EDISON. Центр разработки программного обеспечения
+7 (499) 500-14-94
http://www.edsd.ru
market@edsd.ru

ТРЕБОВАНИЯ
ПО ОФОРМЛЕНИЮ ИСХОДНЫХ ТЕКСТОВ
ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ НА DELPHI

ОГЛАВЛЕНИЕ

Требования по оформлению исходных текстов программного обеспечения на Delphi

1. ВВЕДЕНИЕ

1.1. Цель документа

Целью «Положения по оформлению исходных текстов на Object Pascal при выполнении проектов разработки программного обеспечения» (далее — Положения) является регламентация оформления исходных текстов на языке программирования Object Pascal при разработке ПО для обеспечения необходимого уровня разрабатываемого ПО и облегчения совместного использования исходных текстов.

1.2. Сфера применения

Положение предназначено для использования в области разработки программного обеспечения.

Положение является частью документационного обеспечения основной деятельности — разработки заказного программного обеспечения.

Утвержденное Положение имеет статус внутреннего стандарта и обязательно для исполнения в проектах разработки программного обеспечения.

1.3. Применяемость Положения

Настоящее положения необходимо применять ко всем новым продуктам. Существующие продукты необходимо приводить в соответствии с Положением по мере необходимости.

1.4. Процедура сопровождения Положения

Настоящее Положение разрабатывается руководителем проекта, согласуется на Техническом совете и затем утверждается.

Сопровождение, версионный контроль и доведение Положения до всех сотрудников осуществляет руководитель проекта. Номер версии присваивается в процессе ввода Положения в действие. Руководитель проекта осуществляет сбор предложений и замечаний, которые формируются в ходе выполнения проектов разработки программного обеспечения. При исправлении ошибок или несоответствий Положению присваивается следующий по порядку вспомогательный номер версии (после разделительной точки); при изменении и вводе в действие новых элементов организации или технологии работ новой версии Положения присваивается следующий по порядку основной номер. При вводе в действие новой версии Положение публикуется на сервере, сотрудники уведомляются о выпуске новой версии по электронной почте.

1.5. Термины, сокращения и определения

Сокращение, термин	Расшифровка сокращения или термина	Категория на английском языке
Разработчик	Ключевая роль в рабочей группе проекта разработки программного обеспечения, отвечающая за кодирование и отладку программного обеспечения	Developer
ПО	Программное Обеспечение	Software
Продукт	Результат, произведенный в ходе выполнения проекта	Product
Проект	Ограниченная во времени деятельность, направленная на разработку уникального продукта	Project
БД	База данных, используемая для хранения информации	Database

2. ФАЙЛЫ ИСХОДНОГО КОДА

2.1. Именование исходных файлов

Все файлы должны иметь смысловое название.

Расширения файлов должны быть в нижнем регистре.

Имя основного файла продукта не должно содержать префиксов и должно начинаться с заглавной буквы, например: **`Sample.dpr`**. Если при создании имени используется несколько слов, то необходимо использовать заглавную букву для каждого слова в имени, например: **`MySampleProject.dpr`**. Не допускается использовать символ подчеркивания для разделения нескольких слов в наименовании продукта. Исключением является продукт, разрабатываемый в качестве сервиса Windows NT, его наименование должно начинаться с префикса «**`AX_`**», например: **`AX_TestService.dpr`**.

Имя файлов модулей должно начинаться с префикса «**`un`**», за префиксом должно следовать смысловое название модуля с заглавной буквы, например: **`unSample.pas`**. Если при создании имени используется несколько слов, то необходимо использовать заглавную букву для каждого слова в имени, например: **`unMyName.pas`**. Для разделения слов не допускается использовать символ подчеркивания.

Имя файла включения должно соответствовать наименованию основного файла продукта, например: **`MySampleProject.inc`**.

Предопределенные префиксы для наименований файлов модулей:

модуль данных:	**`unDM`**
модуль реализации интерфейса ActiveX:	**`unI`**
форма конфигурирования настроек:	**`unSet`**
форма справочника БД:	**`unRef`**
модуль алгоритмизации взвешивания:	**`unAlg`**

Предопределенные наименования файлов для стандартных модулей:

главная форма:	**unMain.pas**
основной модуль данных:	**unDMMain.pas**
модуль работы с настройками:	**unSetup.pas**
модули, содержащие формы	**начинать с префикса f например fDialog.pas**
модули, содержащие фреймы:	**начинать с префикса frm например frmDialog.pas**

2.2. Организация исходных файлов

Все модули могут содержать следующие элементы в определенном порядке:

- Описание модуля (Description);
- Имя модуля (Unit Name);
- Объявление включаемых файлов (Include files);
- Секцию интерфейса (Interface section);
- Дополнительные определения (Additional defines);
- Объявление используемых модулей (Uses clause);
- Реализацию (Implementation);
- Объявление используемых модулей (Uses clause);
- Закрывающий оператор и точку (A closing and a period).

Модуль всегда начинается с описания, заключенного в символы **{Описание}**. Допускаются описания, состоящих из нескольких строк.

Для визуального разделения элементов между ними должна быть хотя бы одна пустая строка.

Дополнительные определения могут быть структурированы в любом порядке, но нужно соблюдать обязательные условия: в начале файла всегда имя модуля, затем условные директивы и определения, директивы компилятора и файлы включения, затем определение подключений.

Директивы компилятора не должны напрямую включаться в исходный код. Для этого нужно использовать определение включений и подключить глобальный для продукта файл с директивами компилятора: **{$I MySampleProject.inc}**. В случае необходимости можно напрямую переопределить глобальные директивы компилятора. Переопределяющие директивы должны быть обязательно документированы и использоваться только для локального переопределения.
Наименования используемых модулей должны быть записаны с сохранением регистра символов оригинального названия модуля, например: **Classes, SysUtils**. Список используемых модулей необходимо располагать на следующей строке после слова **uses**. Если используются модули из разных продуктов или разных производителей, то необходимо сгруппировать модули по продуктам или производителям и каждую новую группу начинать с новой строки и снабжать комментариями:

```
uses
  Windows, SysUtils, Classes,
  Graphics, Controls, Forms, TypInfo, // модули Delphi
  XXXMyUnit1, XXXMyUnit2; // модули XXX
```

Если список используемых модулей не умещается по ширине в 80 символов, то его необходимо перенести на следующую строку.
Для удобства при работе с системой контроля версий, при внесении изменений в список используемых модулей, каждая строка содержит только один модуль.

2.3. Файл проекта

При работе с любым проектом необходимо устанавливать свойство автоматического инкремента номера сборки.

3. ЗАРЕЗЕРВИРОВАННЫЕ СЛОВА

Стоимость выполненных работ (услуг) по настоящему Договору определяется стоимостью 1 (одного) нормо-часа и составляет 600 (шестьсот) рублей за 1 (один) нормо-час, в т.ч. НДС 18%.
Основанием для проведения взаимных расчетов между Исполнителем и Заказчиком являются надлежащим образом оформленные счета-фактуры и акты сдачи-приемки выполненных работ (услуг), подписанные представителями Сторон.
Количество подлежащих оплате нормо-часов определяется по фактическому времени выполнения работ (услуг) либо по предварительной договоренности при согласовании задания.
Изменение стоимости 1 (одного) нормо-часа допускается только в случае существенного изменения обстоятельств на рынке выполнения работ (услуг) по сопровождению программы «1С: Предприятие» на территории города Кемерово.
Изменение стоимости 1 (одного) нормо-часа оформляется Протоколом соглашения об установлении цены за 1 (один) нормо-час, подписанным обеими Сторонами, который является неотъемлемой частью настоящего Договора.
Оплата по настоящему Договору осуществляется Заказчиком путем перечисления денежных средств с его расчетного счета на расчетный счет Исполнителя.
Срок оплаты - в течение 5 (пяти) банковских дней с момента подписания акта сдачи-приемки выполненных работ (услуг).

4. НАИМЕНОВАНИЯ

4.1. Общие положения

Все наименования должны иметь смысловое значение. При составлении имен за основу берутся слова литературного английского языка.
Имя может начинаться с префикса (определяется для каждого конкретного случая в пунктах 4.1–4.7 данного Положения), за префиксом должно следовать смысловое название с заглавной буквы. Если при создании имени используется несколько слов, то необходимо использовать заглавную букву для каждого слова в имени. Для разделения слов допускается использовать символ подчеркивания.

Если слово имеет несколько синонимов (напр. «Заказчик» — **Client** и **Customer**), то при проектировании выбирается один из них, и на протяжении всего проекта для обозначения данного понятия всеми разработчиками обязан использоваться только он.

Не допускается смена регистра при обращении к одному и тому же имени. Т.е. при обращении имя обязано быть записано так же, как и при определении.

4.2. Стандартные компоненты

Наименование должно начинаться с префикса, идентифицирующего класс, например: **btSetParams**.

Не допускается использование компонентов сторонних разработчиков.

Соответствия префиксов и классов:

T(*)Command	**cmd**
T(*)Connection	**cn**
T(*)Query	**qry**
T(*)Table	**tb**
TBevel	**bv**
TBitBtn	**bbt**
TButton	**bt**
TCheckBox	**ckb**
TComboBox	**cb**
TDatabase	**db**
TDataSource	**ds**
TDBCheckBox	**dbckb**
TDBComboBox	**dbcb**
TDBEdit	**dbed**
TDBGrid	**dbg**
TDBListBox	**dblb**
TDBMemo	**dbmm**
TDBNavigator	**dbnv**
TDBText	**dbtx**
TDrawGrid	**dg**
TEdit	**ed**
TGroupBox	**gb**
THeader	**hd**
TImage	**im**
TLabel	**lbl**
TListBox	**lb**
TListView	**lv**
TMainMenu	**mme**
TMaskEdit	**med**
TMemo	**mm**
TMenuItem	**mi**
TNoteBook	**nb**
TOutline	**ol**
TPanel	**pn**
TPopupMenu	**pm**
TProgressBar	**pb**
TRadioButton	**rb**
TRadioGroup	**rg**
TScrollbar	**sb**
TScrollBox	**sb**
TShape	**sh**
TSpeedButton	**sbt**

TSpinEdit	**se**
TStaticText	**st**
TStatusBar	**stb**
TStringGrid	**sg**
TTabbedNotebook	**tnb**
TTabSet	**ts**
TTimer	**tm**
TTrackBar	**tb**
TTreeView	**tv**

4.3. Пользовательские типы

Наименование типа должно начинаться с префикса **T**.

Для элементов перечисляемых типов используются двух- или трехбуквенные префиксы. Данные префиксы определяют назначение элементов и состоят из букв, входящих в название типа (**TMessageType = (mtError, mtInfo)**).

При определении перечисления имя типа должно состоять из наименования базового типа с добавлением суффикса **Set**, например: **TMessageTypeSet**.

4.4. Классы

Наименование класса должно начинаться с префикса **T**. Исключением являются классы исключения, которые должны начинаться с префикса **E**.

Данные всегда должны располагаться только в приватной секции, и названия переменных должны всегда начинаться с префикса **F**.

Имена процедур для установки свойств должны состоять из префикса **Set** и имени свойства. Имена процедур для чтения свойств должны состоять из префикса **Get** и имени свойства.

Свойства логического типа должны состоять из префикса **Is и имени свойства**, если они используются только для чтения, например: **IsVisible**.

При использовании метода класса необходимо указывать процедурные скобки, даже если метод не содержит параметров, например: **GetIsVisible()**.

Все определяемые события должны иметь префикс **On**. При этом тип процедуры события должен формироваться в виде **TNotifyEvent**, **TCloseEvent**, **TOpenEvent** и т.п.

Все объявления внутри класса должны быть расположены либо по смысловым группам, либо по алфавиту.

Предопределенные префиксы для классов:

формы	**Tfm**
фреймы	**Tfrm**
модули данных	**Tdm**
форма конфигурирования настроек	**TfmSet**
форма справочника БД	**TfmRef**
класс работы с настройками	**TSetup**

Предопределенные наименования классов:

главная форма	**TfmMain**
основной модуль данных	**TdmMain**

4.5. Интерфейсы

Наименование должно начинается с префикса **I**.

Остальные требования аналогичны пункту 4.3 данного Положения.

4.6. Процедуры и функции

Наименование не должно содержать префикса.

Наименование должно выражать действие.

Наименование параметров функции должно начинаться с префикса **A**. Остальные требования к именованию параметров аналогичны пункту 4.7 данного Положения.

При использовании процедур и функций необходимо указывать процедурные скобки, даже если метод не содержит параметров, например: `IsVisible()`.

4.7. Переменные

Наименование глобальной переменной должно начинаться с префикса **glvar**. Глобальные переменные использовать ? как?????

Каждая переменная должна объявляться на новой строке. Пример:

```
var
Count: Integer;
MaxValue,
MinValue: LongWord;
```

Возможно использовать несмысловые названия в следующих случаях:

- если переменная используется только как счетчик цикла, то в качестве названия могут использоваться буквы: **i, j, k, m, n**.
- если переменная используется только для индексации массива, то в качестве названия могут использоваться буквы: **i, j, k, m, n**.

Предопределенные элементы в названиях переменных:

счетчик кол-ва элементов	слово	**Count**
имя файла	слово	**FileName**
максимальное значение	префикс	**Max**
минимальное значение	префикс	**Min**
номер элемента	префикс	**Num**
номер элемента	суффикс	**No**
ключ для поиска значений	префикс	**Id**
первый элемент	префикс	**First**
последний элемент	префикс	**Last**
текущий элемент	префикс	**Current**
следующий элемент	префикс	**Next**
предыдущий элемент	префикс	**Prev**

4.8. Константы

Наименования должны быть в верхнем регистре, и для разделения слов должен использоваться только символ подчеркивания.

Остальные требования аналогичны пункту 4.6 данного руководства.

Строковые константы необходимо группировать и выносить в секцию `resourcestring`.

5. ЗАПИСЬ ОПЕРАТОРОВ И ВЫРАЖЕНИЙ

Данный раздел содержит типовые конструкции операторов и выражений. Использование других вариантов конструкций недопустимо.

5.1. Общие положения

Каждый оператор должен заканчиваться точкой с запятой.

Если оператор содержит блок более 3-х размеров экрана, то закрывающая операторная скобка должна иметь поясняющий комментарий, указывающий, какой оператор закрывается.

Реализация набора, состоящего из 4-х и более операторов, повторяющихся 2 и более раз, должна быть оформлена в виде отдельной процедуры или функции.

Не рекомендуется использование вложенных процедур.

Каждый простой оператор должен располагаться на одной строке, в случае необходимости длинный оператор можно перенести на новую строку, новая строка должна начинаться с отступа в два пробела. Если выражение входит в конструкцию, требующую отступа на следующей строке, то при переносе выражения использовать

двойной отступ. Если переносится выражение, то новая строка должна начинаться со знака операции. Например:

```
if (A and B
    and C and D) then

MyValue := MyValue
    + (SomeVeryLongStatement / OtherLongStatement);
```

В выражении каждое отдельное действие должно заключаться в скобки. Само выражение также должно заключаться в скобки. Не рекомендуется использование длинных выражений. Если необходимо записать длинное выражение, то рекомендуется использование вспомогательных переменных. Например:

```
MyValue := ((i < 7) and ((j < 8) or (k = 4)));

MyValue := (SomeStatement / OtherStatement);
```

Если вместо простого оператора используется составной, то операторные скобки могут иметь тот же отступ, что и простой оператор, вместо которого используется составной. А все операторы, заключенные в эти операторные скобки, должны иметь дополнительный отступ.

5.2. Оператор if

Простой оператор:

```
if (Выражение) then
    Оператор1;

if (Выражение) then
    Оператор1
else
    Оператор2;
```

Сложный оператор:

```
if (Выражение) then
begin
    Оператор1;
    Оператор2;
    ...
end;

if (Выражение) then
begin
    Оператор1;
    Оператор2;
    ...
end
else
begin
    Оператор3;
    Оператор4;
    ...
end;
```

5.3. Оператор for

Простой оператор:

```
for Счетчик_Цикла := Выражение to(downto) Выражение do
    Оператор1;
```

Сложный оператор:

```
for Счетчик_Цикла := Выражение to(downto) Выражение do
begin
    Оператор1;
    Оператор2;
    ...
end;
```

5.4. Оператор while

Простой оператор:

```
while (Выражение) do
    Оператор1;
```

Сложный оператор:

```
while (Выражение) do
begin
    Оператор1;
    Оператор2;
    ...
end;
```

5.5. Оператор repeat until

Простой оператор:

```
repeat
    Оператор1;
until (Выражение);
```

Сложный оператор:

```
repeat
    Оператор1;
    Оператор2;
    ...
until (Выражение);
```

5.6. Оператор case

Простой оператор:

```
case (Выражение) of
    Выражение1: Оператор1;
    Выражение2: Оператор2;
    ...
else
    Оператор1;
end;
```

Сложный оператор:

```
case (Выражение) of
    Выражение1:
      begin
        Оператор1_1;
        Оператор1_2;
        ...
```

```
        end;
    Выражение2:
        begin
            Оператор2_1;
            Оператор2_2;
            ...
        end;
    ...
else
    Оператор1;
    Оператор2;
    ...
end;
```

5.7. Оператор with

Простой оператор:

```
with Список_объектов do
    Оператор1;
```

Сложный оператор:

```
with Список_объектов do
begin
    Оператор1;
    Оператор2;
    ...
end;
```

5.8. Оператор try

Простой оператор:

```
try
    Оператор1;
    Оператор2;
    ...
except
    Оператор3;
    ...
end;
```

Сложный оператор:

```
try
    Оператор1;
    Оператор2;
    ...
except
    on Класс_Исключения1 do Оператор1;
    on Класс_Исключения2 do
        begin
            Оператор1;
            Оператор2;
            ...
        end;
end;
```

6. КОММЕНТАРИИ

Для тела метода необходимо наличие комментариев, достаточных для понимания логики работы кода; не менее одной строки-комментария на три строки исходного кода.

Необходимо наличие комментария для всех объявлений в интерфейсной части.

Для процедур и функций и методов класса необходимо краткое описание выполняемого действия, а также краткое описание смысла передаваемых параметров и возвращаемого результата.

Допускается отсутствие комментариев для приватных членов класса.

В интерфейсной части модуля каждая группа функций, относящихся к одной подкатегории, должна отделяться от другой группы функций тремя строками блочного комментария шириной 80 символов с описанием подкатегории.

В секции реализации каждая подкатегория или класс должны отделяться строкой, состоящей из символов равенства (**=**), закомментированных однострочным комментарием и пустой строкой до и после группы функций. Длина строки должна составлять 80 символов.

Каждая функция из одной группы или методы класса должны разделяться между собой строкой, состоящей из символов минуса (**-**), закомментированных однострочным комментарием. Длина строки должна составлять 80 символов.

7. ИСПОЛЬЗОВАНИЕ ПРОБЕЛОВ И ОТСТУПОВ

7.1. Использование пустых строк

Пустые строки должны использоваться в следующих местах:

- после декларации пакета;
- после секции импорта;
- между объявлениями классов;
- между объявлениями пользовательских типов;
- между реализациями методов;
- между интерфейсными секциями, которые логически связаны между собой.

7.2. Использование пробелов

Не допускается использовать вместо пробелов табуляцию.

Пробелы должны использоваться в следующих местах:

- до и после оператора присваивания;
- до и после арифметических операторов в выражениях;
- до и после логических операторов в выражениях;
- после запятой при перечислении параметров функции во время вызова функции;
- после точки с запятой при перечислении параметров функции во время объявления функции;
- после двоеточия при объявлении переменной, параметров функции и типа результата функции;
- до и после знака «=» при объявлении типов и констант;

Не допускается использование пробелов в следующих местах:

- между именем метода и открывающей скобкой;
- после открывающей скобки или перед закрывающей;
- после открывающей квадратной скобки [или перед закрывающей];
- перед точкой с запятой;
- перед двоеточием при объявлении переменной, параметров функции и типа результата функции;
- между унарным оператором и его операндом.

7.3. Использование отступов

Не допускается использовать для отступов символ табуляции.

Всегда необходимо использовать два пробела для всех уровней отступа.

Первым уровнем, для которого не нужно делать отступы, считаются зарезервированные слова **`program`**, **`unit`**, **`uses`**, **`type`**, **`interface`**, **`implementation`**, **`initialization`** и **`finalization`**. Первым уровнем считаются объявления функции в секции реализации, а также зарезервированные слова **`var`**, **`const`**, **`type`** и **`begin..end`**, входящие в реализацию функции. Во всех остальных случаях необходимо наличие отступа.

Код внутри блока **`begin..end`** должен располагаться на следующем уровне. Завершающая операторная скобка (**`end`**) должна находиться на том же уровне, что и открывающая операторная скобка (**`begin`**).
Операторы одного уровня вложенности должны начинаться с одинаковыми отступами.
При объявлении класса области видимости располагаются на одном уровне с идентификатором класса. Все данные внутри класса располагаются на следующем уровне.
Отступы для операторов описаны в пункте 6 данного Положения.

8. BORLAND DEVELOPER STUDIO

При работе в Borland Developer Studio необходимо документировать модель кода. Для того чтобы приступить, необходимо включить в проекте поддержку Model View — при переходе к Model View среда разработки задаст вопрос о включении этой поддержки.

Дополнительные свойства нужно создавать вручную в исходном тексте для дополнительного разъяснения тонкости работы.

Документировать нужно каждый элемент структуры модели. Общий принцип — чтобы каждый элемент, по которому генерируется гиперссылка, был документирован.

На каждый класс создается минимум один пример его использования с комментариями. Пример помещается в свойство **Example** (пример) этого класса.

Сложные методы классов также обязательно снабжать примером использования Example.

Свойства, обязательные к заполнению, для различных типов объектов:

Описание проекта в целом:

Название проекту давать такое, чтобы описывался общий смысл проекта.

Заполнить поле Description (описание).

Модуль (Unit):

Заполнить поле Description. Документировать NameSpace.

Класс, структура:

Заполнить поле Author (автор), Since (дата начала работы), Version (версия объекта), самостоятельно создать поле Description.

Переменная:

Заполнить поле Author, Since, при необходимости самостоятельно создать поле Description.

Метод, процедура, функция, конструктор, деструктор, свойство:

Заполнить поле Author, Since, Input (что получает на входе), Output (какой результат выдает функция), Precondition (состояние системы перед началом работы функции), Postcondition (состояние системы после работы функции), Semantic (какая работа выполняется функцией). Документировать Input и Output.

Прототип функции:

Заполнить поле Author, Since, Version, самостоятельно создать поле Description.

Константа, перечислимый тип:

Заполнить поле Author, Since, самостоятельно создать поле Description.

9. ДОПОЛНИТЕЛЬНЫЕ ТРЕБОВАНИЯ К КОДУ

- Прописывать параметры функций const или var.
- При вызове функций Windows API проверять успешность их выполнения, или обернуть в Win32Check, или использовать RaiseLastOSError.

• Использовать модульное тестирование для критически важных модулей.
• Во всех модулях, явно работающих с указателями, необходимо в заголовке использовать директиву компиляции {$T+}.
• При перекрытии виртуального метода (override) вызывать унаследованный метод (inherited). За исключением тех случаев, когда поведение унаследованного метода полностью заменяется (что бывает редко). И обязательно вызываем унаследованные конструкторы и деструкторы.
• Если какой бы то ни было член класса (пусть даже на этапе разработки) должен быть выведен из использования, не нужно его сразу удалять. Надо пометить его директивой "deprecated". Иначе у других разработчиков могут возникнуть проблемы с компиляцией в тот момент, когда нет времени на переработку кода. Со временем использования "deprecated" членов нужно избегать.
• В функциях, являющихся обработчиками межплагинных событий или являющихся сервисными функциями плагина или менеджера, необходимо в подавляющем большинстве случаев перенаправлять вызов в некий объект, который представляет соответственно плагин или менеджер. Кроме того, этот вызов должен быть обрамлен конструкцией try...except, и в блоке except результат функции должен быть определен как ошибка.
• В классах секции private и protected использовать только с директивой strict. Таким образом избегаем видимости извне членов класса, не рассчитанных на это.
• При объявлении типа записи для нового параметра необходимо использовать packet record, иначе размер полученной структуры будет зависеть от директив компиляции, касающихся выравнивания. Это может привести к некорректности обработки параметра.
• Основная идея Synchronize — выполнить код в контексте главного потока приложения. Для этого отправляется сообщение в главный цикл обработки сообщений. Там, в обработчике TApplication.Idle (const Msg: TMsg), и происходит выполнение синхронной процедуры.
• Метод TThread.Synchronize выполняется синхронно для вызвавшего потока. Т.е. поток, вызвавший Synchronize, не получит управление до тех пор, пока синхронная процедура не будет выполнена в основном потоке. Предположим, что в главном потоке приложения было принято решение прекратить выполнение дополнительного потока. Для этого в потоке выставлен флаг Terminated и делается ожидание (WaitFor). А в потоке в этот момент вызван Sinchronize. Произойдет Deadlock — главный поток ждет завершения вторичного потока, а тот, в свою очередь, ждет, пока главный поток выполнит синхронную процедуру. Исключительные случаи, когда можно использовать Synchronize: если жизнь вторичного потока не бесконечна и главному потоку никогда не придется останавливать его, то вызовы Synchronize могут быть безопасными.
• Для ОС XP, 2003, Vista необходимо:
  - Для упрощенного (не прозрачного) десктопа установить флаг WS_EX_COMPOSITED. Все контролы будут работать нормально.
  - Для Vista с Aero WS_EX_COMPOSITED не работает, его устанавливать нельзя, т.к. из-за него пропадает при каких-то условиях прозрачность окна.
  - Выставить для всех оконных контролов флаг DubbleBuffered = true. Отказаться от использования неверно работающих контролов — PageControl, StaticText (список неполный). Для Panel Caption = '', FullRepaint = true;
  - Для ScrollBox возможно использовать промежуточную подложку между отображаемым в ScrollBox'е содержимым. Например: ScrollBox — FlowPanel — Buttons0-N. И для FlowPanel установить флаг WS_EX_COMPOSITED, это позволит передвигать содержимое за полосы прокрутки плавно, без рывков.
• TLabel и TStaticText не использовать для отображения меняющегося во времени выполнения текста. Вместо этого использовать TEditBox.
• Для загрузки BMP картинок из ресурсов следует использовать функцию LoadImage вместо LoadBitmap, так как при загрузке битмапки LoadBitmap автоматически делает преобразование в устройство-зависимый формат. Т.е. при установленной на рабочем столе глубине цвета в 16 бит произойдет преобразование картинки из 32-битной, имеющей альфа-канал, в 16-битную без альфа-канала. Что приведет к неправильному отображению на экране при выводе картинки с использованием функции AlphaBlend.

Для корректной загрузки следует пользоваться функцией LoadImage, с указанием флага LR_CREATEDIBSECTION.

Например:

```
LoadImage(SysInit.HInstance ,'PresentationCounterIcon',
 IMAGE_BITMAP, 0, 0,  LR_CREATEDIBSECTION);
```

10. ЛИСТ ОЗНАКОМЛЕНИЯ

ФИО сотрудника	Дата	Подпись