

Глава 23. Буфер обмена	559
23.1. Буфер обмена и стандартные компоненты Delphi	560
23.2 Объект Clipboard	561
23.3 Картинки и буфер обмена.	562
23.4 Создание собственного формата для работы с буфером.....	566



Глава 23. Буфер обмена

Кнопки «Копировать» и «Вставить» есть практически в любом полноценном приложении. Я думаю, что ты тоже захочешь вставить такую возможность в свои программы. В этой главе я дам максимум полезной теоретической и практической информации, чтобы ты смог сделать свои программы более привлекательными, добавив возможность переноса данных между приложениями.

Если в твоей программе есть строки ввода *TEdit*, то они уже умеют работать с буфером обмена. Попробуй щёлкнуть в любой такой строке правой кнопкой и перед тобой откроется меню, в котором есть все необходимые пункты. ОС Windows очень много делает за нас и нам не приходится заботиться о таких мелочах.

С другими компонентами Windows дело обстоит немного сложнее. Тут нам придётся немного поработать на клавиатуре. Но эта работа не так уж сложна и ты убедишься в этом, когда закончишь чтение этой главы.

23.1. Буфер обмена и стандартные компоненты Delphi

Большинство компонентов Delphi уже готовы к работе с буфером обмена. В основном это касается тех компонентов, которые содержат какие-либо данные, которые пользователь может захотеть поместить в буфер обмена. Если такая возможность нужна, то у компонента будут методы с именами:

1. *CutToClipboard* – вырезать в буфер обмена;
2. *CopyToClipboard* – копировать в буфер обмена;
3. *PasteFromClipboard* – вставить из буфера обмена.

Давай посмотрим на эти метода в действии. Создай новое приложение, и помести на его форму три кнопки: «Вырезать», «Копировать» и «Вставить». По центру окна растяни компонент *TMemo*. Мою форму ты можешь увидеть на рисунке 23.1.1.

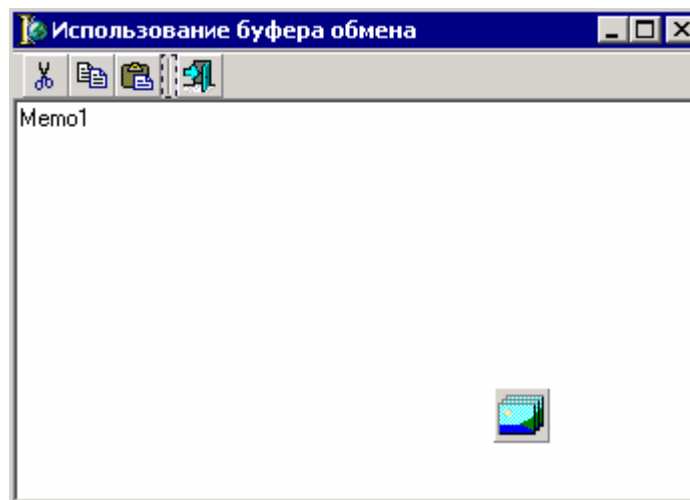


Рисунок 23.1.1 Форма будущей программы

Теперь создадим обработчики событий для кнопок. Для кнопки «Вырезать» напиши следующий код:

```
procedure TForm1.ToolButton1Click(Sender: TObject);
begin
  Memo1.CutToClipboard;
end;
```

Здесь всего лишь вызывается один метод *CutToClipboard* компонента *Memo1*. Этот метод вырежет выделенный текст в буфер обмена.

По событию *OnClick* для кнопки «Копировать» пишем следующий код:


```
procedure TForm1.ToolButton2Click(Sender: TObject);
begin
  Memo1.CopyToClipboard;
end;
```

Здесь происходит копирование выделенного фрагмента текста в буфер обмена с помощью метода *CopyToClipboard*.

Ну и для кнопки вставить пишем вызов метода *PasteFromClipboard* компонента *Memo1*. С помощью этого метода, всё содержимое буфера обмена будет вставлено в компонент *Memo1*.

Теперь попробуй запустить приложение и скопировать какой-нибудь текст в буфер обмена с помощью нашей кнопки «Копировать». После этого нажми на кнопку «Вставить» и скопированный текст появится в текущей позиции курсора. Если ты скопируешь в буфер какое-нибудь изображение из любого графического редактора и попытаешься вставить его в компонент *Memo1*, то ничего не произойдёт. Система сама следит за форматом данных, хранимых в буфере обмена. Чуть позже я покажу, как сделать так, чтобы кнопка «Вставить» была активна только тогда, когда в буфере есть данные и они соответствуют нужному формату.

Попробуй щёлкнуть правой кнопкой мыши внутри компонента *Memo1*. Перед тобой так же должно появиться всплывающее меню с необходимыми для работы с буфером обмена пунктами.

 На компакт диске, в директории \Примеры\Глава 23\Memo Clipboard ты можешь увидеть пример этой программы.

23.2 Объект Clipboard

Для работы с буфером обмена в Delphi есть объект *Clipboard*. Несмотря на то, что это объект, его не надо инициализировать (как и *TApplication* и *TPrinter*), а можно использовать без всяких инициализаций. Достаточно только подключить в разделе **uses** модуль *Clipbrd* и он становится тебе доступным.

У этого объекта не так уж и много свойств и методов, так что будем их рассматривать на практике. Для начала модернизируем пример, написанный в прошлой части с использованием этого объекта. Открой предыдущий пример и сразу добавь в раздел **uses** модуль *Clipbrd*. Теперь по событию *OnClick* для кнопки «Копировать» напиши следующий код:

```
Clipboard.SetTextBuf(PChar(Memo1.SelText));
```

Здесь я использую метод *SetTextBuf* объекта *Clipboard*. Этот метод копирует переданный в качестве параметра текст в буфер обмена. В качестве этого параметра я передаю выделенный в компоненте *Memo1* текст *Memo1.SelText*. Единственное – это надо привести текст к типу *PChar*.

По нажатию кнопки «Вырезать» пишем следующий код:

```
Clipboard.SetTextBuf(PChar(Memo1.SelText));  
Memo1.SelText:="";
```

«Вырезать» - значит скопировать текст в буфер и потом удалить его из компонента *Memo1*. Именно это я и делаю. В первой строке я написал код, который ты видел в обработчике события *OnClick* для кнопки «Копировать». Во второй строке я обнуляю (удаляю) выделенный текст.

По нажатию кнопки «Вставить» пишем следующий код:

```
Memo1.SelText:=Memo1.SelText+Clipboard.AsText;
```

Свойство *AsText* объекта *Clipboard* указывает на содержимое буфера обмена в виде текста. В этом коде я прибавляю это содержимое к выделенному тексту и вставляю это всё в выделенный текст. Вот таким нехитрым способом я реализовал вставку.

Попробуй запустить пример и убедиться, что всё работает. Как видишь, использование объекта *Clipboard* не намного сложнее и бояться тут нечего.

Теперь посмотрим ещё несколько интересных методов объекта *Clipboard*:

Assign – назначить в буфер обмена объект совместимый с *TPersistent*. К таким объектам относятся картинки *TImage*. В качестве единственного параметра нужно указать объект, содержимое которого нужно скопировать в буфер обмена.

Clear – очистить содержимое буфера обмена.

HasFormat – проверка, какого типа данные хранятся в буфере обмена. Возможны следующие типы данных:

CF_TEXT - буфер содержит текст.

CF_BITMAP - буфер содержит картинку.

CF_METAFILEPICT - буфер содержит векторную картинку.

CF_PICTURE - буфер содержит объект типа *TPicture*.


CF_COMPONENT - буфер содержит компонент.

Я добавил в наш пример одну кнопку, по нажатию которой будет выводиться сообщение, показывающее тип данных хранящихся в буфере обмена данных. По нажатию этой кнопки написан следующий код:

```
procedure TForm1.InfoButtonClick(Sender: TObject);
begin
  if Clipboard.HasFormat(CF_TEXT) then
    Application.MessageBox('Буфер содержит текст', 'Внимание!');
  if Clipboard.HasFormat(CF_BITMAP) then
    Application.MessageBox('Буфер содержит картинку', 'Внимание!');
  if Clipboard.HasFormat(CF_METAFILEPICT) then
    Application.MessageBox('Буфер содержит векторную картинку', 'Внимание!');
  if Clipboard.HasFormat(CF_PICTURE) then
    Application.MessageBox('Буфер содержит объект типа TPicture', 'Внимание!');
  if Clipboard.HasFormat(CF_COMPONENT) then
    Application.MessageBox('Буфер содержит компонент', 'Внимание!');
end;
```

SetComponent – назначить в буфер обмена компонент. В качестве единственного параметра нужно указать компонент, который надо скопировать в буфер.

SetTextBuf - назначить в буфер обмена текстовый буфер. В качестве единственного параметра нужно указать буфер *PChar*, который надо скопировать в буфер.

 На компакт диске, в директории \Примеры\Глава 23\Clipboard ты можешь увидеть пример этой программы.

23.3 Картинки и буфер обмена.

Давай теперь разберёмся, как работать с изображениями в буфере обмена. Как ты знаешь, у нас основной компонент для работы с картинками является *TImage*. Вот давай напишем пример, в котором будет копироваться и вставляться изображение из буфера обмена в компонент *TImage*.

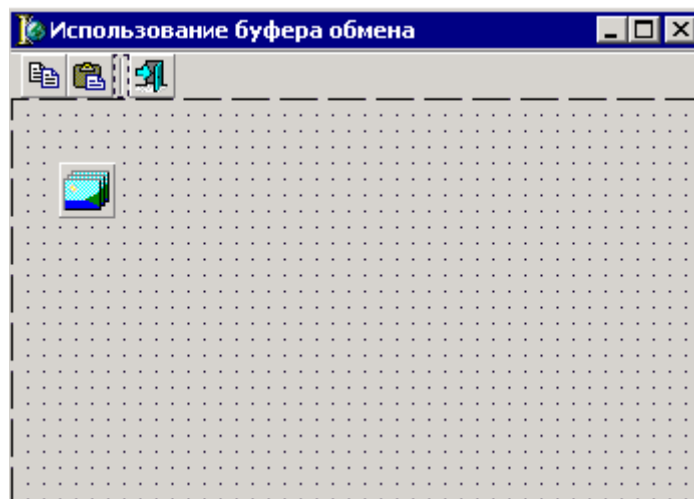


Рисунок 23.3.1 Форма будущей программы

Создай новый проект, и помести на форму две кнопки «Копировать» и «Вставить». По центру окна расположи компонент *TImage*, который будет хранить изображения. Мою форму ты можешь увидеть на рисунке 23.3.1.

По событию *OnClick* для кнопки «Копировать» пишем следующий код:

```
procedure TForm1.CopyButtonClick(Sender: TObject);
begin
  Clipboard.Assign(Image1.Picture);
end;
```

Здесь я просто устанавливаю методом *Assign* в буфер обмена содержимое свойства *Picture* компонента *Image1*. Свойство *Picture* имеет тип *TPicture*, который среди своих родителей имеет объект *TPersistent*, поэтому мы можем использовать метод *Assign*. Посмотри на рисунок 23.3.2, на котором показана иерархия объекта *TPicture*.



Рисунок 23.3.2 Иерархия объекта *TPicture*

Теперь посмотрим на код, который надо написать по нажатию кнопки «Вставить»:

```
procedure TForm1.PasteButtonClick(Sender: TObject);
begin
  Image1.Picture.Assign(Clipboard);
end;
```

Здесь происходит обратное назначение свойству *Picture* содержимое буфера обмена с помощью метода *Assign*.

В принципе, пример готов и на этом можно было бы остановиться, но я здесь же хочу показать, как сделать так, чтобы кнопка «Вставить» была доступна только тогда, когда в буфере обмена находится именно картинка. Для этого сначала движемся в раздел **private** и объявляем там следующее:

```
private
{ Private declarations }
FClipboardOwner:HWND;
procedure WMDrawClipboard(var Msg: TWMDrawClipboard);
message WM_DRAWCLIPBOARD;
```

Я объявил здесь переменную *FclipboardOwner* типа *HWND*. Это тип, который используется для идентификации окна. Вспомни, каждый раз, когда нам нужно было получить или передать указатель на окно, то мы использовали свойство *Handle*. Вот это свойство имеет тип *HWND*, и здесь мы явно объявили переменную такого же типа, чтобы мы могли работать с окнами.

Здесь так же объявлена процедура *WMDrawClipboard* с одним лишь параметром типа *TWMDrawClipboard*. Это процедура обработчик события *WM_DRAWCLIPBOARD*. Об этом говорит соответствующая надпись после объявления процедуры и точки с запятой. Там у нас стоит ключевое слово **message** и имя сообщение, на которое должна откликаться процедура. Вот таким нехитрым способом мы вручную описали обработчик системного сообщения, которого нет в Delphi. Имена всех сообщений Windows ты можешь найти в директории Delphi, поддиректории *Source/Rtl/Win*, в файле *Messages.pas*. Такие имена всегда начинаются с приставки *WM_* (Windows Message – сообщение Windows).

Итак, описанная нами процедура будет вызываться каждый раз, когда изменилось содержимое буфера обмена. Теперь нажимаем Ctrl+Shift+C и Delphi создаёт для нас пустую заготовку описанной процедуры. В ней пишем следующее:

```
procedure TForm1.WMDrawClipboard(var Msg: TWMDrawClipboard);
begin
  SendMessage(FClipboardOwner, WM_DRAWCLIPBOARD, 0, 0);
  Msg.Result := 0;
  ClipboardChanged;
end;
```

Первые две строчки я опущу, и описывать не буду. Поверь мне, они необходимы. В последней строке я вызываю процедуру *ClipboardChanged*. Она должна выглядеть так:

```
procedure TForm1.ClipboardChanged;
var
  I: Integer;
begin
  PasteButton.Enabled := False;
  for I := 0 to Clipboard.FormatCount - 1 do
  begin
```

```
if Clipboard.HasFormat(CF_BITMAP) then
begin
  PasteButton.Enabled := True;
  Break;
end;
end;
end;
```

Для начала я делаю кнопку «Вставить» неактивной. Потом я запускаю цикл от 0 до количества форматов в буфере обмена *Clipboard.FormatCount*. Внутри цикла происходит проверка, если формат соответствует *CF_BITMAP*, то кнопку «Вставить» можно делать активной и прерывать цикл проверки.

И последнее, что надо сделать в нашей программе – написать обработчик события *OnShow* для нашей главной формы. В нём пишем следующее:

```
procedure TForm1.FormShow(Sender: TObject);
begin
  FClipboardOwner := SetClipboardViewer(Handle);
  ClipboardChanged;
end;
```

В первой строке я вызываю функцию *SetClipboardViewer*. Она устанавливает указанное в качестве параметра окно (наше главное окно) в системе в качестве наблюдателя за буфером обмена. После этого, как только буфер изменится, нашему окну будет отправлено соответствующее сообщение, и мы его поймем процедурой *WMDrawClipboard*.

Во второй строке я вызываю процедуру *ClipboardChanged*, чтобы при старте программы произошла проверка, а вдруг там находится картинка или наоборот. Если этой проверки не производить, то программа после запуска ещё не будет знать, что находится в буфере обмена, пока он не изменится и программа не получит соответствующего сообщения.

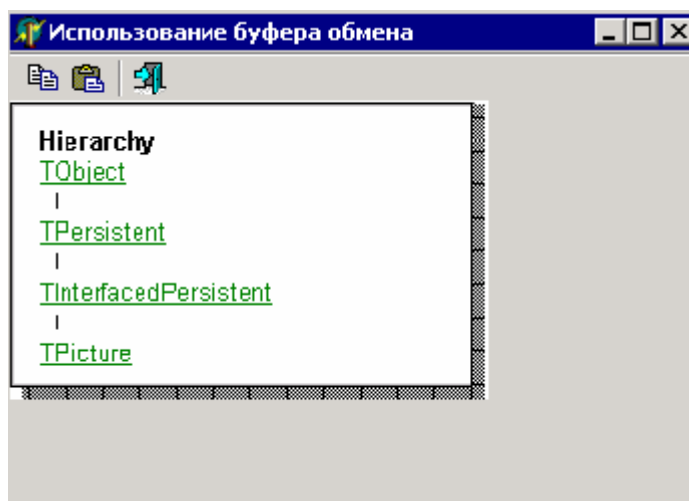


Рисунок 23.3.3 Пример работы программы.

Вот теперь на этом всё. Пример рабочей программы ты можешь увидеть на рисунке 23.3.3. На этом рисунке я вставил в нашу программу изображение иерархии объекта

TPersistent. Попробуй запустить свой вариант программы и последить за кнопкой «Вставить». Запусти любые другие программы и попробуй в них поместить в буфер данные разного типа. Как только ты поместишь туда картинку, так сразу же твоя программа отреагирует на это и сделает кнопку «Вставить» активной.

 На компакт диске, в директории \Примеры\Глава 23\Image ты можешь увидеть пример этой программы.

23.4 Создание собственного формата для работы с буфером.

Представь себе ситуацию, когда в твоей программе есть какой-то объект и нужно дать пользователю возможность копировать её в буфер и вставлять в нужное место. Стандартные форматы данных для буфера *CF_TEXT*, *CF_BITMAP*, *CF_METAFILEPICT*, и так далее не подходят, но данные копировать надо. Лично я с такой ситуацией встречаюсь практически в каждой своей программе. В таких случаях нужно создать свой собственный формат данных, с которым и будет работать буфер обмена.

Язык Delphi – это объектный язык и тут я Америки для тебя уже не открываю. А по условиям объектного программирования, чтобы добавить новые возможности к уже существующему объекту нужно создать его потомка. В данном случае мы по идее должны вывести потомка из *TClipboard* и наделить его великолепными и уникальными возможностями по форматированию нужных нам данных. В данном случае это будет глупо. Объектное программирование хорошее дело, но только когда оно в меру. В данном случае нам не понадобится наследственность и мы не будем возиться с родителями и детками.

Итак, создай новый проект и сразу же создай в нём новый модуль (File->New->Unit). Delphi создаст пустой модуль, который мы сохраним под именем *ClipboardFormatUnit*. Результат – модуль вот с таким вот содержимым:

```
unit ClipboardFormatUnit;

interface

implementation

end.
```

После ключевого слова **interface** добавляем раздел **type** и объявление структуры *TLineData* и нового объекта *TLineClipboard*.

```
type
  TLineData=record
    Name:String[100];
    LastName:String[100];
    Bothday:String[10];
    Age:Integer;
    Telephone:String[15];
  end;

  TLineClipboard=class
```

```
public
  LineData:TLineData;
  procedure CopyToClipboard;
  procedure PasteFromClipboard;
end;
```

Структура *TLineData* состоит из пяти полей. Именно эту структуру мы будем помещать в буфер обмена. Как ты уже понял, объект *Clipboard* не может работать со структурами, и мы сейчас напишем модуль, с помощью которого мы научим его это делать.

После структуры идёт объявление нового объекта. Здесь мы объявляем новый объект вручную описывая все его методы и свойства. Чаще всего за нас это делал Delphi. Обрати внимание на то, что он объявлен, как простой объект без каких либо родителей (*TLineClipboard=class*). Несмотря на это, он будет иметь родителя – *TObject*, потому что все объекты должны иметь родителя и если ничего не указано, то будет использоваться базовый объект *TObject*. У нового объекта будет только одно свойство типа структуры *TLineData* и два метода для копирования и вставки данных в буфер обмена.

Теперь, после раздела **type** напишем **var** и опишем одну переменную:

```
var
  CF_PERSONDATA:word;
```

В этой переменной будет храниться указатель на зарегистрированный формат для буфера обмена. Давай не будем откладывать это дело на потом, а сразу же реализуем регистрацию в системе этого нового формата. Для этого в конце модуля, перед последним «**end.**» пишем:

```
initialization
  CF_PERSONDATA:=RegisterClipboardFormat('CF_PDATA');

end.
```

Здесь мы объявили блок **initialization**, который всегда выполняется автоматически при обращении к модулю или любому его содержимому. В этом блоке я присваиваю переменной *CF_PERSONDATA* результат выполнения функции *RegisterClipboardFormat*. Эта функция регистрирует новый формат для буфера обмена с именем указанным в качестве единственного параметра. Результат выполнения функции – число, идентифицирующее зарегистрированный формат данных. После этого, этот формат можно увидеть в свойстве *Formats* объекта *Clipboard*.

Вот теперь перейдём к реализации функций записи и чтения данных из буфера.

```
procedure TLineClipboard.CopyToClipboard;
var
  Data:THandle;
  DataPtr:Pointer;
begin
  //Выделяем память под данные
  Data:=GlobalAlloc(GMEM_MOVEABLE, SizeOf(LineData));
```

```

try
  DataPtr:=GlobalLock(Data);
  Move(LineData, DataPtr^, SizeOf(TLineData));

  //Заполняем буфер обмена
  Clipboard.Open;
  Clipboard.SetAsHandle(CF_PERSONDATA, Data);
  Clipboard.AsText:=LineData.Name+#13#10+LineData.LastName+#13#10+
    LineData.Bothday+#13#10+IntToStr(LineData.Age)+#13#10+
    LineData.Telephone;
  Clipboard.Close;
  GlobalUnlock(Data);
except
  GlobalFree(Data);
end;
end;
end;

```

В процедуре объявлено две переменные:

Data – в эту переменную мы будем выделять память для хранения структуры, которую надо будет поместить в буфер обмена.

DataPtr – указатель на выделенную для переменной *Data* память.

В самом начале процедуры я присваиваю переменной *Data* результат вызова функции *GlobalAlloc*. Эта функция выделяет нужный кусок памяти в глобальной памяти. У неё два параметра:

1. Параметры (флаги) выделяемой памяти. Здесь я указал флаг *GMEM_MOVEABLE*, который указывает на то, что память может быть перемещаемой. Например, когда ОС не хватает оперативной памяти, то она может выгрузить некоторую часть памяти на диск и по мере надобности вернуть эту память обратно. Такой флаг позволяет ОС производить такие манипуляции.
2. Размер выделяемой памяти. Здесь я указываю размер *SizeOf* структуры *LineData*.

Следующим этапом я блокирую выделенную память с помощью функции *GlobalAlloc* и получаю указатель на выделенную память, который сохраняется в переменной *DataPtr*.

У нас есть выделенная память, и мы её временно заблокировали для работы с ней. Теперь мы должны скопировать в эту память структуру, которая должна быть помещена в буфер обмена. Для этого я пользуюсь процедурой *Move*, которая копирует данные из одного участка памяти (первый параметр – структура *LineData*) в другой участок (второй параметр – выделенная память *DataPtr*). Третий параметр – это размер копируемых данных. Я указываю тут размер нашей структуры.

Память подготовлена и в неё занесены данные для буфера обмена. Теперь можно приступить к установке этих данных в буфер. Для начала буфер надо открыть – *Open* для записи. Следующим этапом я заносу в буфер данные в виде структуры (зарегистрированного нами формата *CF_PERSONDATA*). Для этого я использую метод *SetAsHandle*. У этого метода два параметра:

1. Тип заносимых данных.
2. Данные.

Дальше, я заносу данные в виде текста, чтобы программы, которые не знают о существовании моего собственного формата, могли прочитать данные в виде текста. Для этого текст надо занести в свойство *AsText* объекта *Clipboard*. В это свойство я заносу все поля структуры, разделённые символами *#13#10*, Символ *#13* означает конец строки, а *#10*

означает перевод каретки на новую строку. Получается, что каждый параметр будет занесён в виде отдельной строки. Чуть позже ты увидишь, как это выглядит на практике.

Данные занесены, можно уничтожить всё, что мы натворили. Для начала закрываю буфер обмена с помощью вызова метода *Close* объекта *Clipboard*. Потом вызываю метод *GlobalUnlock*, который разблокирует память. Уничтожить выделенную память (вызов *GlobalFree*) нужно только если во время записи произошла какая-то ошибка, поэтому я поместил вызов этой процедуры в блоке **except..end**. Если ошибок не было, то память должна остаться целой и невредимой, потому что там храниться структура.

Вот теперь разберёмся с кодом, который получает данные из буфера обмена. Взгляни на процедуру *PasteFromClipboard*:

```
procedure TLineClipboard.PasteFromClipboard;
var
  Data:THandle;
  DataPtr:Pointer;
begin
  Data:=Clipboard.GetAsHandle(CF_PERSONDATA);
  if Data=0 then exit;

  DataPtr:=GlobalLock(Data);
  Move(DataPtr^, LineData, SizeOf(TLineData));

  GlobalUnlock(Data);
end;
```

В первой строке кода я пытаюсь получить данные из буфера обмена с помощью функции *GetAsHandle*. Этой функции нужно передать формат, в котором мы хотим получить данные. Если значение, которое нам вернула *GetAsHandle* равно нулю, то данных в буфере нет, или они имеют несовместимый формат. В этом случае процедура прерывает своё выполнение.

Если всё нормально, то в переменной *Data* будет указатель на блок памяти с данными буфера обмена. Для их получения я блокирую память и пользуюсь уже знакомой процедурой *Move* для копирования данных из памяти буфера обмена (первый параметр) в структуру *LineData* (второй параметр).

Данные получены, можно разблокировать память с помощью процедуры *GlobalUnlock*.

Всё, модуль готов. Теперь переходим к написанию основной программы, которая будет использовать созданный нами формат данных для копирования и вставки через буфер обмена. У нас уже создан новый проект, в котором мы написали модуль *ClipboardFormatUnit*. Перейди в основное окно, и помести на форму следующие компоненты:

1. Две кнопки «Копировать» и «Вставить».
2. Компонент *StringGrid*. В нём нужно изменить свойство *goEditing* в разделе *Option* на *true*, чтобы мы могли редактировать ячейки в сетке.
3. *Memo* в котором мы будем отображать содержимое буфера в виде текста.

Мою форму ты можешь увидеть на рисунке 23.4.1. Для реализации кнопок я использовал компонент *ToolBar*, чтобы программа выглядела более эстетично и красиво, но ты можешь поступить и по-другому.

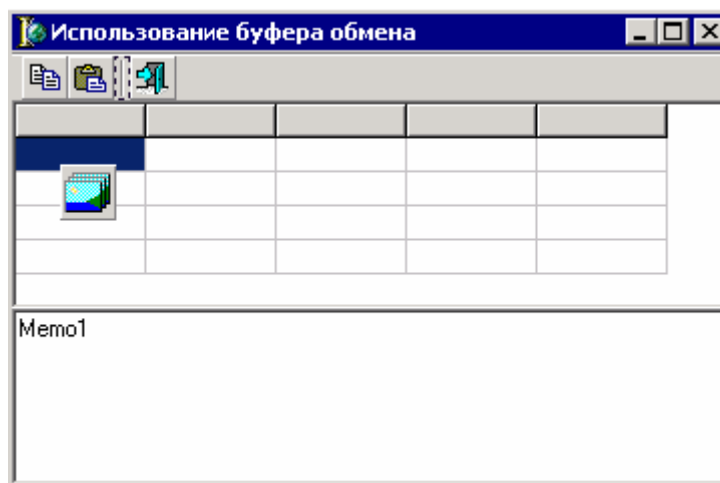


Рисунок 23.4.1 Форма будущей программы

Сразу же добавь в раздел **uses** наш модуль *ClipboardFormatUnit*, чтобы ты мог использовать новый формат из главной формы.

Теперь создадим обработчик события *OnClick* для кнопки «Копировать» и напишем в нём следующее:

```

procedure TForm1.CopyButtonClick(Sender: TObject);
var
    LineClipboard:TLineClipboard;
begin
    LineClipboard:=TLineClipboard.Create;

    LineClipboard.LineData.Name:=StringGrid1.Cells[0, StringGrid1.Row];
    LineClipboard.LineData.LastName:=StringGrid1.Cells[1, StringGrid1.Row];
    LineClipboard.LineData.Bothday:=StringGrid1.Cells[2, StringGrid1.Row];
    LineClipboard.LineData.Age:=StrToInt(StringGrid1.Cells[3, StringGrid1.Row]);
    LineClipboard.LineData.Telephone:=StringGrid1.Cells[4, StringGrid1.Row];

    LineClipboard.CopyToClipboard;

    LineClipboard.Free;
end;

```

В разделе **var** у меня объявлена одна переменная типа *TLineClipboard* – объект для работы с буфером обмена, который мы написали в модуле *ClipboardFormatUnit*. В первой строке кода я инициализирую эту переменную.

Потом я заполняю структуру *LineData* объекта *LineClipboard* данными из выделенной строки сетки. По завершению этого процесса я копирую данные в буфер обмена с помощью вызова метода *CopyToClipboard* объекта *LineClipboard*.

Данные скопированы, значит, объект уже не нужен, и его можно уничтожить. Для этого я вызываю метод *Free*.

По нажатию кнопки «Вставить» пишем следующий код:

```

procedure TForm1.PasteButtonClick(Sender: TObject);
var
    LineClipboard:TLineClipboard;
begin

```

```

LineClipboard:=TLineClipboard.Create;

if Clipboard.HasFormat(CF_PERSONDATA) then
begin
  LineClipboard.PasteFromClipboard;
  StringGrid1.Cells[0, StringGrid1.Row]:=LineClipboard.LineData.Name;
  StringGrid1.Cells[1, StringGrid1.Row]:=LineClipboard.LineData.LastName;
  StringGrid1.Cells[2, StringGrid1.Row]:=LineClipboard.LineData.Bothday;
  StringGrid1.Cells[3, StringGrid1.Row]:=IntToStr(LineClipboard.LineData.Age);
  StringGrid1.Cells[4, StringGrid1.Row]:=LineClipboard.LineData.Telephone;
end;

LineClipboard.Free;

Memo1.Lines.Clear;
Memo1.PasteFromClipboard;
end;

```

Опять же, здесь объявлена переменная *LineClipboard*, которая инициализируется в первой строке кода. После этого я проверяю, если буфер обмена содержит информацию в формате *CF_PERSONDATA* (это созданный нами формат), то мы читаем буфер с помощью метода *PasteFromClipboard*. После этого я заполняю поля текущей строки из структуры *LineData* объекта *LineClipboard*.

В самом конце процедуры я очищаю компонент *Memo1* и заставляю его с помощью метода *PasteFromClipboard* прочитать данные из буфера. Этот компонент не знает о существовании нашего формата и читает данные из буфера обмена как текст (это его родной формат). Получается, что мы увидим в компоненте то, что мы записали в свойство *AsText* объекта *Clipboard*. Посмотри на рисунок 23.4.2 и убедись в этом. Там я заполнил поля первой строки, скопировал строку в буфер и потом вставил данные в третью строку. Одновременно со вставкой в компонент *StringGrid* произошла вставка текста буфера в компонент *Memo*.

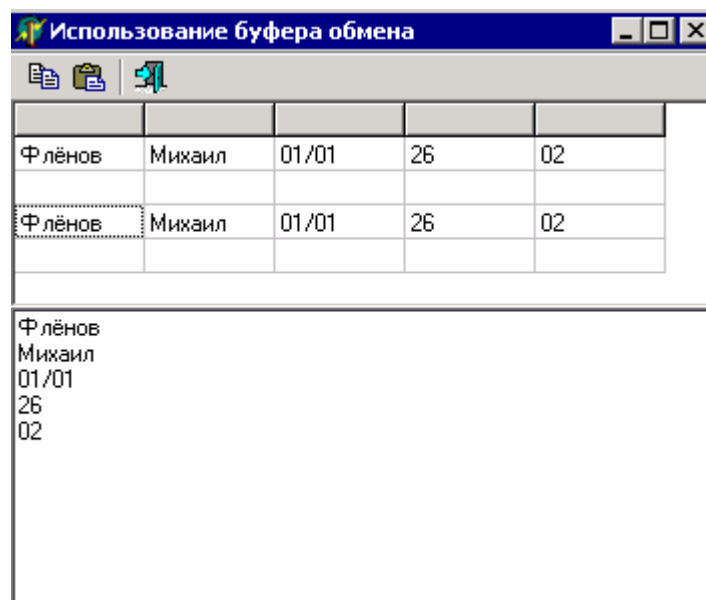



Рисунок 23.4.2. Результат работы программы

 На компакт диске, в директории **\Примеры\Глава 23\New Format** ты можешь увидеть пример этой программы.

