetSignalFolderName;

  SGMrInfo.Cells[0, 5] := 'Файл настройки 1';
  SGMrInfo.Cells[1, 5] := pmr.GetProjectName;

  try
    SGMrInfo.Cells[0, 4] := 'Каталог с файлами ГФ/КФ';
    //получение имени каталога с файлами ГФ/КФ осуществояется
    //через получение свойства с кодом RCPROP_CALIBRDBNAME
    if pmr.GetProperty( RCPROP_CALIBRDBNAME, v) = S_OK then
      SGMrInfo.Cells[1, 4] := v
    else
      SGMrInfo.Cells[1, 4] := '';

    SGMrInfo.Cells[0, 6] := 'Файл настройки 2';
    //Получение имени файла настройки производится через переменные окружения
    //ПО Recorder, имя переменной "\Recorder\System\CfgName".
    //Проверка корректности вызова и получение размера строки имени
    if pmr.GetEnvironmentString( '\Recorder\System\CfgName', nil, len) then
    begin
      //выделение буфера строки под чтение имени файла настройки
      SetLEngth(tmp, len);
      //чтение строки имени
      pmr.GetEnvironmentString( '\Recorder\System\CfgName', PChar(tmp), len);
      SGMrInfo.Cells[1, 6] := tmp;
    end
    else
      SGMrInfo.Cells[1, 6] := '';


    SGMrInfo.Cells[0, 7] := 'Период обновления';
    if pmr.GetProperty( RCPROP_REFRESHPERIOD, v) = S_OK then
    begin
      SGMrInfo.Cells[1, 7] := v;
      EditUpdateTime.Text := v;
    end
    else
    begin
      SGMrInfo.Cells[1, 7] := '';
      EditUpdateTime.Text := '0.3';
    end;

    SGMrInfo.Cells[0, 8] := 'Время отображения';
    if pmr.GetProperty( RCPROP_VIEWTIME, v) = S_OK then
    begin
      SGMrInfo.Cells[1, 8] := v;
      EditVectorTime.Text := v;
    end
    else
    begin
```

```
      SGMrInfo.Cells[1, 8] := '';
      EditVectorTime.Text := '1';
    end;
  except
  end;
end;

//Метод получения, расшифровки и отображения флагов состояния
procedure TTestForm.BtnMrGetStateClick(Sender: TObject);
//Для преобразования состояния в строку формируется таблица соответствия
//битов состояния и строк описывающих данное состояние.
//Структура для описания одной строки таблицы соответствия...
type
  TStateString = record
    Mask:  longword; //маска на слово состояния
    State: longword; //состояние и ...
    Text:  string;   //соответствующая ему строка
  end;

const
  //Объявления констант упрщающих представление кода программы
  RS_ALLRESET = RS_NEEDLINKSREFRESH + RS_NEEDSOFTWARERESET;
  RS_NHWR     = RS_NEEDHARDWARERESET;

  StateStringLen = 14;
type
  TStateStringTable = array[0..StateStringLen-1] of TStateString;

const
  //Массив соответствия...
  stlinks: TStateStringTable = (
    {Маска--------------------Состояние----------------Описание-------------------}
    ( Mask: RS_STOP;          State: RS_STOP;          Text: 'Остановлен'),
    ( Mask: RS_VIEW;          State: RS_VIEW;          Text: 'Просмотр'),
    ( Mask: RS_REC;           State: RS_REC;           Text: 'Запись'),
    ( Mask: RS_PLAYING;       State: RS_PLAYING;       Text: 'Проигрывание'),
    ( Mask: RS_HARDWAREFAULT; State: RS_HARDWAREFAULT; Text: 'Аппаратная ошибка'),
    ( Mask: RS_INITFAULT;     State: RS_INITFAULT;     Text: 'Программная ошибка'),
    ( Mask: RS_NHWR;          State: RS_NHWR;          Text: 'Необходим перезапуск'),
    ( Mask: RS_ALLRESET;      State: RS_ALLRESET;      Text: 'Необходим перезапуск'),
    ( Mask: RS_CONFIGCHANGED; State: RS_CONFIGCHANGED; Text: 'Конфигурация сменена'),
    ( Mask: RS_CONFIGMODE; State: RS_CONFIGMODE;    Text: 'Режим настройки включен'),
    ( Mask: RS_CONFIGMODE;    State: 0;             Text: 'Режим настройки выключен'),
    ( Mask: RS_PACKETLOST;    State: RS_PACKETLOST;    Text: 'Были потери ранее'),
    ( Mask: RS_RECEIVEERROR;  State: RS_RECEIVEERROR;  Text: 'Имеются потери'),
    ( Mask: RS_TERMINATION;   State: RS_TERMINATION;   Text: 'Завершение работы')
  );
var
  State: integer;
  i: integer;
begin
  //Очистка списка отображающего перечень флагов состояния
  ListBoxRecStates.Clear;
  //получение кода состояния
  State:= TTestPlugin( FPlugin).FIRecorder.GetState(RS_FULLMASK);
  //в цикле проверка филагов и просмотр таблицы соответствия,
  //добавление строк описания флагов состояний в общий список
  for i := 0 to StateStringLen - 1 do
  begin
    if (State and stlinks[i].Mask) = stlinks[i].State then
          ListBoxRecStates.Items.Add( stlinks[i].Text);
  end;
end;

procedure TTestForm.BtnMrSetInfoClick(Sender: TObject);
begin
//
end;

{-------------------------------------------------------------------------------}
{                      Страница промотра и изменения списка групп               }
```

```
{-------------------------------------------------------------------------------}
//Проверить корректность выбранной строки в таблице групп (на форме), автоматическое
//разрешение или запрещение кнопок удаления и переименования объекта группы.
procedure TTestForm.CheckGrpSelection( const row: integer);
var tmp: boolean;
begin
  //Группа выбрана корректно, если строка не нулевая (не заголовок)
  //и не последняя, (последняя является служебной)
  tmp:= (row > 0) and ( row < (SGGroups.RowCount - 1));
  BtnGrpDel.Enabled := tmp;
  BtnGrpRename.Enabled:= tmp;
end;

//Добавление группы в таблицу (на форме) отображения параметров групп.
//Праматметром является ссылка на интерфейс группы, при помощи этогоинтерфейса
//метод получает информацию о группе и добавляет строку в табблицу.
function TTestForm.AddGroupToList( pgrp: ITagsGroup): boolean;
var
  buffer: array [byte] of char;
  Index: integer;
  v: OleVariant;
begin
  //получить индекс последней строки в таблице,
  //это будет инлекс новой добавленной строки
  Index:= SGGroups.RowCount - 1;
  try
    //увеличение кол-во строк таблицы
    SGGroups.RowCount := SGGroups.RowCount + 1;

    //получение имени группы
    if FAILED(pgrp.GetName( @buffer)) then raise Exception.Create('');
    SGGroups.Cells[0, Index]:= PChar(@buffer);

    //получение строки описания группы
    if FAILED(pgrp.GetProperty( TGRPROP_DESCRIBE, v)) then raise Exception.Create('');
    SGGroups.Cells[1, Index]:= v;

    //получение признака активности группы
    if pgrp.GetRecEnabled() then
      SGGroups.Cells[2, Index]:= 'Активна'
    else
      SGGroups.Cells[2, Index]:= 'Не активна';

    //получение признака необходимости сохранения измеренных
    //данных всех тегов группы в отдельный (для группы) каталог
    if FAILED(pgrp.GetProperty( TGRPROP_WRITETOPERSONALFRAME, v))
       then raise Exception.Create('');
    if v then
      SGGroups.Cells[3, Index]:= 'Да'
    else
      SGGroups.Cells[3, Index]:= 'Нет';

    //получение строки имени каталога, в который сохраняются измеренные данные
    if FAILED(pgrp.GetProperty( TGRPROP_PERSONALFRAMENAME, v))
       then raise Exception.Create('');
    SGGroups.Cells[4, Index]:= v;

    Result:= true;

  //если возникла ошибка при получении значений свойств группы
  except
    //..отмена добавления строки в таблицу
    SGGroups.Rows[Index].Clear;
    SGGroups.RowCount := SGGroups.RowCount - 1;

    Result:= false;
  end;
end;

//Перед отображением страницы групп,...
procedure TTestForm.TSGroupsShow(Sender: TObject);
```

```
begin
  //Очистка таблицы групп
  SGGroups.RowCount:= 2;
  SGGroups.Cells[0, 0]:= 'Наименование';
  SGGroups.Cells[1, 0]:= 'Описание';
  SGGroups.Cells[2, 0]:= 'Активность';
  SGGroups.Cells[3, 0]:= 'Отдельный каталог';
  SGGroups.Cells[4, 0]:= 'Каталог с данными';
  SGGroups.Rows[1].Clear;

  //В данный момент выбрана первая строка и кнопки
  //удаление/переименования должны быть запрещены.
  CheckGrpSelection( SGGroups.Row);
end;

//Обработка нажатия кнопки получения информации о группах...
//Метод производит получение списка групп ПО Recorder, для
//каждой группы параметры выводятся в таблицу. Таблица реализо
//вана компонентой TStringGrid.
procedure TTestForm.BtnGetGroupsClick(Sender: TObject);
var
  pmr: IRecorder;
  pgrp: ITagsGroup;
  count: integer;
  i: integer;
begin
  //Предварительная очистка таблицы
  SGGroups.RowCount := 2;
  SGGroups.Rows[1].Clear;

  //Получение ссылки на объект Recorder`a (у plug-in`a)
  pmr := TTestPlugin(FPlugin).FIRecorder;

  //получить кол-во групп
  count:= pmr.GetGroupsCount();
  //в цикле по всем группам...
  for i:= 0 to count - 1 do
  begin
    //получить интерфейс группы, по номеру
    pointer(pgrp) := pmr.GetGroupByIndex( i);
    //добавить в таблицу и отобразить...
    AddGroupToList( pgrp);
    pgrp:= nil;
  end;
  //проверить выбранную строку...
  CheckGrpSelection( SGGroups.Row);
end;

//Обработка изменения выбранной строки в таблице
procedure TTestForm.SGGroupsSelectCell(Sender: TObject; ACol,
  ARow: Integer; var CanSelect: Boolean);
begin
  //проверить строку таблицы на корректность
  CheckGrpSelection( ARow);
end;

//Обработка нажатия кнопки добавления (создания новой) группы
procedure TTestForm.BtnGrpAddClick(Sender: TObject);
var
  GrpName: string;  //имя новой группы
  prec: IRecorder;
  pgrp: ITagsGroup; //ссылка на интерфейс новой группы
begin
  //создание формы для ввода имени новой группы
  FormRename := TFormRename.Create( self);
  try
    GrpName := 'Новая группа';
    //Активизация формы редактирования имени новой группы
    if FormRename.Execute( GrpName) then  //
    begin
      //если успешно удалось перейти к режиму изменения настройки, то...
```

```
      if BeginConfigure then
      begin
        try
          //получение ссылки на объект Recorder
          prec:= TTestPlugin(FPlugin).FIRecorder;
          //добавление новой группы с именем GrpName
          pointer(pgrp):= prec.CreateTagsGroup( PChar( GrpName));
          //если удалось создать группу и ссылка на новую группу не нулевая, то...
          if Assigned( pgrp) then
             //добавить группу в таблицу
             AddGroupToList( pgrp)
          else
             Application.MessageBox( 'Не удалось сздать группу!',
                                      'Ошибка!', MB_OK + MB_ICONERROR);
        finally
          //проццесс изменения настройки завершен
          EndConfigure;
        end;
      end
      else
         Application.MessageBox( 'Нет возможности перейти в режим настройки!',
                                 'Ошибка!', MB_OK + MB_ICONERROR);
    end;
  finally
    //удаление объекта формы редактора имени группы
    FreeAndNil (FormRename);
  end;
end;

//Обработка нажатия кнопки удаления группы.
//Должна быть удалена та группа, которая выбрана в таблице.
procedure TTestForm.BtnGrpDelClick(Sender: TObject);
var
  prec: IRecorder;
  pgrp: ITagsGroup;
begin
  //Проерка корректности выбора строки таблицы групп...
  //... не нулевая (не заголовок) и не последняя (не служебная)...
  if (SGGroups.Row > 0) and (SGGroups.Row < (SGGroups.RowCount - 1)) then
  begin
    //переход к режиму изменения настройки и проверка корректности перехода
    if BeginConfigure then
    begin
      try
        //получение ссылки на объект Recorder
        prec:= TTestPlugin(FPlugin).FIRecorder;
        //получение ссылки на объект группы по имени группы,
        //имя группы извлекается из таблицы групп
        pointer(pgrp):= prec.GetGroupByName( PChar(SGGroups.Cells[0, SGGroups.Row]));
        //если группа была правильно найдена по имени, ссылка на объект определена
        if Assigned( pgrp) then
        begin
          //удаление и проверка првильности удалнимя
          if SUCCEEDED( prec.DeleteTagsGroup( pgrp)) then
          begin
            pgrp:= nil;
            //удаление строки из таблицы
            DelRowWithUpShift( SGGroups, SGGroups.Row);
          end
          else
            Application.MessageBox( 'Не удалось удалить группу!',
                                     'Ошибка!', MB_OK + MB_ICONERROR);
        end;
      finally
        //в завершении необходимо выйти из режима изменения настройки
        EndConfigure;
      end;
    end
    else
       Application.MessageBox( 'Нет возможности перейти в режим настройки!',
                               'Ошибка!', MB_OK + MB_ICONERROR);
```

```
  end
  else
    Application.MessageBox( 'Внутренняя ошибка, неправильно выбрана группа!',
                            'Ошибка!', MB_OK + MB_ICONERROR);

  //проверить корректность выбранной строки в таблице
  CheckGrpSelection( SGGroups.Row);
end;

//Обработка нажатия кнопки переименования группы.
//Используется группа выбранная в таблитце.
procedure TTestForm.BtnGrpRenameClick(Sender: TObject);
var
  GrpName: string; // строка имени группы
  prec: IRecorder;
  pgrp: ITagsGroup;
begin
  //Проерка корректности выбора строки таблицы групп...
  //... не нулевая (не заголовок) и не последняя (не служебная)...
  if (SGGroups.Row > 0) and (SGGroups.Row < (SGGroups.RowCount - 1)) then
  begin
    //Создание формы для редактирования имени группы
    FormRename := TFormRename.Create( self);
    try
      //получение имени группы из таблице (по номеру выбранной строки)
      GrpName := SGGroups.Cells[0, SGGroups.Row];

      //Активизация формы редактирования имени
      if FormRename.Execute( GrpName) then  //
      begin
        //переход к режиму настройки и проверка корректности перехода...
        if BeginConfigure then
        begin
          try
            //получение ссылки на объект Recorder`a
            prec:= TTestPlugin(FPlugin).FIRecorder;

            //получение ссылки на группу по ("строму") имени группы
            pointer(pgrp):= prec.GetGroupByName( PChar(SGGroups.Cells[0,
SGGroups.Row]));

            //группа найдена и корректно получена ссылка на объект группы
            if Assigned( pgrp) then
            begin
              //установка "нового" имени и проверка корректности установки
              if SUCCEEDED( pgrp.SetName( PChar(GrpName))) then
              begin
                pgrp:= nil;
                //Изменение имени группы в таблице
                SGGroups.Cells[0, SGGroups.Row] := GrpName;
              end
              //не удалось изменить имя группы
              else
                Application.MessageBox( 'Не удалось переименовать группу!',
                                        'Ошибка!', MB_OK + MB_ICONERROR);
            end;
          finally
            //в завершение необходимо выйти из режима изменения настройки
            EndConfigure;
          end;
        end
        else
           Application.MessageBox( 'Нет возможности перейти в режим настройки!',
                                   'Ошибка!', MB_OK + MB_ICONERROR);
      end;
    finally
      //удаление не нужной более формы редакторования имени
      FreeAndNil (FormRename);
    end;
  end
  else
```

```
    Application.MessageBox( 'Внутренняя ошибка, неправильно выбрана группа!',
                            'Ошибка!', MB_OK + MB_ICONERROR);
end;

{-------------------------------------------------------------------------------}
{                      Страница промотра и изменения списка тегов               }
{-------------------------------------------------------------------------------}

//Добавление тега в таблицу (на форме) отображения параметров тега
function TTestForm.AddTagToList( ptag: ITag): boolean;
var
  Index: integer;
  v: OleVariant;                {временная переменная для получения свойств тега}
  pgrp: ITagsGroup;
  buffer: array [byte] of char; {буфер для получения строк}
begin
  //Индекс новой добавляемой строки,
  Index:= SGTags.RowCount - 1;
  try
    //увеличить кол-во строк таблицы
    SGTags.RowCount := SGTags.RowCount + 1;

    //установить в таблицу имя тега
    SGTags.Cells[0, Index]:= ptag.GetName;

    //получить строку описания тега, через свойство...
    if not ptag.GetProperty( TAGPROP_DESCRIBE, v) then raise Exception.Create('');
    SGTags.Cells[1, Index]:= v;

    //получить у тега ссылку на группу
    if FAILED(ptag.GetGroup( pgrp)) and Assigned(pgrp) then raise Excep-
tion.Create('');
    //получить у группы имя и установить в таблицу
    {необходимо быть уверенным, что строка имени группы не больше буфера}
    if FAILED(pgrp.GetName( @buffer)) then raise Exception.Create('');
    SGTags.Cells[2, Index]:= PChar( @buffer);

    //получить строку размерности
    if not ptag.GetProperty( TAGPROP_UNITS, v) then raise Exception.Create('');
    SGTags.Cells[ 3, Index] := v;

    //получить строку адреса физического канала
    if not ptag.GetProperty( TAGPROP_HARDWAREADRESS, v) then raise Excep-
tion.Create('');
    SGTags.Cells[ 4, Index] := v;

    //получить код типа связи с физическим каналом
    SGTags.Cells[ 5, Index] := LinkStateToString( ptag.GetLinkState);

    //получить тип тега (записываемы, читаемый и т.д.)
    if not ptag.GetProperty( TAGPROP_TYPE, v) then raise Exception.Create('');
    SGTags.Cells[ 6, Index] := AccessRightToString( v);

    //получить и отобразить частоту дискретизации
    SGTags.Cells[ 7, Index] := FloatToStr( ptag.GetFreq);

    //получить и отобразить тип данных тега
    if not ptag.GetProperty( TAGPROP_DATATYPE, v) then raise Exception.Create('');
    SGTags.Cells[ 8, Index] := VarTypeToString(v);

    //получить и отобразить минимальное значение
    if not ptag.GetProperty( TAGPROP_MINVALUE, v) then raise Exception.Create('');
    SGTags.Cells[ 9, Index] := v;

    //получить и отобразить максимальное значение
    if not ptag.GetProperty( TAGPROP_MAXVALUE, v) then raise Exception.Create('');
    SGTags.Cells[10, Index] := v;

    Result:= true;
  //если в процессе получения свойств тега произвошлпа ошибка, то...
  except
```

```
    //строку в таблицу и не добавляли...
    SGTags.Rows[Index].Clear;
    SGTags.RowCount := SGTags.RowCount - 1;

    Result:= false;
  end;
end;

//Функция преобразования кода типа данных Variant в строковое представление
//Правильного преобразования нет, пока только цифра в строку
function TTestForm.VarTypeToString(const v: Variant): string;
begin
  Result:= IntToStr( VarType(v));
end;

//Функция преобразования кода типа связи тега с физическим каналом в строку
//Правильного преобразования нет, пока только цифра в строку
function TTestForm.LinkStateToString(const v: LINKSTATE): string;
begin
  Result:= IntToStr( integer(v));
end;

//Функция преобразования кода типа доступа к тегу в строку
//Правильного преобразования нет, пока только цифра в строку
function TTestForm.AccessRightToString(const v: integer): string;
begin
  Result:= IntToStr( v and TTAG_OUTPUT);
end;

//проверить корректность выбранной строки в таблице
//тегов (на форме), автоматическое разрешение или запрещение
//кнопок удаления и переименования объекта тега
procedure TTestForm.CheckTagSelection( const row: integer);
var tmp: boolean;
begin
  tmp:= (row > 0) and ( row < (SGTags.RowCount - 1));
  BtnTagDel.Enabled := tmp;
  BtnTagRename.Enabled:= tmp;
end;

//Метод обработки собятия отображения страницы списка тегов.
//Перед тем как показать страницу необходимо инициализировать ее,
//необходимо очистить таблицу тегов...
procedure TTestForm.TSTagsShow(Sender: TObject);
begin
  //Очистка таблицы тегов, инициализация строки заголовка
  SGTags.RowCount := 2;
  SGTags.Rows[1].Clear;
  SGTags.Cells[ 0, 0] := 'Имя';
  SGTags.Cells[ 1, 0] := 'Описание';
  SGTags.Cells[ 2, 0] := 'Группа';
  SGTags.Cells[ 3, 0] := 'Размерность';
  SGTags.Cells[ 4, 0] := 'Физический адрес';
  SGTags.Cells[ 5, 0] := 'Связь с каналом';
  SGTags.Cells[ 6, 0] := 'Права доступа';
  SGTags.Cells[ 7, 0] := 'Частота дискретизации';
  SGTags.Cells[ 8, 0] := 'Тип данных';
  SGTags.Cells[ 9, 0] := 'Минимум';
  SGTags.Cells[10, 0] := 'Максимум';

  //кнопки удаления и переименования тегов должны быть запрещены
  CheckTagSelection( SGTags.Row);
end;

//Обработка нажатия кнопки получения списка тегов.
//Производится получение списка тегов у объекта Recorder, получение
//свойств тегов и добавление строк в таблицу тегов.
procedure TTestForm.BtnGetTagsClick(Sender: TObject);
var
  prec: IRecorder;
  ptag: ITag;
```

```
  i: integer;
  Count: integer;
begin
  //предварительная очистка таблицы тегов
  SGTags.RowCount := 2;
  SGTags.Rows[1].Clear;

  //получение ссылки на объект Recorder
  prec:= TTestPlugin(FPlugin).FIRecorder;
  //получение общего кол-ва тегов
  Count:= prec.GetTagsCount;
  //в цикле получение ссылки на тег по индексу тега
  for i:= 0 to Count - 1 do
  begin
    //получение ссылки на тег
    pointer(ptag):= prec.GetTagByIndex (i);
    //добавление строки с параметрами тега в таблицу
    AddTagToList(ptag);
    //освобождение ссылки на тег
    ptag := nil;
  end;

  //проверить корректность выбранной строки ( тега) в таблице
  CheckTagSelection( SGTags.Row);
end;

//Обработка выбора строки в таблице тегов
procedure TTestForm.SGTagsSelectCell(Sender: TObject; ACol,
  ARow: Integer; var CanSelect: Boolean);
begin
  //проверить корректность выбранной строки ( тега) в таблице
  CheckTagSelection( ARow);
end;

//Обарботка нажатия кнопки добавления нового тега.
//Производится получение списка физических каналов, предоставление
//оператору выбора физического канала, создание тега для физического канала.
procedure TTestForm.BtnTagAddClick(Sender: TObject);
var
  PhChans: TStringList;  {Список строк описывающих физические каналы}
  prec: IRecorder;
  ptag: ITag;
  Count: integer;
  ChanIDs: DynCardinals; {Массив идентификаторов физических каналов}
  Addresses: DynStrings; {Массив строк адресов физических каналов}
  Devices: DynStrings;   {Массив строк имен устройств физических каналов}
  i: integer;
begin
  {Создание объекта списка строк}
  PhChans:= TStringList.Create;
  {Создание объекта формы для выбора физического канала}
  FormPhChans := TFormPhChans.Create( Self);

  try
    {Получение ссылки на объект Recorder}
    prec:= TTestPlugin(FPlugin).FIRecorder;

    //получение списка идентификаторов и строк адресов доступных физических каналов
    if  GetPhChansDsc( ChanIDs, Addresses, Devices) and (Length( ChanIDs) > 0) then
    begin
      //формирование из строки адреса и строки имени устройства
      Count := Length( ChanIDs);
      //строку полного имени устройства, получение таких строк
      //для всех доступных физических кналов, эти строки будут использоваться для
      //отображения оператору, чтобы он мог выбрать для какого кнала создается тег.
      for i:= 0 to Count - 1 do
        PhChans.Add(  Devices[i] +'{'+ Addresses[i] + '}');

      //Активизация формы со списокм физических каналов, предоставление оператору
      //возможности выбора физического канала для которого создается тег.
      //Форма возвращает номер выбранного оператором канкла в списке PhChans.
```

```
      if FormPhChans.Execute( PhChans, i) and (i >= 0) and (i < PhChans.Count) then
      begin
        //переход к режиму настройкии проверка корректности перехода
        if BeginConfigure then
        begin
          try
            //по номеру физического канала выбранного оператором
            //получаем идентификатор канала и имя, на основе этих
            //данных создается тег
            pointer(ptag) := prec.CreateTag( PChar(PhChans.Strings[i]),
                                             LS_HARDWARE, ChanIds[i]);
            //если тег создан успешно и получена ссылка на него, то...
            if Assigned(ptag) then
               //добавить тег в таблицу
               AddTagToList( ptag)
            else
               Application.MessageBox('Не удалось создать новый тег!',
                                      'Ошибка!', MB_OK + MB_ICONERROR);
          finally
            //в завершении необходимо выйти из режима настройки
            EndConfigure;
          end;
        end
        else
          Application.MessageBox('Нет возможности перейти в режим настройки!',
                                 'Ошибка!', MB_OK + MB_ICONERROR);
      end;
    end
    else
       Application.MessageBox( 'Нет свободных физических каналов, либо'  + #13#10 +
                               'ошибка получения параметров физических каналов!',
                               'Ошибка!', MB_OK + MB_ICONERROR );
  finally
    //Удаление ненужной формы и списка строк
    FreeAndNil (PhChans);
    FreeAndNil(FormPhChans);
  end;
  //Проверка корректного выбора строки в таблице тегов
  CheckTagSelection( SGTags.Row);
end;

//Метод обработчик события нажатия кнопки создания виртуального тега.
//Выполняется ввод оператором имени нового тега и создание виртульаного тега.
procedure TTestForm.BtnTagAddVirtClick(Sender: TObject);
var
  prec: IRecorder;
  ptag: ITag;
  TagName: string; //переменная для формирования имени виртуального тега
begin
  //создание формы для ввода имени нового тега
  FormRename := TFormRename.Create( Self);

  try
    //получение ссылки на объект Recorder
    prec:= TTestPlugin(FPlugin).FIRecorder;

    //имя нового тега по умолчанияю
    TagName := 'Новый тег';

    //Активизация формы ввода имени тега
    if FormRename.Execute( TagName) then
    begin
      //переход в режим настройки и проверка корректности перехода
      if BeginConfigure then
      begin
        try
          //создание нового тега с необходимым именем
          pointer(ptag) := prec.CreateTag( PChar( TagName), LS_VIRTUAL, TagName);
          //если тег создан и на него корректно получана ссылка, то...
          if Assigned(ptag) then
             //добавить тег в таблицу
```

```
            AddTagToList( ptag)
          else
            Application.MessageBox('Не удалось создать новый тег!',
                                   'Ошибка!', MB_OK + MB_ICONERROR);
        finally
          //в завершении необходимо выйти из режима настройки
          EndConfigure;
        end;
      end
      else
        Application.MessageBox('Нет возможности перейти в режим настройки!',
                               'Ошибка!', MB_OK + MB_ICONERROR);
    end;
  finally
    //Удаление ненужной формы
    FreeAndNil(FormRename);
  end;
  //Проверка корректного выбора строки в таблице тегов
  CheckTagSelection( SGTags.Row);
end;

//Обработка нажатия кнопки переименования тега.
//Выполняется попытка переименовать тег, который выбран в
//таблице тегов.
procedure TTestForm.BtnTagRenameClick(Sender: TObject);
var
  TagName: string; {Переменная для формирования нового имени тега}
  prec: IRecorder;
  ptag: ITag;
begin
  //Проерка корректности выбора строки таблицы тегов...
  //... не нулевая (не заголовок) и не последняя (не служебная)...
  if (SGTags.Row > 0) and (SGTags.Row < (SGTags.RowCount - 1)) then
  begin
    //создание объекта формы для редактирования имени тега
    FormRename := TFormRename.Create( self);
    try
      //получение "старого" имени тега из таблицы (по номеру выбранной строки)
      TagName := SGTags.Cells[0, SGTags.Row];
      //Активизация формы редактирования имени,
      //форма возвращает "новое" имя тега в переменной TagName.
      if FormRename.Execute( TagName) then  //
      begin
        //переход к режиму настройки, и проверка корректности перехода
        if BeginConfigure then
        begin
          try
            //получение ссылки на объект Recorder
            prec:= TTestPlugin(FPlugin).FIRecorder;
            //получение ссылки на тег по "старому" имени тега
            pointer(ptag):= prec.GetTagByName( PChar(SGTags.Cells[0, SGTags.Row]));

            //Проверка корректности полученной ссылки на тег
            if Assigned( ptag) then
            begin
              //изменение имени тега
              if ptag.SetName( PChar(TagName)) then
              begin
                ptag:= nil;
                //Изменение имени группы в таблице
                SGTags.Cells[0, SGTags.Row] := TagName;
              end
              else
                Application.MessageBox( 'Не удалось переименовать тег!',
                                        'Ошибка!', MB_OK + MB_ICONERROR);
            end;
          finally
            //в завершении необходимо выйти из режима настройки
            EndConfigure;
          end;
        end
```

```
        else
           Application.MessageBox( 'Нет возможности перейти в режим настройки!',
                                   'Ошибка!', MB_OK + MB_ICONERROR);
      end;
    finally
      //Удаление ненужной формы
      FreeAndNil (FormRename);
    end;
  end
  else
    Application.MessageBox( 'Внутренняя ошибка, неправильно выбран тег!',
                            'Ошибка!', MB_OK + MB_ICONERROR);
end;

//Обработчик события нажатия кнопки удаления тега.
//Выполняется попытка удалить тега, выбранный в таблице.
procedure TTestForm.BtnTagDelClick(Sender: TObject);
var
  prec: IRecorder;
  ptag: ITag;
begin
  //Проерка корректности выбора строки таблицы тегов...
  //... не нулевая (не заголовок) и не последняя (не служебная)...
  if (SGTags.Row > 0) and (SGTags.Row < (SGTags.RowCount - 1)) then
  begin
    //переход к режиму настройки, и проверка корректности перехода
    if BeginConfigure then
    begin
      try
        //получение ссылки на объект Recorder
        prec:= TTestPlugin(FPlugin).FIRecorder;
        //получение ссылки на тег по "старому" имени тега
        pointer(ptag):= prec.GetTagByName( PChar(SGTags.Cells[0, SGTags.Row]));

        //Если получена корректная ссылка на тег, то...
        if Assigned( ptag) then
        begin
          //по ссылке тег удаляется
          if prec.CloseTag(ptag) then
          begin
            ptag:= nil;
            //удаление строки из таблицы
            DelRowWithUpShift( SGTags, SGTags.Row);
          end
          else
            Application.MessageBox( 'Не удалось удалить тег!',
                                    'Ошибка!', MB_OK + MB_ICONERROR);
        end;
      finally
        //в завершении необходимо выйти из режима настройки
        EndConfigure;
      end;
    end
    else
       Application.MessageBox( 'Нет возможности перейти в режим настройки!',
                               'Ошибка!', MB_OK + MB_ICONERROR);
  end
  else
    Application.MessageBox( 'Внутренняя ошибка, неправильно выбран тег!',
                            'Ошибка!', MB_OK + MB_ICONERROR);

  //Проверка корректного выбора строки в таблице тегов
  CheckTagSelection( SGTags.Row);
end;

{-------------------------------------------------------------------------------}
{                      Страница промотра списка физических каналов              }
{-------------------------------------------------------------------------------}

//Получение трех массивов описывающих список физических адресов.
//Данный метод производит получение списка "свободных" физических каналов,
```

```
//для которых еще не созданы теги, и заполняет три динамических массива:
//массив идентификаторов каналов, массив строк адресов, массив строк имен
//измерительных устройств, которым принадлежат физические каналы.
function TTestForm.GetPhChansDsc( var ChanIDs: DynCardinals; var Addresses: Dyn-
Strings;
                                  var Devices: DynStrings): boolean;
var
  buffer: array [byte] of char; {Буфер для получения строк}
  buflen: integer;              {Переменная для хранения размера буфера строк}
  prec: IRecorder;
  Count: integer;
  i: integer;
begin
  try
    {Получение ссылки на объект Recorder}
    prec:= TTestPlugin(FPlugin).FIRecorder;
    //получение кол-ва физических каналов
    Count:= prec.dvchGetAvailableChansCount;

    //Установить размеры результирующих динамических массивов
    SetLength(ChanIDs, Count);    //идентификаторы
    SetLength(Addresses, Count);  //строки адресов
    SetLength(Devices, Count);    //строки имен устройств

    //если ко-лво физических каналов больше 0, то...
    if Count > 0 then
    begin
      //Заполнение массива идентификаторов каналов
      if FAILED(prec.dvchGetChansList( ChanIDs[0])) then raise Exception.Create('');

      //цикл по всем идентификаторам каналов
      for i:= 0 to Count - 1 do
      begin
        buflen:= sizeof(buffer) - 1;
        //получение строки адреса физического канала по идентификатору,
        //получение строки происходит в строковый буфер buffer,
        //необходимо быть уверенным, что строка не будет больше буфера.
        if FAILED( prec.dvchGetAddress(ChanIDs[i], @buffer, buflen)) then
           raise Exception.Create('');
        //установка строки в динамический массив
        Addresses[i]:= PChar(@buffer);

        buflen:= sizeof(buffer) - 1;
        //получение строки имени устройства по идентификатору канала
        if FAILED( prec.dvchGetDeviceName(ChanIDs[i], @buffer, buflen)) then
           raise Exception.Create('');
        //установка строки в динамический массив
        Devices[i]:= PChar(@buffer);
      end;
    end;

    Result:= true;
  //Если ошибка произошла с функциями получения
  //параметров физических каналов.
  except
    SetLength( ChanIDs, 0);
    SetLength( Addresses, 0);
    SetLength( Devices, 0);
    Result:= false;
  end;
end;

//Метод обработчик события отображения страницы списка физических каналов,
//производится инициализация страницы
procedure TTestForm.TSPChansShow(Sender: TObject);
begin
  //Очистка таблицы физических каналов
  SGPhChans.RowCount := 2;
  //Инициализация строки заголовка таблицы
  SGPhChans.Cells[0, 0] := 'Устройство';
  SGPhChans.Cells[1, 0] := 'Физичекский адрес';
```

```
  //очистка служебной (последней) строки
  SGPhChans.Rows[1].Clear;
end;

//Обработка нажатия кнопки получения списка доступных физических каналов
procedure TTestForm.BtnGetPhChansClick(Sender: TObject);
var
  ChanIDs: DynCardinals; {Массив идентификаторов физических каналов}
  Addresses: DynStrings; {Массив строк адресов физических каналов}
  Devices: DynStrings;   {Массив строк имен устройств физических каналов}
  Count: integer;
  i: integer;
begin
  //получение списка идентификаторов и т.и. доступных физических каналов
  if  GetPhChansDsc( ChanIDs, Addresses, Devices) then
  begin
    //получить кол-во доступных физических каналов
    Count := Length( ChanIDs);

    //установить размер (кол-во) строк таблицы
    SGPhChans.RowCount := Count + 2;
    //Вывести в таблицу параметры доступных физических каналов
    for i:= 0 to Count - 1 do
    begin
      SGPhChans.Cells[0, i + 1] := Devices[i];   {имя устройства}
      SGPhChans.Cells[1, i + 1] := Addresses[i]; {строка адреса}
    end;
  end
  else
     Application.MessageBox( 'Нет свободных физических каналов, либо'  + #13#10 +
                             'ошибка получения параметров физических каналов!',
                             'Ошибка!', MB_OK + MB_ICONERROR );
end;

{-------------------------------------------------------------------------------}
{                              Страница промотра ГФ/КФ                          }
{-------------------------------------------------------------------------------}

//Метод по имени выбранного на форме тега производит получение
//ссылки на тег и при помощи фрейма отображения ГФ/КФ производит
//вывод параметров ГФ/КФ.
function TTestForm.ShowCurrentCalibration: boolean;
var
  Index : integer;
  prec: IRecorder;
  ptag: ITag;
begin
  //получить индекс, выбраного в данный момент, тега
  Index := ComboBoxTags.ItemIndex;
  //если индекс на корректен, то...
  if (Index >= 0) and ( Index < ComboBoxTags.Items.Count) then
  begin
    //получить ссылку на Recorder
    prec:= TTestPlugin(FPlugin).FIRecorder;
    //получить ссылку на тег по имени
    pointer(ptag) := prec.GetTagByName( PChar(ComboBoxTags.Items[Index]));
    //отобразить КФ/ГФ по ссылке на тег, для этого вызывается фрейм
    //отображения, параметром ему передается ссылка на тег и признак
    // КФ (аппаратной функции), в зависимости от того, какую
    //страницу выбрал оператор (КФ или ГФ)
    Result:= FrameClbShow.ShowCalibration ( ptag, TabControlClb.TabIndex = 0);
  end
  else
    //Отображение пустого окна
    Result:= FrameClbShow.ShowCalibration ( nil, TabControlClb.TabIndex = 0);
end;

//Обработка отображения страницы промотра ГФ/КФ
procedure TTestForm.TSClbsShow(Sender: TObject);
begin
  ShowCurrentCalibration; {Обновить отображение ГФ/КФ}
```

```
end;

//Обработка события смены текущего тега
procedure TTestForm.ComboBoxTagsChange(Sender: TObject);
begin
  ShowCurrentCalibration; {Обновить отображение ГФ/КФ}
end;

//Обработка события смены просматриваемой функции ГФ/КФ
procedure TTestForm.TabControlClbChange(Sender: TObject);
begin
  ShowCurrentCalibration; {Обновить отображение ГФ/КФ}
end;

//Обработка события просмотра списка тегов, получение списка
//тегов и заполнение элемента ComboBoxTags, чтобы оператор мог
//выбрать тот тег, который его интересует.
procedure TTestForm.ComboBoxTagsDropDown(Sender: TObject);
var
  i : integer;
  prec: IRecorder;
  ptag: ITag;
begin
  //очистка старого списка
  ComboBoxTags.Clear;
  //очистка отображения
  ShowCurrentCalibration;
  //получение ссылки на объект Recorder
  prec:= TTestPlugin(FPlugin).FIRecorder;
  //в цикле получение списка тегов
  for i:= 0 to prec.GetTagsCount - 1  do
  begin
    //получение ссылки на тег, по индексу
    pointer(ptag) := prec.GetTagByIndex( i);
    //добавление строки имени тега в список строк ComboBoxTags
    ComboBoxTags.Items.Add( ptag.GetName );
    ptag := nil;

  end;
end;

//Обработка нажатия кнопки устновки сестовой КФ/ГФ.
//Это метод обращается к фрейму отображения ГФ/КФ, для
//установки тегу соответствующей функции.
procedure TTestForm.BtnSetTestFuncClick(Sender: TObject);
var
  Index : integer;
  prec: IRecorder;
  ptag: ITag;
begin
  //получить индекс, выбраного в данный момент, тега
  Index := ComboBoxTags.ItemIndex;
  //если индекс на корректен, то...
  if (Index >= 0) and ( Index < ComboBoxTags.Items.Count) then
  begin
    //получить ссылку на Recorder
    prec:= TTestPlugin(FPlugin).FIRecorder;
    //получить ссылку на тег по имени
    pointer(ptag) := prec.GetTagByName( PChar(ComboBoxTags.Items[Index]));

    //обязательный переход к режиму настройки
    if BeginConfigure then
    begin
      try
        //Установить тестовую КФ/ГФ, для этого вызывается фрейм
        //отображения, параметром ему передается ссылка на тег и признак
        // КФ (аппаратной функции), в зависимости от того, какую
        //страницу выбрал оператор (КФ или ГФ)
        if not FrameClbShow.SetTestFunc ( ptag, TabControlClb.TabIndex = 0) then
           Application.MessageBox('Ошибка устновки тестовой ГФ/КФ!', 'Ошибка!',
                                  MB_OK + MB_ICONERROR);
```

```
      finally
        //..обязательный выход из режима настройки
        EndConfigure;
      end;
    end
    else
       Application.MessageBox( 'Нет возможности перейти в режим настройки!',
                               'Ошибка!', MB_OK + MB_ICONERROR);
  end
end;

end.
```

A.4.4 Модуль «ClbShowFrameUnit.pas»

```
{-----------------------------------------------------------------------}
{ Проект (Модуль) реслизации plug-in`а для измерительного ПО Recorder   }
{ Модуль описания класса фрейма отображения любой ГФ. Отображение       }
{ необходимого фрейма в зависимости от типа ГФ.                         }
{ Компилятор: Borland Delphi 6.0                                        }
{ НПП "ООО Мера" 2004г.                                                 }
{-----------------------------------------------------------------------}


unit ClbShowFrameUnit;

interface

uses
  Windows, Messages, SysUtils, Classes,
  Graphics, Controls, Forms, Dialogs,
  StdCtrls, ExtCtrls,
  ClbScaleFrameUnit, ClbLineFrameUnit,
  ClbIntFrameUnit, ClbPolFrameUnit,
  tags, transf;

type
  {Специализированный фрейм для отображения параметров объекта ГФ/КФ}
  {любого тега. Фрейм получает из тега ссылку на объект ГФ/КФ.      }
  {В зависимости от типа объекта ГФ/КФ, активизируется спец. фрейм  }
  {и с его помощью производится отображение.}
  TFrameClbShow = class(TFrame)
    GroupBoxCommon: TGroupBox;
    ComboBoxClbType: TComboBox;
    CheckBoxUse: TCheckBox;
    PanelClbEditor: TPanel;
  private
    FClbScl : TFrameClbScale; {ссылка на фрейм для отображение "Масштаба"}
    FClbLin : TFrameClbLine;  {ссылка на фрейм для отображение "Линейная"}
    FClbInt : TFrameClbInt;   {ссылка на фрейм для отображение "Таблица интеполяции"}
    FClbPol : TFrameClbPol;   {ссылка на фрейм для отображение "Полином"}

    {ссылка на фрейм, который отображается в данный момент,}
    {эта ссылка может быть равна nil, если ничего не отображается}
    FCurrentEditor : TFrame;

    {Внутренний метод для активизации специализированного фрейма}
    procedure SetCurrentEditor( Editor: TFrame);

    {Метод преобразования типа ГФ/КФ в индекс для отображения}
    function TTToIndex(TT: TXTYPE): integer;
  public
    {конструктор - инициализация внутренних объектов}
    constructor Create(AOwner: TComponent); override;
    {деструктор - удаление внутренних объектов}
    destructor Destroy; override;
```

```
    {Метод обображение в фрейме параметров необходимой ГФ/КФ тега.}
    {В функцию передается ссылка на тег и признак использования
    КФ (аппаратной функции). Если признак DevClb равен true, то
    фрейм должен получить у тега и отобразить КФ, иначе фрейм должен
    у тега получить и отобразить ГФ}
    function ShowCalibration( ptag: ITag; const DevClb: boolean): boolean;

    {Метод установки для тега тестовой функции ГФ/КФ.}
    {В функцию передается ссылка на тег и признак использования
    КФ (аппаратной функции). Если признак DevClb равен true, то
    фрейм должен установить у тега КФ, иначе фрейм должен
    у тега установить ГФ}
    function SetTestFunc ( ptag: ITag; const DevClb: boolean): boolean;
  end;

implementation

{$R *.dfm}

constructor TFrameClbShow.Create(AOwner: TComponent);
begin
  //вызов метода предка
  inherited Create(AOwner);
  //Создание спец. фреймов для отображения параметров КФ/ГФ
  FClbScl := TFrameClbScale.Create( nil);
  FClbLin := TFrameClbLine.Create( nil);
  FClbInt := TFrameClbInt.Create( nil);
  FClbPol := TFrameClbPol.Create( nil);
end;

destructor TFrameClbShow.Destroy;
begin
  //Удаление не нужных более фреймов
  FreeAndNil( FClbScl);
  FreeAndNil( FClbLin);
  FreeAndNil( FClbInt);
  FreeAndNil( FClbPol);
  //вызов метода предка
  inherited Destroy;
end;

{Внутренний метод для активизации специализированного фрейма}
procedure TFrameClbShow.SetCurrentEditor( Editor: TFrame);
begin
  //если был фрейм ранее активный, то...
  if Assigned( FCurrentEditor) then
    //сокрыть его
    FCurrentEditor.Parent := nil;

  //сохранить ссылку на новый фрейм
  FCurrentEditor := Editor;

  //если новый фрейм определен, то его необходимо отобразить
  if Assigned( FCurrentEditor) then
  begin
    //новый фрейм укладывается в спец. панель PanelClbEditor,
    //новый фрейм получает ссылку на объект владельца
    FCurrentEditor.Parent := PanelClbEditor;
    FCurrentEditor.Align := alClient;
  end;
end;

{Метод установки для тега тестовой функции ГФ/КФ.}
{В функцию передается ссылка на тег и признак использования
КФ (аппаратной функции). Если признак DevClb равен true, то
фрейм должен установить у тега КФ, иначе фрейм должен
у тега установить ГФ}
function TFrameClbShow.SetTestFunc ( ptag: ITag; const DevClb: boolean): boolean;
var
  pclb: ITransformer;
  v: OleVariant;
```

```
begin
  try
    //если ссылка на тег не определена, то... установить ничего нельзя
    if not Assigned(ptag) then raise Exception.Create('');

    //В качестве тестовой устанавливается ГФ/КФ "Таблица линейной интеполяции",
    //производится проверка корректности создания объекта ГФ/КФ.
    if (not FClbInt.SetTestFunc( pclb)) or
       (not Assigned(pclb)) then raise Exception.Create('');

    //подготовка параметра для установки ГФ/КФ.
    v := pclb;

    //если КФ, то...
    if DevClb then
      //..установить КФ (аппаратную функцию)
      Result:= ptag.SetProperty( TAGPROP_DEVTARE, v)
    else
    begin
      //...подготовка к установке ГФ...
      if not ptag.Notify(TN_ERASETAGTX,0) then raise Exception.Create('');
      //..установить ГФ (канальную функцию)
      Result:= ptag.SetProperty( TAGPROP_TARE, v);
    end;

    //если ошибка при установке КФ/ГФ,
    if (not Result) then raise Exception.Create('');

    //Отображение установленной функции
    Result := ShowCalibration( ptag, DevClb);
  except
    Result := false;
  end;
end;

{Метод обображение в фрейме параметров необходимой ГФ/КФ тега.}
{В функцию передается ссылка на тег и признак использования
КФ (аппаратной функции). Если признак DevClb равен true, то
фрейм должен получить у тега и отобразить КФ, иначе фрейм должен
у тега получить и отобразить ГФ}
function TFrameClbShow.ShowCalibration( ptag: ITag; const DevClb: boolean): boolean;
var
  punk: IUnknown;
  pclb: ITransformer;
  v: OleVariant;
  bres: boolean;
begin
  try
    //если ссылка на тег не определена, то...нечго отображать
    if not Assigned(ptag) then raise Exception.Create('');

    //если КФ, то...
    if DevClb then
      //..получить ссылку КФ (аппаратную функцию)
      bres:= ptag.GetProperty( TAGPROP_DEVTARE, v)
    else
      //..получить ссылку ГФ (канальную функцию)
      bres:= ptag.GetProperty( TAGPROP_TARE, v);

    //если ошибка при получении КФ/ГФ,
    //либо объект не определен, то... нечго отображать
    if (not bres) then raise Exception.Create('');

    //получение из интерфейса IUnknown необходимого ITransformer
    punk:= v;
    if FAILED( punk.QueryInterface( ITransformer, pclb))  then
       raise Exception.Create('');

    //проанализировать и устновить признак использования
    if DevClb then
      //для КФ определен признак - свойство у тега
```

```
      CheckBoxUse.Checked :=
          ptag.GetProperty( TAGPROP_DEVSIGNALTYPE, v) and (v <> 0)
    else
      //для ГФ - всегда используется
      CheckBoxUse.Checked := true;

    //установить отображение строки типа КФ/ГФ
    ComboBoxClbType.ItemIndex := TTToIndex(pclb.GetTXType);

    //ветвление по типу КФ/ГФ
    case pclb.GetTXType of
      //активизация соответствующего фрйема
      TXT_NULL: begin
        SetCurrentEditor( nil );
        Enabled := false;
      end;
      TXT_SCALE: begin
        SetCurrentEditor( FClbScl);
        FClbScl.UpdateClb( pclb);
        Enabled := true;
      end;
      TXT_LINE: begin
        SetCurrentEditor( FClbLin );
        FClbLin.UpdateClb( pclb);
        Enabled := true;
      end;
      TXT_INTEPOLATETABLE: begin
        SetCurrentEditor( FClbInt);
        FClbInt.UpdateClb( pclb);
        Enabled := true;
      end;
      TXT_POLYNOME: begin
        SetCurrentEditor( FClbPol);
        FClbPol.UpdateClb( pclb);
        Enabled := true;
      end;
      else //неизвестный тип КФ/ГФ - фрейм деактивировать
      begin
        SetCurrentEditor( nil );
        Enabled := false;
      end;
    end;

  except
    //Ошибка работы с объектом тега или КФ/ГФ, либо объект КФ/ГФ отсутствует
    //Фрейм деактивируется, ничего не отображается
    SetCurrentEditor( nil );
    ComboBoxClbType.ItemIndex := -1;
    Enabled := false;
  end;
  Result := true;
end;

type
  {Тип для создания таблицы соответствия}
  {типа ГФ/КФ и индекса}
  TIndexToCFType = record
    TT: TXTYPE;
    Index: integer;
  end;

const
  IndexToCFTypeTableSize = 5; {размер таблицы соответствия}

type
  {тип таблицы соответствия}
  TIndexToCFTypeTable = array[ 0 .. IndexToCFTypeTableSize - 1] of TIndexToCFType;

const
  {константная переменная - таблица соответствия типа ГФ/КФ и индекса}
  IndexToCFTypeTable : TIndexToCFTypeTable = (
```

```
    (TT: TXT_NULL;            Index: 0),
    (TT: TXT_SCALE;           Index: 1),
    (TT: TXT_LINE;            Index: 2),
    (TT: TXT_INTEPOLATETABLE; Index: 3),
    (TT: TXT_POLYNOME;        Index: 4)
  );

{Метод преобразования типа ГФ/КФ в индекс для отображения.}
{Подразумевается индекс в строке ComboBoxClbType.}
function TFrameClbShow.TTToIndex(TT: TXTYPE): integer;
var
  i: integer;
begin
  {поиск в цикле по типу ГФ/КФ}
  for i:= 0 to IndexToCFTypeTableSize - 1 do
    if IndexToCFTypeTable[i].TT = TT then
    begin
      {был найден необходимый тип, вернуть индекс...}
      Result:= IndexToCFTypeTable[i].Index;
      exit;
    end;
  Result:= 5;
end;

end.
```

A.4.5 Модуль «ClbScaleFrameUnit.pas»

```
{-----------------------------------------------------------------------}
{ Проект (Модуль) реслизации plug-in`а для измерительного ПО Recorder   }
{ Модуль описания класса фрейма отображения ГФ "Масштабный коэффициент" }
{ Компилятор: Borland Delphi 6.0                                        }
{ НПП "ООО Мера" 2004г.                                                 }
{-----------------------------------------------------------------------}

unit ClbScaleFrameUnit;

interface

uses
  Windows, Messages, SysUtils, Classes,
  Graphics, Controls, Forms, Dialogs,
  StdCtrls,
  transf, transformers;

type
  {Фрейм отображения ГФ "Масштабный коэффициент" }
  TFrameClbScale = class(TFrame)
    Label1: TLabel;
    EditScale: TEdit;
  public
    {Отобразиь во фрейме параметры ГФ "Масштабный коэффициент"}
    {Паремтром функции является ссылка на интерфейс объекта ГФ}
    function UpdateClb( tgdt: ITransformer; const bFromControl: boolean = false): boo-
lean;
  end;

implementation

{$R *.dfm}

{Отобразиь во фрейме параметры ГФ "Масштабный коэффициент"}
{Паремтром функции является ссылка на интерфейс объекта ГФ}
function TFrameClbScale.UpdateClb( tgdt: ITransformer;
                                   const bFromControl: boolean = false): boolean;
var
  pscl : IScale;
  Scale: double;
```

```
begin
  //Получить необходимый интерфейс ГФ "Масштабный коэффициент"
  //если интерфейс получен успешно, то...
  if SUCCEEDED(tgdt.QueryInterface( IScale, pscl)) then
  begin
    //..получить значение коэффициента, и ...
    pscl.GetScale( Scale);
    //..отобразить его
    EditScale.Text := FloatToStr( Scale);
    Result := true;
  end
  else
    Result := false;
end;
end.
```

A.4.6 Модуль «ClbLineFrameUnit.pas»

```
{-----------------------------------------------------------------------}
{ Проект (Модуль) реслизации plug-in`а для измерительного ПО Recorder }
{ Модуль описания класса фрейма отображения ГФ "Линейная"             }
{ Компилятор: Borland Delphi 6.0                                      }
{ НПП "ООО Мера" 2004г.                                               }
{-----------------------------------------------------------------------}

unit ClbLineFrameUnit;

interface

uses
  Windows, Messages, SysUtils, Classes,
  Graphics, Controls, Forms, Dialogs,
  StdCtrls,
  transf, transformers;

type
  {Фрейм отображения ГФ "Линейная"}
  TFrameClbLine = class(TFrame)
    Label1: TLabel;
    Label2: TLabel;
    EditA: TEdit;
    EditB: TEdit;
  public
    {Отобразиь во фрейме параметры ГФ "Линейная"}
    {Паремтром функции является ссылка на интерфейс объекта ГФ}
    function UpdateClb( tgdt: ITransformer;
                        const bFromControl: boolean = false): boolean;
  end;

implementation

{$R *.dfm}

{Отобразиь во фрейме параметры ГФ "Линейная"}
{Паремтром функции является ссылка на интерфейс объекта ГФ}
function TFrameClbLine.UpdateClb( tgdt: ITransformer;
                                  const bFromControl: boolean = false): boolean;
var
  plin: ILinear;
  a, b: double;
begin
  //Получить необходимый интерфейс ГФ "Линейная"
  Result := SUCCEEDED( tgdt.QueryInterface( ILinear, plin) );
  //если интерфейс получен успешно, то...
  if Result then
  begin
    //получение и отображение коэффициентов
    plin.GetA(a);
```

```
    plin.GetB(b);
    EditA.Text := FloatToStr(a);
    EditB.Text := FloatToStr(b);
  end;
end;

end.
```

A.4.7 Модуль «ClbIntFrameUnit.pas»

```
{-----------------------------------------------------------------------}
{ Проект (Модуль) реслизации plug-in`а для измерительного ПО Recorder }
{ Модуль описания класса фрейма отображения ГФ "Таблица интерполяции" }
{ Компилятор: Borland Delphi 6.0                                        }
{ НПП "ООО Мера" 2004г.                                                 }
{-----------------------------------------------------------------------}

unit ClbIntFrameUnit;

interface

uses
  Windows, Messages, SysUtils, Classes,
  Graphics, Controls, Forms, Dialogs,
  StdCtrls, Grids, ActiveX,
  transf, transformers;

type
  {Фрейм отображения ГФ "Таблица интерполяции".}
  TFrameClbInt = class(TFrame)
    StringGridNodes: TStringGrid;
    Label1: TLabel;
    CheckBoxExtrapolate: TCheckBox;
  public
    {Отобразиь во фрейме параметры ГФ "Таблица интерполяции"}
    {Паремтром функции является ссылка на интерфейс объекта ГФ}
    function UpdateClb( tgdt: ITransformer;
                        const bFromControl: boolean = false): boolean;

    //Метод создания и получения ссылки на объект ГФ
    //с тестовыим значениеми параметров.
    function SetTestFunc( var tgdt: ITransformer): boolean;
  end;

implementation

{$R *.dfm}

{Отобразиь во фрейме параметры ГФ "Таблица интерполяции"}
{Паремтром функции является ссылка на интерфейс объекта ГФ}
function TFrameClbInt.UpdateClb( tgdt: ITransformer;
                                 const bFromControl: boolean = false): boolean;
var
  ttab: IInterpolate;
  Count: integer;
  i: integer;
  x, y: double;
begin
  //Получить необходимый интерфейс ГФ "Таблица интерполяции"
  Result := SUCCEEDED(tgdt.QueryInterface(IInterpolate, ttab));
  //если интерфейс получен успешно, то...
  if Result then
  begin
    //получить кол-во элементов в таблице
    Count:= ttab.GetNodesCounter;

    //Установить параметры отображения таблицы
    StringGridNodes.RowCount := Count + 2;
```

```
    StringGridNodes.Rows[StringGridNodes.RowCount - 1].Clear;

    //Заголовок таблицы отображения
    StringGridNodes.Cells[0, 0] := '№';
    StringGridNodes.Cells[1, 0] := 'x';
    StringGridNodes.Cells[2, 0] := 'f(x)';

    //Отображение таблицы поэлементно
    for i:= 0 to Count - 1 do
    begin
      //получение элемента таблицы у объекта ГФ
      ttab.GetNode(x, y, i);
      StringGridNodes.Cells[0, i + 1] := IntToStr(i + 1);
      StringGridNodes.Cells[1, i + 1] := FloatToStr(x);
      StringGridNodes.Cells[2, i + 1] := FloatToStr(y);
    end;
    //признак экстраполяции за границами...
    CheckBoxExtrapolate.Checked := ttab.GetExtrapolateMode;
  end;
end;

//Метод создания и получения ссылки на объект ГФ
//с тестовыим значениеми параметров.
function TFrameClbInt.SetTestFunc( var tgdt: ITransformer): boolean;
var
  pint: IInterpolate;
begin
  try
    //Создание объекта ГФ/КФ
    if FAILED( CoCreateInstance(CLSID_InterpolateTransformer, nil, CLSCTX_ALL,
               IInterpolate, pint)) then raise Exception.Create('');

    //Установка трех узлов (элементов) "Таблицы линейной интеполяции"
    if FAILED( pint.SetNode( 2, 10, -1)) then raise Exception.Create('');
    if FAILED( pint.SetNode( 4, 100, -1)) then raise Exception.Create('');
    if FAILED( pint.SetNode( 8, 1000, -1)) then raise Exception.Create('');

    //получение общего интерфейса
    if FAILED( pint.QueryInterface( ITransformer, tgdt)) then raise Excep-
tion.Create('');

    Result := true;
  except
    tgdt := nil;
    Result := false;
  end;
end;

end.
```

A.4.8 Модуль «ClbPolFrameUnit.pas»

```
{-----------------------------------------------------------------------}
{ Проект (Модуль) реслизации plug-in`а для измерительного ПО Recorder }
{ Модуль описания класса фрейма отображения ГФ "Полином"              }
{ Компилятор: Borland Delphi 6.0                                      }
{ НПП "ООО Мера" 2004г.                                               }
{-----------------------------------------------------------------------}

unit ClbPolFrameUnit;

interface

uses
  Windows, Messages, SysUtils, Classes,
  Graphics, Controls, Forms, Dialogs,
  StdCtrls, Grids,
  transf, transformers;
```

```
type
  {Фрейм отображения ГФ "Полином"}
  TFrameClbPol = class(TFrame)
    StringGridCoeffs: TStringGrid;
    Label1: TLabel;
  public
    {Отобразиь во фрейме параметры ГФ "Полином"}
    {Паремтром функции является ссылка на интерфейс объекта ГФ}
    function UpdateClb( tgdt: ITransformer;
                        const bFromControl: boolean = false): boolean;
  end;

implementation

{$R *.dfm}

{Отобразиь во фрейме параметры ГФ "Полином"}
{Паремтром функции является ссылка на интерфейс объекта ГФ}
function TFrameClbPol.UpdateClb( tgdt: ITransformer;
                                 const bFromControl: boolean = false): boolean;
var
  i: integer;
  Count: integer;
  ppol : IPolynomial;
  coeff: double;
begin
  //Получить необходимый интерфейс ГФ "Полином"
  Result := SUCCEEDED(tgdt.QueryInterface( IPolynomial, ppol));
  //если интерфейс получен успешно, то...
  if Result then
  begin
    //получить кол-во коэффициентов
    Count:= ppol.GetPower + 1;

    //Установить параметры таблицы отображения
    StringGridCoeffs.RowCount := Count + 2;
    StringGridCoeffs.Rows[StringGridCoeffs.RowCount].Clear;

    //Заголовок таблицы отображения
    StringGridCoeffs.Cells[0, 0] := '№';
    StringGridCoeffs.Cells[1, 0] := 'Коэффициент';

    //в цикле получение и отображение каждого коэффциента
    for i := 0 to Count - 1 do
    begin
      ppol.GetCoefficient( coeff, i);
      StringGridCoeffs.Cells[0, i + 1] := IntToStr(i);
      StringGridCoeffs.Cells[1, i + 1] := FloatToStr(coeff);
    end;
  end;
end;

end.
```

A.4.9 Модуль «ChanFormUnit.pas»

```
{-----------------------------------------------------------------------}
{ Проект (Модуль) реслизации plug-in`a для измерительного ПО Recorder }
{ Модуль описания класса формы для выбора имени физического канала    }
{ из списка.                                                          }
{ Компилятор: Borland Delphi 6.0                                      }
{ НПП "ООО Мера" 2004г.                                               }
{-----------------------------------------------------------------------}

unit ChanFormUnit;

interface
```

```
uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, StdCtrls;

type
  {Форма для выбора имени физического канала из списка.}
  TFormPhChans = class(TForm)
    ListBoxPhChans: TListBox;
    Button1: TButton;
    Button2: TButton;
  public
    //Метод активизации формы, форма отображается (модальный режим).
    //Метод возвращает true, если оператор сделал выбор, иначе false.
    //Параметром метода является список строк и результат выбора
    //оператора - индекс выбранного элемента.
    function Execute( List: TStringList; var SelIndex: integer): boolean;
  end;

var
  FormPhChans: TFormPhChans;

implementation

{$R *.dfm}

//Метод активизации формы, форма отображается (модальный режим).
//Метод возвращает true, если оператор сделал выбор, иначе false.
//Параметром метода является список строк и результат выбора
//оператора - индекс выбранного элемента.
function TFormPhChans.Execute( List: TStringList; var SelIndex: integer): boolean;
begin
  //Отображение необходимого списка строк
  ListBoxPhChans.Items := List;
  //Модальный режим отображения формы
  Result:= ShowModal = mrOk;
  //если оператор нажал "Применить", то...
  if Result then
    //...вернуть номер выбранного элемента списка строк.
    SelIndex := ListBoxPhChans.ItemIndex;
end;

end.
```

A.4.10 Модуль «RenameFormUnit.pas»

```
{-----------------------------------------------------------------------}
{ Проект (Модуль) реслизации plug-in`а для измерительного ПО Recorder   }
{ Модуль описания класса формы для переименования объекта               }
{ Компилятор: Borland Delphi 6.0                                        }
{ НПП "ООО Мера" 2004г.                                                 }
{-----------------------------------------------------------------------}

unit RenameFormUnit;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, StdCtrls;

type
  {Форма для переименования объекта, отображается и редактируется строка имени.}
  TFormRename = class(TForm)
    EditName: TEdit;
    Button1: TButton;
    Button2: TButton;
    Label1: TLabel;
```

```
  public
    {Метод активизации формы редактора имени объекта, начальное имя
    передается параметром метода, если имя оператором изменено, то
    новое возвращается этим - же параметром. Метод возвращает true
    если оператор применил изменение строки...}
    function Execute( var EditedName: string): boolean;
  end;

var
  FormRename: TFormRename;

implementation

{$R *.dfm}

{Метод активизации формы редактора имени объекта, начальное имя
передается параметром метода, если имя оператором изменено, то
новое возвращается этим - же параметром. Метод возвращает true
если оператор применил изменение строки...}
function TFormRename.Execute( var EditedName: string): boolean;
begin
  EditName.Text :=  EditedName;

  {Модальное отображение диалога}
  Result:= ShowModal = mrOk;
  if Result then
    EditedName := EditName.Text;
end;

end.
```

A.5 Plug-in, генерирующий сигналы

A.5.1 Модуль проекта библиотеки «test_wrt.pas»

```
{-----------------------------------------------------------------------}
{ Проект (Модуль) реслизации plug-in`a для измерительного ПО Recorder }
{ Компилятор: Borland Delphi 6.0                                        }
{ НПП "ООО Мера" 2004г.                                                 }
{-----------------------------------------------------------------------}

library test_wrt;

uses
  Windows,
  SysUtils,
  Classes,
  blaccess in '..\interfaces\blaccess.pas',
  DevAPI in '..\interfaces\DevAPI.pas',
  journal in '..\interfaces\journal.pas',
  modules in '..\interfaces\modules.pas',
  plugin in '..\interfaces\plugin.pas',
  recorder in '..\interfaces\recorder.pas',
  signal in '..\interfaces\signal.pas',
  tags in '..\interfaces\tags.pas',
  transf in '..\interfaces\transf.pas',
  transformers in '..\interfaces\transformers.pas',
  waitwnd in '..\interfaces\waitwnd.pas',
  PluginClass in 'PluginClass.pas';

{$R *.res}

{-------------------------------------------------------------Экспортируемые функции}
{Ниже описаны и реализованы пять функций, которые должна экспортировать}
{библиотека plug-in`a для Recorder`a}

{Получение типа объекта, реализованного в билиотеке}
function GetPluginType: integer; cdecl;
begin
```

```
  Result:= PLUGIN_CLASS; {В данном случае, это тип объекта plug-in`a}
end;
{Функция создания экземпляра plug-in`a.}
{Функция объявлена таким образом, как будто она возвращает не интерфейс,
а указатель. Причина такого объявления обоснована в документе описания.}
function CreatePluginClass: pointer{IRecorderPlugin}; cdecl;
begin
  {Данный plug-in поддерживает создание всего одного экземпляра,
   поэтому поэтому просто возвращается ссылка на глобальный объект.}
  {Сам объект не создается здаесь! Объект создается при загрузке библиотеки.}
  Result := pointer(GPluginInstance);
end;
{Функция удаления plug-in`a.}
function DestroyPluginClass(piPlg: IRecorderPlugin): integer; cdecl;
begin
   {Так как объект один и глобальный, то он здесь не удаляется.}
   {Объект plug-in`a удаляет себя сам (как любой COM объект),
    если счетчик ссылок его интерфейсов станет равен нулю.}
   Result:= 0;
end;
{Функция получения строки описания plug-in`a}
function GetPluginDescription: LPCSTR; cdecl;
begin
   {Описание извлекается из глобальной описательной структуры}
   Result:= LPCSTR(GPluginInfo.Dsc);
end;
{Функция получения полного описания plug-in`a}
procedure GetPluginInfo(var lpPluginInfo: PLUGININFO); cdecl;
begin
   {Описание извлекается из глобальной описательной структуры}
   {Копирование строк производится именно функциями StrCopy()
    из-за того, что структура описания plug-in`a описана
    в языке C++}
  StrCopy( @lpPluginInfo.name, LPCSTR(GPluginInfo.Name));
  StrCopy( @lpPluginInfo.describe, LPCSTR(GPluginInfo.Dsc));
  StrCopy( @lpPluginInfo.vendor, LPCSTR(GPluginInfo.Vendor));
  lpPluginInfo.version:= GPluginInfo.Version;
  lpPluginInfo.subversion:= GPluginInfo.SubVertion;
end;

{---------------------Объявление экспортируемых функций-----------------------}
{Объявление строковых имен экспортируемых функций}
exports GetPluginType        name 'GetPluginType';
exports CreatePluginClass    name 'CreatePluginClass';
exports DestroyPluginClass   name 'DestroyPluginClass';
exports GetPluginDescription name 'GetPluginDescription';
exports GetPluginInfo        name 'GetPluginInfo';

end.
```

A.5.2 Модуль «PluginClass.pas»

```
{-------------------------------------------------------------------------}
{ Проект (Модуль) реслизации plug-in`a для измерительного ПО Recorder }
{ Модуль класса тестового plug-in`a, описание и реализация             }
{ Компилятор: Borland Delphi 6.0                                       }
{ НПП "ООО Мера" 2004г.                                                }
{-------------------------------------------------------------------------}

{В данном модуле описан и реализован класс TTestPlugin тестового plug-in`a.}

unit PluginClass;
interface
uses
   Windows,
   SyncObjs, Classes, ExtCtrls,
   recorder, tags, plugin, blaccess;

const
```

```
   VT_Name: string = 'v_plg_tag';
   VT_SF: double = 1000; {aka sampling frequency, Гц}
   VT_UPDATE_PERIOD: integer = 100; {период обновления данных}
   VT_SignalFrequency: double = 11; {Гц}
   VT_Amplitude : double = 200; {Амплитуда генерируемого сигнала}

type
   {Класс тестового plug-in`a. Наследует классу TInterfacedObject,
    при этом наследуется реализация IUnknown и механизм подсчета
    ссылок соответсвенно. Класс реализует интерфейс IRecorderPlugin -
    интерфейс plug-in`a.}
   TTestPlugin = class(TInterfacedObject, IRecorderPlugin)
   public
      {Конструктор}
      constructor Create;
      {Деструктор}
      destructor Destroy; override;

   public    //IRecorderPlugin
      // Создание плагина
      function _Create(pOwner: IRecorder): boolean; stdcall;
      // Конфигурирование
      function Config: boolean; stdcall;
      // Вызов окна настройки
      function Edit: boolean; stdcall;
      // Запуск - активизация работы plug-in`a
      function Execute: boolean; stdcall;
      // Приостановка работы
      function Suspend: boolean; stdcall;
      // Возобновление работы
      function Resume: boolean; stdcall;
      // Уведомление о внешних событиях
      function Notify(const dwCommand: DWORD;
                      const dwData: DWORD): boolean; stdcall;
      // Получение имени
      function GetName: LPCSTR; stdcall;
      // Получить свойство
      function GetProperty(const dwPropertyID: DWORD;
                           var Value: OleVariant): boolean; stdcall;
      // Задать свойство
      function SetProperty(const dwPropertyID: DWORD;
                           {const} Value: OleVariant): boolean; stdcall;

      // Узнать можно ли завершить работу плагина
      function CanClose: boolean; stdcall;
      // Завершить работу плагина
      function Close:  boolean; stdcall;
   protected
      {Ссылка на интерфейс для управления Recorder`ом}
      FIRecorder: IRecorder;
      FTag:   ITag;
      FDataBuffer: array of double;
      FLastWriteTime: DWORD;
      FPhi: double;   //Текущее значение угла
      FAlpha: double; //Приращение угла в одной точке

      //Метод подготовки к изменению конфигурации
      function  BeginConfigure: boolean;
      //Метод подготовки после того, как конфигурация была изменена
      procedure EndConfigure;

      //Метод обработки события начала изменения конфигурации
      function  OnBeginConfigure: boolean;
      //Метод обработки события завершения изменения конфигурации
      procedure OnEndConfigure;
      //Метод обработки события начала измерения
      procedure OnStart;
      //Метод обработки события завершения измерения
      procedure OnStop;
   protected
      FTimer: TTimer;
```

```
      procedure OnTimer(Sender: TObject);
   end;


var
  {В библиотеке создается всего один глобальный экземпляр plug-in`a, и }
  {соответсвенно есть переменная ссылка на интерфейс этого plug-in`a.}
  {Экземпляр plug-in`a создается по загрузке библиотеки, и должен удаляться,}
  {когда счетчик ссылок станет равен нулю, это должно происходить перед}
  {выгрузкой библиотеки (если подсчет ссылок организоан правильно).    }
  GPluginInstance: IRecorderPlugin = nil;

type
   {Тип для хранения информации о plug-in`e}
   {Этот тип удобнее использовать в Delphi, чем PLUGININFO}
  TInternalPluginInfo = record
    Name:           string; //наименование plug-in`a
    Dsc:            string;//строка описания
    Vendor:  string;//строка наименования фирмы
    Version: integer;//номер версии
    SubVertion:     integer;//номер подверсии
  end;

const
   {Глобальная переменная для храения описания plug-in`a.}
  GPluginInfo: TInternalPluginInfo = (
    Name:           'Delphi Тест';
    Dsc:            'Тестирование';
    Vendor:  'ООО НПП Мера';
    Version: 0;
    SubVertion:     1;
  );

implementation
uses
  SysUtils, Forms, Variants;

{Конструктор} {Создание формы тестового plug-in`a}
constructor TTestPlugin.Create;
begin
  FTag:= nil;
  FTimer:= TTimer.Create( nil);
  FTimer.OnTimer := OnTimer;
  FTimer.Interval := VT_UPDATE_PERIOD;
  FTimer.Enabled := false;
end;
{Деструктор} {Удаление формы тестового plug-in`a}
destructor TTestPlugin.Destroy;
begin
  FreeAndNil( FTimer);
  inherited Destroy;
end;

//IRecorderPlugin
// Создание плагина
function TTestPlugin._Create(pOwner: IRecorder): boolean;
begin
  FIRecorder:= pOwner; // сохранение полученной ссылки на интерфейс Recorder`a
  Result:= true;
end;

// Конфигурирование
function TTestPlugin.Config: boolean;
begin
  Result:= true;
end;
// Вызов окна настройки
function TTestPlugin.Edit: boolean;
begin
  Result:= true;
end;
```

```
// Запуск - активизация работы plug-in`a
function TTestPlugin.Execute: boolean;
var
  v: OleVariant;
begin
  //если успешно удалось перейти к режиму изменения настройки, то...
  if BeginConfigure then
  begin
    try
      //добавление нового тега
      pointer(FTag):= FIRecorder.CreateTag( PChar(VT_Name),
                                            LS_VIRTUAL, FTag);
      //если не удалось создать тег и ссылка на новый тег не нулевая, то...
      if not Assigned( FTag) then
         Application.MessageBox( 'Не удалось сздать тег!',
                                  'Ошибка!', MB_OK + MB_ICONERROR)
      else
      begin
        {Установка требуемой частоты дискретизации}
        FTag.SetFreq( VT_SF);
        //Установка типа данных тега
        v:= 0; //по умолчанию, присваивая переменной типа Variant
               //действительное значение, получаем Variant с типом
               //Currency (хотя нужен тип Double)
        v:= VarAsType(v, varDouble);
        FTag.SetProperty( TAGPROP_DATATYPE, v);

        //становка минимума и максимума
        v := VT_Amplitude * 1.1 / 2; //максимум от значения амплитуды
        v:= VarAsType(v, varDouble);
        FTag.SetProperty( TAGPROP_MINVALUE , v);

        v := - VT_Amplitude * 1.1 / 2; //минимум от значения амплитуды
        v:= VarAsType(v, varDouble);
        FTag.SetProperty( TAGPROP_MAXVALUE, v);
      end;
    finally
      EndConfigure; //проццесс изменения настройки завершен
    end;
  end
  else
     Application.MessageBox( 'Нет возможности перейти в режим настройки!',
                              'Ошибка!', MB_OK + MB_ICONERROR);
  Result:= true;
end;
// Приостановка работы
function TTestPlugin.Suspend: boolean;
begin
  Result:= true;
end;
// Возобновление работы
function TTestPlugin.Resume: boolean;
begin
  Result:= true;
end;
// Уведомление о внешних событиях
function TTestPlugin.Notify(const dwCommand: DWORD; const dwData: DWORD): boolean;
begin
  case dwCommand of
    PN_ENTERRCCONFIG: begin // перешел в режим настройки
      OnBeginConfigure;     //обработать начало изменения настройки
      Result:= true;
    end;
    PN_LEAVERCCONFIG: begin // вышел из режима настройки
      OnEndConfigure;       //обработать завершение изменения настройки
      Result:= true;
    end;
    PN_RCSTART: begin //начало измерения
      OnStart;        //обработка события начала измерения
      Result:= true;
    end;
```

```
    PN_RCSTOP: begin //конец измерения
      OnStop;        //обработка события
      Result:= true;
    end;
    else
      Result:= false; //прочие сообщения не обрабатываем
  end;
end;

// Получение имени
function TTestPlugin.GetName: LPCSTR;
begin
   Result:= LPCSTR(GPluginInfo.Name); {Из глобальной структуры описания plug-in`a}
end;

// Получить свойство
function TTestPlugin.GetProperty(const dwPropertyID: DWORD;
                  var Value: OleVariant): boolean;
begin
  case dwPropertyID of
     PLGPROP_INFOSTRING: begin    {Получить строку описания plug-in`a}
        Value := GPluginInfo.Dsc; {Из глобальной структуры описания plug-in`a}
        Result:= true;
     end;
     else
        Result:= false;
  end;
end;
// Задать свойство
function TTestPlugin.SetProperty(const dwPropertyID: DWORD;
                  {const} Value: OleVariant): boolean;
begin
  Result:= true;
end;

// Узнать можно ли завершить работу плагина
function TTestPlugin.CanClose: boolean;
begin
  Result:= true;
end;
// Завершить работу плагина
function TTestPlugin.Close:  boolean;
begin
  Result:= true;
end;

//Метод подготовки к изменению конфигурации
function  TTestPlugin.BeginConfigure: boolean;
begin
  //Нельзя менять настройку если режим настройки не включен
  Result:= SUCCEEDED(FIRecorder.EnterConfigMode( Self, 0));
end;

//Метод выхода из режима настройки
procedure TTestPlugin.EndConfigure;
begin
  //Изменения внесенные в настройку ПО Recorder вступят
  //в силу только после выхода из режима настройки
  FIRecorder.LeaveConfigMode( Self, 0)
end;

//Метод обработки события начала изменения конфигурации
function  TTestPlugin.OnBeginConfigure: boolean;
begin
  Result := false;
end;

//Метод обработки события завершения изменения конфигурации
procedure TTestPlugin.OnEndConfigure;
var
  BufferSize: integer;
```

```
  v: OleVariant;
begin
  //Получить размер одного блока из буфера тега.
  if Assigned( FTag) then
  begin
    //Получить у тега размер одного блока буфера
    FTag.GetProperty( TAGPROP_BUFFSIZE, v);
    BufferSize:= v;
    //Выделить блок памяти для формирования данных.
    SetLength( FDataBuffer, BufferSize);
  end;

  //Расчет приращения угла в одной точке измерения
  //Угол в радианах, зависит от частоты сигнала и
  //частоты дискретизации тега...
  FAlpha := 2*Pi*VT_SignalFrequency / VT_SF;

  FPhi:= 0;
end;

//Метод обработки события начала измерения
procedure TTestPlugin.OnStart;
begin
  FTimer.Enabled := true;
  FLastWriteTime:= GetTickCount();
end;

//Метод обработки события завершения измерения
procedure TTestPlugin.OnStop;
begin
  FTimer.Enabled := false;
end;

procedure TTestPlugin.OnTimer(Sender: TObject);
var
  CurTime: DWORD;     {Текущее время в миллисекундах от запуска компьютера}
  DeltaTime: integer; {Время прошедшее с последней операции записи}
  BufferSize: integer;{Размер блока данных}
  n, i: integer;      {Счестчики циклов, по блокам и по элементам блоков}
  BlocksCount: integer; {Кол-во блоков, которые необходимо записать}
begin
  {Текущее время в миллисекундах от запуска компьютера}
  CurTime:= GetTickCount();
  //Вычисляем время прошедшее от последнего шага генерации данных.
  //Предполагаем, что нпрерывная работы plug-in`а по времени
  //не будет дольше 49.7 суток и переменные CurTime, FLastWriteTime
  //не переполнятся.
  DeltaTime:= CurTime - FLastWriteTime;
  //Размер одного блока данных
  BufferSize:= Length( FDataBuffer);
  //Кол-во блоков, которые необходимо записать на данном шаге
  BlocksCount:= Trunc(DeltaTime * VT_SF / 1000 / BufferSize);
  //Если кол-во блоков не нулевое, то  необходимо запомнить
  //время текущего шага записи как последнего
  if BlocksCount >= 1 then
     FLastWriteTime := CurTime; //Запоминаем время текущего шага
  //По времени прошедшему от последнего шага генерации сигнала,
  //по частоте дискретизации и по размеру буфера определяем
  //кол-во блоков, которое необходимо передать в тег на данном шаге...
  for n:= 0 to BlocksCount - 1 do
  begin
    //В цикле заполняем каждый блок, каждое значение в блоке
    for i:= 0 to BufferSize - 1 do
    begin
      //Расчитываем угол, для синуса...
      FPhi := FPhi + FAlpha; //..., где
      //FAlpha - это изменение угла от одного до другого значения
      //(генерируемых) данных в буфере, зависит от частоты
      //генерируемого сигнала и от частоты дискретизации тега.

      //Периодический сигнал синус, вычитание периода...
```

```
      if (FPhi >= 2 * Pi) then
         FPhi := FPhi - 2 * Pi;

      //Расчет значения генерируемых данных (по углу)
      FDataBuffer[i] := (VT_Amplitude / 2) * sin( FPhi);
    end;
    //Установить блок данных в буфер тега...
    FTag.PushData( pointer(FDataBuffer)^, -1);
  end;
  //
end;

initialization
   {Код создания одного экземпляра объетка plug-in`a, объект создается и
   в глобальной переменной сохраняется ссылка на интерфейс объекта. Объект
   удаляется автоматически, когда глобальная ссылка освобождается, то есть
   когда библиотека plug-in`a выгружается.}
   GPluginInstance:= TTestPlugin.Create;
end.
```

В Приложение. Распечатка кода программы plug-in`a, разработанного при помощи Borland® C++ Builder

В.1 Минимальный код plug-in`a.

В.1.1 Модуль "TestPlugin.h"

```
/*-------------------------------------------------------------------------*/
/* Проект (Модуль) реслизации plug-in`a для измерительного ПО Recorder */
/* Модуль класса тестового plug-in`a, описание                             */
/* Компилятор: Borland C++ Builder 5.0                                     */
/* НПП "ООО Мера" 2004г.                                                   */
/*-------------------------------------------------------------------------*/

#ifndef TestPluginH
#define TestPluginH

#include <windows.h>
#include <rcplugin.h>

//---------------------------------------------------------------------------
class TTestPlugin: public IRecorderPlugin
{
private:
public:
    __fastcall TTestPlugin(void);
public: //IRecorderPlugin:
    // Создание плагина
    virtual bool STDMETHODCALLTYPE create(IRecorder* a_pOwner=NULL);
    // Конфигурирование
    virtual bool STDMETHODCALLTYPE config();
    // Вызов окна настройки
    virtual bool STDMETHODCALLTYPE edit();

    // Запуск
    virtual bool   STDMETHODCALLTYPE execute();
    // Приостановка работы
    virtual bool   STDMETHODCALLTYPE suspend();
    // Возобновление работы
    virtual bool   STDMETHODCALLTYPE resume();
    // Уведомление о внешних событиях
    virtual bool   STDMETHODCALLTYPE notify(DWORD a_dwCommand, DWORD a_dwData=0);
    // Получение имени
```

```
    virtual LPCSTR STDMETHODCALLTYPE getname();
    // Получить свойство
    virtual bool   STDMETHODCALLTYPE getproperty(DWORD a_dwPropertyID, VARIANT&
a_Value);
    // Задать свойство
    virtual bool   STDMETHODCALLTYPE setproperty(DWORD a_dwPropertyID, VARIANT
a_Value);
    // Узнать можно ли завершить работу плагина
    virtual bool   STDMETHODCALLTYPE canclose();
    // Завершить работу плагина
    virtual bool   STDMETHODCALLTYPE close();

private: //IUnknown
    DWORD FRefCount;
public:
    virtual HRESULT STDMETHODCALLTYPE QueryInterface(
        /* [in] */ REFIID riid,
        /* [iid_is][out] */ void __RPC_FAR *__RPC_FAR *ppvObject);

    virtual ULONG STDMETHODCALLTYPE AddRef( void);

    virtual ULONG STDMETHODCALLTYPE Release( void);
};
//---------------------------------------------------------------------------
extern PLUGININFO tstinfo;
//---------------------------------------------------------------------------

#endif
```

B.1.2 Модуль "TestPlugin.cpp"

```
/*-------------------------------------------------------------------------*/
/* Проект (Модуль) реслизации plug-in`a для измерительного ПО Recorder */
/* Модуль класса тестового plug-in`a, реализация                           */
/* Компилятор: Borland C++ Builder 5.0                                     */
/* НПП "ООО Мера" 2004г.                                                   */
/*-------------------------------------------------------------------------*/

#include <system.hpp>
#include "TestPlugin.h"

__fastcall TTestPlugin::TTestPlugin()
{
    FRefCount = 0;
}

PLUGININFO tstinfo = {
    "Builder C++ plug-in test",          //name[101]
    "The test plug-in for programmer guide.", //describe[201];
    "NPP \"Mera\"",                      //vendor[201];
    1,                                   //version;
    0                                    //subversion;
};

//---------------------------------------------------------------------------
//IRecorderPlugin:
// Создание плагина
bool STDMETHODCALLTYPE TTestPlugin::create(IRecorder* a_pOwner)
{
   return true;
}
// Конфигурирование
bool STDMETHODCALLTYPE TTestPlugin::config()
{
   return true;
}
// Вызов окна настройки
bool STDMETHODCALLTYPE TTestPlugin::edit()
{
   return true;
```

```
}

// Запуск
bool   STDMETHODCALLTYPE TTestPlugin::execute()
{
   return true;
}
// Приостановка работы
bool   STDMETHODCALLTYPE TTestPlugin::suspend()
{
   return true;
}
// Возобновление работы
bool   STDMETHODCALLTYPE TTestPlugin::resume()
{
   return true;
}
// Уведомление о внешних событиях
bool   STDMETHODCALLTYPE TTestPlugin::notify(DWORD a_dwCommand,
                                             DWORD a_dwData)
{
   return true;
}
// Получение имени
LPCSTR STDMETHODCALLTYPE TTestPlugin::getname()
{
   return tstinfo.name;
}
// Получить свойство
bool   STDMETHODCALLTYPE TTestPlugin::getproperty(DWORD a_dwPropertyID,
                                                  VARIANT& a_Value)
{
    switch (a_dwPropertyID)
    {
    case PLGPROP_INFOSTRING:     //Получить строку описания plug-in`a
       VariantCopy( &a_Value, &Variant(tstinfo.describe).operator VARIANT());
       return true;
    default:
       return false;
    }
}
// Задать свойство
bool   STDMETHODCALLTYPE TTestPlugin::setproperty(DWORD a_dwPropertyID,
                                                  VARIANT a_Value)
{
   return false;
}
// Узнать можно ли завершить работу плагина
bool   STDMETHODCALLTYPE TTestPlugin::canclose()
{
   return true;
}
// Завершить работу плагина
bool   STDMETHODCALLTYPE TTestPlugin::close()
{
   return true;
}

//------------------------------------------------------------------------------

HRESULT STDMETHODCALLTYPE TTestPlugin::QueryInterface(
        /* [in] */ REFIID riid,
        /* [iid_is][out] */ void __RPC_FAR *__RPC_FAR *ppvObject)
{
    if (!ppvObject) return E_FAIL;
    *ppvObject = NULL;

    if (riid == IID_IUnknown)
        *ppvObject = this;

    else if (riid == IID_IRecorderPlugin)
```

```
        *ppvObject = static_cast<IRecorderPlugin*> (this);

    if (*ppvObject)
    {
       reinterpret_cast<IUnknown*>(*ppvObject)->AddRef();
       return S_OK;
    }
    else
       return E_NOINTERFACE;
}

ULONG STDMETHODCALLTYPE TTestPlugin::AddRef( void)
{
    return ++ FRefCount;
}

ULONG STDMETHODCALLTYPE TTestPlugin::Release( void)
{
   DWORD tmp = --FRefCount;
   /*
   if (tmp == 0)
      delete this;
   */
   return tmp;
}
```

B.1.3 Модуль проекта библиотеки "ctest.cpp"

```
/*-------------------------------------------------------------------------*/
/* Проект (Модуль) реслизации plug-in`a для измерительного ПО Recorder */
/* Модуль динамически линкуемой библиотеки тестового plug-in`a         */
/* Компилятор: Borland C++ Builder 5.0                                     */
/* НПП "ООО Мера" 2004г.                                                   */
/*-------------------------------------------------------------------------*/

#include <windows.h>
#include <string.h>
#include <rcplugin.h>

#include "TestPlugin.h"

#pragma argsused
int WINAPI DllEntryPoint(HINSTANCE hinst, unsigned long reason, void* lpReserved)
{
        return 1;
}

//---------------------------------------------------------------------------
//Один глобальный статический объект plug-in`a.
TTestPlugin GPlugin;

//---------------------------------------------------------------------------
//Перечень экспортируемых модулем функций, функции объявлены как
//экспортируемые "_export", формат вызова "_cdecl", обрамление
//имен функций как для языка "C".
extern "C" {

//Получение типа класса plug-in`a
int _export _cdecl GetPluginType(void)
{
    //Единственное значение, которое используется в данный момент
    return PLUGIN_CLASS;
}

//Метод (создания) получения ссылки на объект plug-in`a
IRecorderPlugin * _export _cdecl CreatePluginClass(void)
{
    //В данном модле plug-in`a используется всего один экземпляр класса
```

```
    //plug-in`a, причем он объявлен как статическая переменная. Таким
    //образом данная функция возвращает только ссылку на объект plug-in`a.
    return &GPlugin;
}

//Метод (удаления) освобождения ссылки на объект plug-in`a
int _export _cdecl DestroyPluginClass(IRecorderPlugin* a_piPlg)
{
    return 0;
}

//Метод получения строки описания plug-in`a
const char* _export _cdecl GetPluginDescription(void)
{
    return tstinfo.describe;
}


//Метод получения получения расширенной описательной информации о plug-in`e
void _export _cdecl GetPluginInfo(LPPLUGININFO lpPluginInfo)
{
    if(lpPluginInfo)
    {
        //В структуру (параметр) копируются строки описания...
        strcpy(lpPluginInfo->name,     tstinfo.name);
        strcpy(lpPluginInfo->describe, tstinfo.describe);
        strcpy(lpPluginInfo->vendor,   tstinfo.vendor);
        //...и номера версий...
        lpPluginInfo->version=     tstinfo.version;
        lpPluginInfo->subversion=  tstinfo.subversion;
    }
}

}
```

B.2 Plug-in с функциями обработки и отображения данных

B.2.1 Модуль "TestPlugin.h"

```
/*-------------------------------------------------------------------------*/
/* Проект (Модуль) реслизации plug-in`a для измерительного ПО Recorder */
/* Модуль класса тестового plug-in`a, описание                             */
/* Компилятор: Borland C++ Builder 5.0                                     */
/* НПП "ООО Мера" 2004г.                                                   */
/*-------------------------------------------------------------------------*/

#ifndef TestPluginH
#define TestPluginH

#include <vcl.h>
#include <windows.h>
#include <rcplugin.h>

#include "TestFormUnit.h"

//---------------------------------------------------------------------------
class TTestPlugin: public IRecorderPlugin
{
private:
   TTestForm * FTestForm;
public:
    __fastcall TTestPlugin(void);
public: //IRecorderPlugin:
    // Создание плагина
    virtual bool STDMETHODCALLTYPE create(IRecorder* a_pOwner=NULL);
    // Конфигурирование
    virtual bool STDMETHODCALLTYPE config();
```

```
    // Вызов окна настройки
    virtual bool STDMETHODCALLTYPE edit();

    // Запуск
    virtual bool   STDMETHODCALLTYPE execute();
    // Приостановка работы
    virtual bool   STDMETHODCALLTYPE suspend();
    // Возобновление работы
    virtual bool   STDMETHODCALLTYPE resume();
    // Уведомление о внешних событиях
    virtual bool   STDMETHODCALLTYPE notify(DWORD a_dwCommand,
                                            DWORD a_dwData=0);
    // Получение имени
    virtual LPCSTR STDMETHODCALLTYPE getname();
    // Получить свойство
    virtual bool   STDMETHODCALLTYPE getproperty(DWORD a_dwPropertyID,
                                                 VARIANT& a_Value);
    // Задать свойство
    virtual bool   STDMETHODCALLTYPE setproperty(DWORD a_dwPropertyID,
                                                 VARIANT a_Value);
    // Узнать можно ли завершить работу плагина
    virtual bool   STDMETHODCALLTYPE canclose();
    // Завершить работу плагина
    virtual bool   STDMETHODCALLTYPE close();

private: //IUnknown
    DWORD FRefCount;
public:
    virtual HRESULT STDMETHODCALLTYPE QueryInterface(
        /* [in] */ REFIID riid,
        /* [iid_is][out] */ void __RPC_FAR *__RPC_FAR *ppvObject);

    virtual ULONG STDMETHODCALLTYPE AddRef( void);

    virtual ULONG STDMETHODCALLTYPE Release( void);
};
//---------------------------------------------------------------------------
extern PLUGININFO tstinfo;
//---------------------------------------------------------------------------

#endif
```

B.2.2 Модуль "TestPlugin.cpp"

```
/*-------------------------------------------------------------------------*/
/* Проект (Модуль) реслизации plug-in`a для измерительного ПО Recorder */
/* Модуль класса тестового plug-in`a, реализация                           */
/* Компилятор: Borland C++ Builder 5.0                                     */
/* НПП "ООО Мера" 2004г.                                                   */
/*-------------------------------------------------------------------------*/

#include <system.hpp>
#include "TestPlugin.h"

__fastcall TTestPlugin::TTestPlugin()
{
    FRefCount = 0;
}

PLUGININFO tstinfo = {
    "Builder C++ plug-in test",               //name[101]
    "The test plug-in for programmer guide.", //describe[201];
    "NPP \"Mera\"",                           //vendor[201];
    1,                                        //version;
    0                                         //subversion;
};
```

```
//---------------------------------------------------------------------------
//IRecorderPlugin:
// Создание плагина
bool STDMETHODCALLTYPE TTestPlugin::create(IRecorder* a_pOwner)
{
   //создание формы, инициализация формы ссылкой на Recorder
   FTestForm = new TTestForm( a_pOwner);
   return true;
}
// Конфигурирование
bool STDMETHODCALLTYPE TTestPlugin::config()
{
   return true;
}
// Вызов окна настройки
bool STDMETHODCALLTYPE TTestPlugin::edit()
{
   FTestForm->Show();
   return true;
}

// Запуск
bool   STDMETHODCALLTYPE TTestPlugin::execute()
{
   FTestForm->Show();
   return true;
}
// Приостановка работы
bool   STDMETHODCALLTYPE TTestPlugin::suspend()
{
   return true;
}
// Возобновление работы
bool   STDMETHODCALLTYPE TTestPlugin::resume()
{
   return true;
}
// Уведомление о внешних событиях
bool   STDMETHODCALLTYPE TTestPlugin::notify(DWORD a_dwCommand,
                                             DWORD a_dwData)
{
    switch(a_dwCommand)
    {
    case PN_ENTERRCCONFIG: // перешел в режим настройки
      //В то время пока производится изменение настройки,
      //теги недоступны (потому, что они могут удаляться)}
      //сообщить фрме о начале изменения настройки
      return FTestForm->BeginConfigure();

    case PN_LEAVERCCONFIG:  // вышел из режима настройки
      //Настройка завершена, и необходимо оповестить форму
      //о завершении изменения конфигурации
      return FTestForm->EndConfigure();

    case PN_SHOWINFO:    //команда для отображения собственной формы
      FTestForm->Show(); // отображение формы
      return true;

    default: return false;
    }
}
// Получение имени
LPCSTR STDMETHODCALLTYPE TTestPlugin::getname()
{
   return tstinfo.name; //вернуть адрес строки наименования
}
// Получить свойство
bool   STDMETHODCALLTYPE TTestPlugin::getproperty(DWORD a_dwPropertyID,
                                                  VARIANT& a_Value)
{
    switch (a_dwPropertyID)
```

```
    {
    case PLGPROP_INFOSTRING:     //Получить строку описания plug-in`a
       //...строка tstinfo.describe преобразуется в Variant, используется
       //конструктор типа Variant, далее Variant преобразуется к типу
       //VARIANT, при помощи оператора, а потом создается копия
       //в переменную a_Value.
       VariantCopy( &a_Value,
                    &Variant(tstinfo.describe).operator VARIANT());
       return true;
    default:
       return false;
    }
}
// Задать свойство
bool   STDMETHODCALLTYPE TTestPlugin::setproperty(DWORD a_dwPropertyID,
                                                  VARIANT a_Value)
{
   return false;
}
// Узнать можно ли завершить работу плагина
bool   STDMETHODCALLTYPE TTestPlugin::canclose()
{
   //данный plug-in не имеет оснований (причин)
   //для запрета закрытия пирложения...
   return true;
}
// Завершить работу плагина
bool   STDMETHODCALLTYPE TTestPlugin::close()
{
   delete FTestForm; //удаление формы
   return true;
}

//------------------------------------------------------------------------------
//--------------------Реализация интерфейса IUnknown----------------------------
//------------------------------------------------------------------------------

//Запрос интерфейса
HRESULT STDMETHODCALLTYPE TTestPlugin::QueryInterface(
        /* [in] */ REFIID riid,
        /* [iid_is][out] */ void __RPC_FAR *__RPC_FAR *ppvObject)
{
    //Проверка аргемента и устновка ему значения по умолчанию
    if (!ppvObject) return E_FAIL;
    *ppvObject = NULL;

    //Класс plug-in`a реалищзует всего лишь один интерфейс кроме IUnknown
    if ((riid == IID_IUnknown) ||
        (riid == IID_IRecorderPlugin))
        *ppvObject = static_cast<IRecorderPlugin*> (this);

    //Если ссылка на интерфейс удачно получена, то...
    if (*ppvObject)
    {
       //...необходимо увеличить счетчик ссылок...
       reinterpret_cast<IUnknown*>(*ppvObject)->AddRef();
       return S_OK;
    }
    else
       return E_NOINTERFACE;
}

//Отладочная реализация...
ULONG STDMETHODCALLTYPE TTestPlugin::AddRef( void)
{
    return ++ FRefCount;  //...увеличиение счетчика ссылок
}

//Отладочная реализация...
ULONG STDMETHODCALLTYPE TTestPlugin::Release( void)
{
```

```
   //...уменьшение счетчика ссылок...
   DWORD tmp = --FRefCount;
   /* Удаление не производится, потому, что
      объект plug-in`a является статическим.
   if (tmp == 0)
      delete this;
   */
   return tmp;
}
```

B.2.3 Модуль "TestFormUnit.h"

```
/*-------------------------------------------------------------------------*/
/* Проект (Модуль) реслизации plug-in`a для измерительного ПО Recorder */
/* Модуль класса формы тестового plug-in`a, описание                       */
/* Компилятор: Borland C++ Builder 5.0                                     */
/* НПП "ООО Мера" 2004г.                                                   */
/*-------------------------------------------------------------------------*/

//---------------------------------------------------------------------------

#ifndef TestFormUnitH
#define TestFormUnitH
//---------------------------------------------------------------------------
#include <Classes.hpp>
#include <Controls.hpp>
#include <StdCtrls.hpp>
#include <Forms.hpp>
#include <ComCtrls.hpp>
#include <ExtCtrls.hpp>
#include <Graphics.hpp>
#include <ImgList.hpp>
#include <utilcls.h>
#include <vector>

#include <blaccess.h>
#include <rcplugin.h>

using namespace std;

//Эта форма реализует два различных способа получения данных из тегов.
//Првый способ - это получение оценки. Второй - получение вектора и
//вычисление среднего. Для каждого из методов описан свой собственный
//метод получения данных. Полученные данные всегда преобразуются в строку
//для отображения. Какой из методов будет испеользоваться завсисит от
//того определен ли идентификатор DATATHROUGHVECTOR. Если этот идентификатор
//определен, то используется метод получения вектора данных.
//Выбор используемого метода осуществляется директивами прекомпиляции
//#ifdef, #ifndef, #else, #endif...
#define DATATHROUGHVECTOR

//---------------------------------------------------------------------------
//Форма для тестового plufg-in`a, для отображения измеренных данных.
//Форма производит периодически получение данных у списка тегов и
//отображение данных в табличном виде...
class TTestForm : public TForm
{
__published: // IDE-managed Components
        TListView *TagsListView;
        TPanel *Panel1;
        TBevel *Bevel1;
        TBevel *Bevel2;
        TImage *LogoImage;
        TImageList *TagsImageList;
        TTimer *TimeTimer;
        TTimer *DataUpdateTimer;
        TLabel *Label1;
        TLabel *TimeLabel;
```

```
        void __fastcall TimeTimerTimer(TObject *Sender);
        void __fastcall DataUpdateTimerTimer(TObject *Sender);
private:
        //переменная для сохранения ссылки на Recorder
        TComInterface<IRecorder> FRecorder;

        //Тип для описания вектора тегов
        typedef vector<TComInterface<ITag> > tagslist;
        //переменная вектор тегов; перечень тегов,...
        tagslist FTags; //для которых данные отображаются

        //Подготока окна (таблицы) для отображения данных.
        //Устанавливается необходимое кол-во строк, отображаются
        //имена тегов...
        void __fastcall PrepareTagsView(void);

        //Обновление данных. Получение данных у списка тегов
        //и отображение данных в табличном виде...
        void __fastcall DataUpdate(void);

        //Объявление различных методов для получения данных.
        //Два разных способа получения данных: получение
        //оценки и получение вектора.
        //Какой из методов юудет определен, зависит от того
        //объявлен ли идентификатор DATATHROUGHVECTOR.
#ifdef DATATHROUGHVECTOR
        //Метод получения данных тега через вектор
        AnsiString __fastcall GetDataThroughVector(ITag * pTag,
                                        AnsiString &StrTimeStamp);
        //Внутренний метод для полчения штампа времени к данным.
        AnsiString __fastcall GetTimeStamp( IBlockAccess * pba,
                                        const int nblock);
#else
        //Метод получения данных тега через оценку.
        AnsiString __fastcall GetDataThroughAOD( ITag * pTag);
#endif

public:
        //Переопределенный конструктор, предназначен для того,
        //чтобы запомнить в форме ссылку на Recorder.
        __fastcall TTestForm(IRecorder* pRecorder);
public:
        //Метод для обработки начала изменения настройки.
        bool __fastcall BeginConfigure(void);
        //Метод для обработки завершения изменения настройки.
        bool __fastcall EndConfigure(void);
};
//---------------------------------------------------------------------------
#endif
```

B.2.4 Модуль "TestFormUnit.cpp"

```
/*-------------------------------------------------------------------------*/
/* Проект (Модуль) реслизации plug-in`а для измерительного ПО Recorder */
/* Модуль класса формы тестового plug-in`а, реализация                     */
/* Компилятор: Borland C++ Builder 5.0                                     */
/* НПП "ООО Мера" 2004г.                                                   */
/*-------------------------------------------------------------------------*/

#include <vcl.h>
#pragma hdrstop

#include <malloc.h>
#include "TestFormUnit.h"

//---------------------------------------------------------------------------
#pragma resource "*.dfm"

//---------------------------------------------------------------------------
```

```
//Переопределенный конструктор, предназначен для того,
//чтобы запомнить в форме ссылку на Recorder.
__fastcall TTestForm::TTestForm(IRecorder* pRecorder)
        : TForm( (Classes::TComponent*)0)
{
    FRecorder.Bind( pRecorder, true);
}
//---------------------------------------------------------------------------
void __fastcall TTestForm::TimeTimerTimer(TObject *Sender)
{
    TimeLabel->Caption = TimeToStr( Now());
}
//---------------------------------------------------------------------------
//Метод для обработки начала изменения настройки.
bool __fastcall TTestForm::BeginConfigure(void)
{
    TimeTimer->Enabled = false;
    return true;
}
//---------------------------------------------------------------------------
//Метод для обработки завершения изменения настройки.
bool __fastcall TTestForm::EndConfigure(void)
{
    //Очистка внутреннего списка тегов
    FTags.clear();

    //Получение (обновленного) списка тегов
    int count = FRecorder->GetTagsCount();
    for (int i = 0; i < count; i++)
    {
        //временная переменная...
        TComInterface<ITag> pp;
        //...для получения ссылки на тег
        pp.Bind( FRecorder->GetTagByIndex( i));
        //...и добавления ссылки в список
        FTags.push_back ( pp);
    }

    //Подготовка таблицы для отображениятегов
    PrepareTagsView();

    //Продолжить режим обновления данных
    TimeTimer->Enabled = true;

    return true;
}

//---------------------------------------------------------------------------
void __fastcall TTestForm::DataUpdateTimerTimer(TObject *Sender)
{
    //Обновление данных. Получение данных у списка тегов
    //и отображение данных в табличном виде...
    DataUpdate();
}

//---------------------------------------------------------------------------
//Метод обновления отображения измеренных данных.
//Обновление данных. Получение данных у списка тегов
//и отображение данных в табличном виде...
void __fastcall TTestForm::DataUpdate(void)
{
    int index = 0;
    //цикл по всему списку тегов
    tagslist::iterator i = FTags.begin();
    for (; i != FTags.end(); i ++, index++)
    {
        //проверка корректности таблицы TagsListView и
        //корректности ссылки на тег
        if (TagsListView->Items->Count > index)
        {
          //получение ссылки на элемент таблицы
```

```
          TListItem * pil= TagsListView->Items->Item[index];

#ifdef DATATHROUGHVECTOR //получить блок из буфера
          //получение строкового представления измеренных
          //данных тега и штампа времени
          AnsiString StrTimeStamp;
          AnsiString StrData = GetDataThroughVector( *i, StrTimeStamp);

#else //получить обработанные данные (из оценки)
          AnsiString StrTimeStamp = TimeToStr( Now());
          AnsiString StrData = GetDataThroughAOD( *i);

#endif
          //установка данных в таблицу для отображения
          if (pil->SubItems->Count >= 2)
             pil->SubItems->Strings[1] = StrData;
          if (pil->SubItems->Count >= 3)
             pil->SubItems->Strings[2] = StrTimeStamp;
        }
    }
}

#ifdef DATATHROUGHVECTOR
//Метод получения у тега измеренных данных в виде строки.
//Получение производится через буфер по блокам.
AnsiString __fastcall TTestForm::GetDataThroughVector(ITag * pTag,
                                        AnsiString &StrTimeStamp)
{
    AnsiString Result;
    //получить по ссылке на тег, ссылку на
    //интерфейс блочного доступа
    TComInterface<IBlockAccess> pba;
    pTag->QueryInterface( IID_IBlockAccess, reinterpret_cast<void**>(&pba));
    //блокировать буфер с измеренными данными, это обязательное
    //действие; так как буфер постоянно пополняется измеренными
    //данными - необходимо остановить изменение вектора на
    //время отображения данных.
    pba->LockVector();
    try
    {
        //получить размер блока
        int bl_size=  pba->GetBlocksSize();
        //получить кол-во блоков
        int bl_count= pba->GetBlocksCount();
        //проверка
        if ((bl_count != 0) && (bl_size != 0))
        {
            //выделяем память для чтения блока....
            double * pdata =
              reinterpret_cast<double *>(alloca( sizeof(double) * bl_size));

            //чтение последнего блока измеренных данных
            pba->GetVectorR8( pdata, bl_count - 1, bl_size, true);

            //вычисление среднего арифметического
            double aod= 0;
            for (int i= 0; i < bl_size; i++)
              aod = aod + pdata[i];
            aod= aod / bl_size;

            //получить временной штам текущих данных
            StrTimeStamp= GetTimeStamp( pba, bl_count - 1);
            //формирование строки со значением
            Result = FloatToStr( aod);
        }
    }
    __finally
    {
        //разблокировать вектор
        pba->UnlockVector();
    }
```

```
    return Result;
}
#endif

#ifdef DATATHROUGHVECTOR
//Получение временного штампа для блока данных с номером nblock,
//доступ к блоку данных осуществляется по средствам IBlock
AnsiString __fastcall TTestForm::GetTimeStamp( IBlockAccess * pba,
                                               const int nblock)
{
    union FTR
    {
        __int64 itime;
        FILETIME ftime;
    };

    double CTS;
    FTR temp_time;
    SYSTEMTIME systime; //преобразовать время в необходимый формат

    CTS = pba->GetBlockUTSTime( nblock);
    try
    {
        CTS = CTS * 1000;
        temp_time.itime = CTS;
        temp_time.itime = temp_time.itime * 10000;
        FileTimeToSystemTime( &temp_time.ftime, &systime);
        //и в строку для отображения в таблице
        return TimeToStr (SystemTimeToDateTime( systime));
    }
    catch(...)
    {}
    return "";
}
#endif

#ifndef  DATATHROUGHVECTOR
//---------------------------------------------------------------------------
//Внутренний метод для полчения штампа времени к данным.
AnsiString __fastcall TTestForm::GetDataThroughAOD( ITag * pTag)
{
    Variant vData;

    //Получение среднего арифметического как одного из свойств тега
    if ((pTag != 0) &&
        (pTag->GetProperty(TAGPROP_ESTIMATE, *vData.operator VARIANT *())))
       return vData;
    else
       return "";
}
#endif
//---------------------------------------------------------------------------

//Метод подготовки формы после изменения списка тегов.
//Подготока окна (таблицы) для отображения данных.
//Устанавливается необходимое кол-во строк, отображаются
//имена тегов...
void __fastcall TTestForm::PrepareTagsView(void)
{
    //Список тегов на форме очистить
    TagsListView->Items->Clear();
    //и по всему списку ссылок на теги...
    tagslist::iterator i = FTags.begin();
    for (; i != FTags.end(); i ++)
    {
        //добавить элемент в список отображения
        TListItem* pil = TagsListView->Items->Add();
        //имя тега
        pil->Caption = (*i)->GetName();
        //строка описания тега
        Variant vDsc;
```

```
        if ((*i)->GetProperty( TAGPROP_DESCRIBE, vDsc))
            pil->SubItems->Add(vDsc);
        else
            pil->SubItems->Add("");
        //и оставить место для значения данных тега
        pil->SubItems->Add("");
        //и оставить место для значения временного штампа тега
        pil->SubItems->Add("");
    }
}
```

B.2.5 Модуль проекта библиотеки "test_viw.cpp"

```
/*-------------------------------------------------------------------------*/
/* Проект (Модуль) реслизации plug-in`a для измерительного ПО Recorder */
/* Модуль динамически линкуемой библиотеки тестового plug-in`a         */
/* Компилятор: Borland C++ Builder 5.0                                     */
/* НПП "ООО Мера" 2004г.                                                   */
/*-------------------------------------------------------------------------*/

#include <vcl.h>
#include <windows.h>
#pragma hdrstop

#include "TestPlugin.h"

#pragma argsused
int WINAPI DllEntryPoint(HINSTANCE hinst, unsigned long reason, void* lpReserved)
{
        return 1;
}

//---------------------------------------------------------------------------
//Один глобальный статический объект plug-in`a.
TTestPlugin GPlugin;

//---------------------------------------------------------------------------
//Перечень экспортируемых модулем функций, функции объявлены как
//экспортируемые "_export", формат вызова "_cdecl", обрамление
//имен функций как для языка "C".
extern "C" {

//Получение типа класса plug-in`a
int _export _cdecl GetPluginType(void)
{
    //Единственное значение, которое используется в данный момент
    return PLUGIN_CLASS;
}

//Метод (создания) получения ссылки на объект plug-in`a
IRecorderPlugin * _export _cdecl CreatePluginClass(void)
{
    //В данном модле plug-in`a используется всего один экземпляр класса
    //plug-in`a, причем он объявлен как статическая переменная. Таким
    //образом данная функция возвращает только ссылку на объект plug-in`a.
    return &GPlugin;
}

//Метод (удаления) освобождения ссылки на объект plug-in`a
int _export _cdecl DestroyPluginClass(IRecorderPlugin* a_piPlg)
{
    return 0;
}

//Метод получения строки описания plug-in`a
const char* _export _cdecl GetPluginDescription(void)
{
```

```
    return tstinfo.describe;
}


//Метод получения получения расширенной описательной информации о plug-in`e
void _export _cdecl GetPluginInfo(LPPLUGININFO lpPluginInfo)
{
    if(lpPluginInfo)
    {
        //В структуру (параметр) копируются строки описания...
        strcpy(lpPluginInfo->name,     tstinfo.name);
        strcpy(lpPluginInfo->describe, tstinfo.describe);
        strcpy(lpPluginInfo->vendor,   tstinfo.vendor);
        //...и номера версий...
        lpPluginInfo->version=     tstinfo.version;
        lpPluginInfo->subversion=  tstinfo.subversion;
    }
}

}
```

C Приложение. Распечатка кода программы plug-in`a, разработанного при помощи Microsoft® Visual C++6.0