Управление компонентами и ресурсами приложения

Оглавление

[Доступ к компонентам формы 2](#_Toc370549950)

[Доступ через статическое преобразование типов 2](#_Toc370549951)

[Доступ через динамическое преобразование типов 3](#_Toc370549952)

[Доступ по номеру компонента в иерархии объектов 4](#_Toc370549953)

[Подчиненность объекта и определение его типа 6](#_Toc370549954)

[Работа с ресурсами в Borland C++ Builder 6](#_Toc370549955)

[Ресурсы, которые может использовать приложение 6](#_Toc370549956)

[.exe файл в ресурсах и его выполнение 9](#_Toc370549957)

[Текстовая информация в ресурсах 10](#_Toc370549958)

[Картинки в ресурсах 11](#_Toc370549959)

[Иконки в ресурсах 14](#_Toc370549960)

[Курсоры в ресурсах 16](#_Toc370549961)

[Строки символов в ресурсах 19](#_Toc370549962)

[HTML документы в ресурсах 20](#_Toc370549963)

[.wav и другие музыкальные файлы в ресурсах 20](#_Toc370549964)

# Доступ к компонентам формы

Создавая программу в среде Borland C++ Builder, программист помещает на форму множество компонент (кнопки, меню, таймеры, контейнеры для графики, списки и т. д). Естественно, возникает вопрос - как программно обратиться к тому или иному компоненту. Ответ на него прост - по имени компонента. Но, например, когда на один обработчик событий замкнуто несколько элементов, то вопрос о том как определить какой именно компонент вызвал событие становится не таким простым. В то же время, несколько механизмов Borland C++ Builder позволяют выполнить данную и другие возникающие на практике задачи.

Конструкция **TObject \*Sender** (ссылка на объект, вызвавший событие) присутствует практически во всех обработчиках событий компонент и может адресовать любые объекты. Для того, чтобы воспользоваться таким указателем, необходимо выполнить преобразование типа от абстрактного прародителя компонента (TObject) к конкретному. Преобразования можно выполнить двумя способами - статическим в стиле обычного C++ и динамическим (расширенным ANSI).

## Доступ через статическое преобразование типов

1. **Прямое преобразование типа**. Данный обработчик прячет кнопку, вызвавшую событие ее нажатия:

void \_\_fastcall

TForm1::Button1Click(TObject \*Sender) {

TButton \*ptButton;

ptButton=(TButton \*)Sender;

ptButton->Visible=false;

}

Этот обработчик можно назначить сразу нескольким компонентам, но только класса TButton. При попытке преобразования к другого типу компонента, программа выдаст исключение на преобразование типов. Если же необходимо выполнить преобразование из неизвестного класса, то можно использовать конструкции try...catch, но проще воспользоваться динамическим преобразованием - оператором dynamic\_cast.

2. **Преобразование к TTreeView**. Следующий код делает тоже, что и предыдущий (прячет кнопку):

#include <vcl\comctrls.hpp>

//...

void \_\_fastcall

TForm1::Button1Click(TObject \*Sender) {

TTreeView \* ptTreeViev=(TTreeView \*)Sender;

ptTreeViev->Visible=false;

}

Преобразование к классу TTreeView позволяет достаточно просто получить любое свойство объекта. Например, пусть обработчики событий OnClick двух кнопок направлены на один код Button1Click(). Для кнопки 2 установим свойство Cursor в crHandPoint.

#include <vcl\comctrls.hpp>

//...

void \_\_fastcall

TForm1::Button1Click(TObject \*Sender) {

Memo1->Lines->Clear();

TTreeView \* ptTreeViev=(TTreeView\*) Sender;

Memo1->Lines->Add(ptTreeViev->Name);

Memo1->Lines->Add(ptTreeViev->Align);

Memo1->Lines->Add(ptTreeViev->Left);

Memo1->Lines->Add(ptTreeViev->Cursor);

}

Сравните результаты, получаемые при нажатии обеих кнопок.

## Доступ через динамическое преобразование типов

Динамическое преобразование типов доступно в программе через конструкцию:

dynamic\_cast <TButton\*>

Конструкция пытается преобразовать некоторый тип объекта к указанному типу. При удаче возвращается значение отличное от нуля. Так, если обработчики нажатия двух кнопок компонента TButton и одной кнопки компонента TSpeedButton замкнуты на один код, то при нажатии кнопок компонентов TButton они будут скрыты, при нажатии кнопки компонента TSpeedButton ничего не произойдет:

void \_\_fastcall

TForm1::Button1Click(TObject \*Sender) {

TButton \*ptButton=dynamic\_cast<TButton\*>(Sender);

If (ptButton == NULL) return;

ptButton->Visible=false;

}

Если события от нескольких одинаковых компонент используют одни и те же функции, то целесообразно их замкнуть на один обработчик, а различать объекты внутри обработчика можно по свойству Tag, присвоив ему разные значения для разных компонент.

Например, пусть у нескольких кнопок компонент TSpeedButton в соответствии с их номером установлено свойство Tag, равное номеру кнопки, тогда:

void \_\_fastcall

TForm1::SpeedButton1Click(TObject \*Sender) {

if(dynamic\_cast<TSpeedButton \*>(Sender)) {

int i=((TSpeedButton \*)(Sender))->Tag;

switch(i) {

case 0:

//Код для кнопки 1

break;

case 1:

//Код для кнопки 2

break;

}

// Общий код

}

}

Посредством конструкции ((класс компонента \*)(Sender))-> также можно получить доступ к любому свойству объекта.

## Доступ по номеру компонента в иерархии объектов

Все компоненты на форме хранятся также в массиве Components. При переносе компонента на форму ему автоматически присваивается сквозной номер в массиве Components, а число компонентов на форме хранится в свойстве ComponentCount. Используя это, можно осуществить доступ к любому компоненту формы. Так, следующая функция ищет среди всех компонент формы компоненты TLabel и среди них компоненту с требуемым номером. После нахождения требуемой компоненты записывает в нее некоторый текст:

void \_\_fastcall

vNewText(int viNumber,AnsiString vasNewText) {

AnsiString vasName=AnsiString("Label") + IntToStr(viNumber);

for(int i = 0;i < Form1->ComponentCount; i++) {

if(Form1->Components[i]->ClassNameIs("TLabel")) {

if(((TLabel \*)Form1->Components[i])->Name ==

vasName) {

((TLabel \*)Form1->Components[i])->Name = vasNewText;

//или записать свойство Caption

}

}

}

}

Кроме того можно использовать свойство Tag для определения конкретного компонента. Например, пусть на форме имеется 100 панелей, свойство Tag панели определяет номер панели (1-100). При передачи в функцию числа n - все панели с номером большим n исчезнут.

void \_\_fastcall

TForm1::vVisibleFalse(int viNumber) {

for(int i = 0;i < ComponentCount; i++) {

if(Components[i]->ClassNameIs("TPanel") &&

Components[i]->Tag > viNumber) {

if(Components[i]->Tag <= 100)

((TPanel \*)Components[i])->Visible=false;

}

else if(Components[i]->ClassNameIs("TPanel") &&

Components[i]->Tag <= viNumber &&

Components[i]->Tag != 0)

((TPanel \*)Components[i])->Visible=true;

}

}

Аналогично свойству Tag можно использовать любое другое свойство, которое имеет компонент. В следующем коде эта конструкция использована для позиционирования TImage в зависимости от его номера и коэффициентов, рассчитанных для определения требуемого положения фрагмента изображения (изображение может быть разбито на матрицу фрагментов от 4\*4 до 10\*10):

for(int i = 0;i < ComponentCount; i++) {

if(Components[i]->ClassNameIs("TImage") &&

Components[i]->Tag <= viNumX &&

Components[i]->ClassNameIs("TImage") &&

Components[i]->Tag >= viNumStartX &&

Components[i]->Tag != 0) {

((TImage \*)Components[i])->Left=

Components[i]->Tag\*viK\_X;

((TImage \*)Components[i])->Top=

Components[i]->Tag\*viK\_Y;

}

}

## Подчиненность объекта и определение его типа

Используя функцию ClassType() можно определить класс источника события (в примере для компонента TButton), а далее, применяя функцию ClassParent() можно двигаться вверх по иерархии объектов. Этот код размещен здесь по причине того, что метод ClassName() может быть использован для проверки типа объекта перед его использованием.

void \_\_fastcall

TForm1::Button1Click(TObject \*Sender) {

TClass tClass;

Memo1->Lines->Clear();

tClass = Sender->ClassType();

int i=0;

while(tClass != NULL) {

Memo1->Lines->Add(String(i)+" "+

tClass->ClassName());

tClass = tClass->ClassParent();

i++;

}

}

Пример использования метода ClassType() для проверки класса источника, вызвавшего данное событие:

void \_\_fastcall

TForm1::Button1Click(TObject \*Sender) {

TClass tClass;

tClass = Sender->ClassType();

AnsiString vasS=tClass->ClassName();

if(vasS == "TButton") {

TButton \*ptButton;

ptButton=(TButton\*)Sender;

ptButton->Visible=false;

}

}

# Работа с ресурсами в Borland C++ Builder

## Ресурсы, которые может использовать приложение

Программа Windows - это не только код, но и набор данных типа картинок, строк, значков, диалоговых окон и так далее. Все это хранится внутри .exe модуля программы в виде ***ресурсов***. То есть, ресурсы в Windows можно рассматривать как вид данных, которые уже инициализированы.

Ресурсы не загружаются в память вместе со всей программой, а находятся на диске до тех пор, пока их не вызовет программа. Это позволяет не плодить на диске лишние микрофайлы и экономить оперативную память при выполнении приложения.

Ресурсы возможно помещать в отдельные DLL библиотеки. Например, если в DLL хранятся диалоговые окна, меню, строковые сообщения и т.п., то при необходимости их изменения не понадобится изменять сами программы.

Ресурсы можно разделить на два типа: двоичные и текстовые. Двоичные ресурсы это обычно растровые картинки, иконки, курсоры.., а текстовые ресурсы это строки, шаблоны диалоговых окон и т.п.

Ресурсы в проектах приложений Borland C++ Builder хранятся раздельно. Файл проекта с расширением .dfm содержит информацию о ресурсах окон и компонентов для конструирования самой формы приложением. Он создается автоматически при создании проекта и изменяется при добавлении компонентов на форму. Другой файл с расширением .res (Project1.res) - файл графических ресурсов - хранит курсоры, иконки и рисунки.

Для редактирования графических ресурсов Borland C++ Builder предусмотрен редактор Image Editor. Он позволяет редактировать битовые изображения (.bmp), пиктограммы (.ico), курсоры (.cur), а также сами файлы ресурсов (.res и .dcr).

Файлы ресурсов .rc также могут быть подключены к проекту Borland C++ Builder. При компиляции проекта информация из файлов .rc и информация на которую они ссылаются будет помещена в файл с расширением .res, а ресурсы из этого файла перенесены в .exe файл и становятся доступны приложению.

Каждый тип ресурсов имеет свой идентификатор. Полный список ресурсов можно посмотреть в файле **winuser.h**.

Некоторые типы ресурсов:

* RT\_ACCELERATOR - таблица ускорителей;
* RT\_ANICURSOR - анимированный курсор;
* RT\_ANIICON - анимированная икона;
* RT\_BITMAP - растровая картинка;
* RT\_CURSOR - курсор;
* RT\_DIALOG - диалоговое окно;
* RT\_FONT - шрифт;
* RT\_FONTDIR - набор шрифтов:
* RT\_GROUP\_CURSOR - курсор;
* RT\_GROUP\_ICON - икона;
* RT\_HTML - HTML документ;
* RT\_ICON - иконка;
* RT\_MENU - меню;
* RT\_MESSAGETABLE - таблица сообщений;
* RT\_RCDATA - данные определенные приложением (RAW);
* RT\_STRING - таблица строк:
* RT\_VERSION - ресурс версия;
* RT\_HTML -Документ HTML.

Таким образом - ресурсы это составная часть файлов .exe и .dll. Практически, при создании приложения, файлы ресурсов только переписываются в двоичном виде в .exe(.dll) файлы. Составляющие ресурсы извлекаются и используются по мере выполнения программы.

Для просмотра используемых приложением ресурсов можно использовать функцию WIN32 API EnumResourceTypes().

BOOL EnumResourceTypes (

HMODULE hModule,            // указатель на модуль

ENUMRESTYPEPROC lpEnumFunc, // функция перечисления

LONG\_PTR lParam             // параметры, определенные программистом

);

Пример использования. Поместим на форму один компонент TButton и определяем внешнюю функцию f\_EnumFunc(), которая будет функцией перечисления:

BOOL WINAPI

f\_EnumFunc (HMODULE hModule, LPTSTR lpType, LONG lParam);

//...

BOOL WINAPI

f\_EnumFunc(HMODULE hModule,LPTSTR lpType,LONG lParam) {

char vrgchBuffer[100];

if((ULONG)lpType & 0xFFFF0000) {

wsprintf(vrgchBuffer,"%s\n",lpType);

}

else {

wsprintf(vrgchBuffer,"%u\n",(USHORT)lpType);

}

MessageBox(0,vrgchBuffer,"Ресурс",MB\_ICONEXCLAMATION);

return true;

}

В обработчике события нажатия кнопки TButton пишем код:

void \_\_fastcall

TForm1::Button1Click(TObject \*Sender) {

HMODULE hMod=NULL;

hMod = LoadLibrary("calc.exe");

if(hMod != NULL) {

EnumResourceTypes(hMod,(ENUMRESTYPEPROC)f\_EnumFunc,0);

FreeLibrary(hMod);

}

else Label1->Caption="Нет возможности загрузить модуль программы";

}

Цифры выдаваемые программой говорят о номерах ресурсов, которые использовал программист при ее написании.

## .exe файл в ресурсах и его выполнение

Поставим целью спрятать в ресурсах создаваемого приложения некоторую другую программу, а в процессе выполнения приложения вызвать ее из ресурсов.

Поместим на форме компонент TMemo и два компонента TButton. В обработчике Button1Click напишем код:

void \_\_fastcall

TForm1::Button1Click(TObject \*Sender) {

Memo1->Lines->Clear();

Memo1->Lines->Add("MYEXE EXEFILE calc.exe");

Memo1->Lines->SaveToFile("a.rc");

}

Откомпилируем приложение, при нажатии кнопки Button1 в директории программы будет создан файл со строчкой (которая отобразится в компоненте Memo):

MYEXE EXEFILE calc.exe

Это строка "Имя ресурса | Тип ресурса | Имя программы" - полностью идентифицирует ресурс, а именно - программу calc.exe, которую можно будет спрятать в ресурсах приложения.

Далее, присоеденяем к проекту этот файл (Меню: Project/Add To Project.. выбираем a.rc файл).

Если сейчас попробовать откомпилировать проект, то компилятор выдаст ошибку. Причина - путь к calc.exe должен быть указан полный и, хотя ресурс еще не используется, но уже формируется. Поэтому скопируем calc.exe из директории Windows в директорию приложения (в противном случае необходимо указать полный путь). Приложение компилируется и можно начинать использовать ресурс. Удаляем файл "calc.exe" из директории проекта.

В обработчике Button2Click напишем код:

void \_\_fastcall

TForm1::Button2Click(TObject \*Sender) {

char \*pchResType = "EXEFILE";

TResourceStream\* ptRes =

new TResourceStream((int)HInstance, "MYEXE", pchResType);

//Это не цель, но строка показывает, что

//TResourceStream содержит полную копию .exe файла

ptRes->SaveToFile("calc.exe");

//Запускаем калькулятор

ShellExecute(Handle,"open","calc.exe",NULL,NULL,SW\_SHOWNORMAL);

delete ptRes;

}

Здесь осуществляется выбор ресурса в поток - файла calc.exe (из файла "a.res", который уже создан). TResourceStream - потоковый класс, который обеспечивает доступ к компилируемым ресурсам в приложения. Он имеет множество методов для доступа к ресурсам, в том числе и для сохранения потока.

При нажатии на вторую кнопку калькулятор вызывается даже если его удалить не только из текущей директории приложения, но и из директории Windows.

Это не единственный способ вызова и запуска на выполнения .exe файла из ресурсов, но наиболее простой. Другие способы извлечения ресурсов и работы с потоками будут рассмотрены ниже - там где это или целесообразно, или другим способом задача не выполнима. При желании можно поэкспериментировать с TMemoryStream, WinExecute() и.т.п.

## Текстовая информация в ресурсах

Помещение и извлечение текстовой информации, скопированной в ресурс из файла, и получение доступа к ней мало отличается от примера, описанного выше.

Создадим аналогичный проект с одним компонентом TMemo и двумя компонентами TButton. Шаги создания кода приложения будут аналогичны пройденным в предыдущем примере. Приведем лишь коды обработчиков событий нажатия кнопок.

void \_\_fastcall

TForm1::Button1Click(TObject \*Sender) {

Memo1->Lines->Clear();

Memo1->Lines->Add("MYTEXT RT\_RCDATA readme.txt");

Memo1->Lines->SaveToFile("a.rc");

}

Здесь одно отличие - тип ресурса вместо EXEFILE стал RT\_RCDATA. После включения в проект файла "a.rc", не забудем поместить в папку проекта любой файл readme.txt.

Последний обработчик имеет отличие от вызова программы связанное только с тем, что информация используется в самом приложении.

void \_\_fastcall

TForm1::Button2Click(TObject \*Sender) {

TResourceStream\* ptRes =

new TResourceStream((int)HInstance, "MYTEXT","RT\_RCDATA");

ptRes->SaveToFile("readme.txt");

//Загружаем ресурс

Memo1->Lines->LoadFromStream(ptRes);

delete ptRes;

}

В результате при нажатии кнопки 2 файл "readme.txt" появляется в папке приложения и отображается в компоненте Memo.

## Картинки в ресурсах

Картинки можно поместить в ресурс как используя файл ресурсов проекта (project1.res) так и отдельный файл ресурсов, аналогично описанному выше хранению текста и программ.

Для рассмотрения первой возможности создадим новый проект и поместим на форму компонент TImage и два компонента TButton. В директорию проекта поместим любой рисунок в формате bmp (например "a.bmp") и откроем его в любом графическом редакторе, например в Microsoft Paint. Не компилируя проект запустим Image Editor (меню Tools/Image Editor). Далее откроем project1.res из директории проекта(File/Open, Projects/project1.res). Выполняем правый клик в поле открывшегося окна и в контекстном меню выбираем пункт New/Bitmap.

И сразу видим недостаток этого метода - максимальное число цветов для изображения - 256. Устанавливаем число цветов и размер (его определяем в Paint - меню Рисунок/Атрибуты). В файле у списка Contents появилась новая ветвь Bitmap с именем рисунка Bitmap1. (Можно воспользовавшись контекстным меню пунктом "Rename" и переименовать рисунок, например в MYBITMAP, но для наглядности делать этого не будем). Выполним двойной клик на имени рисунка - откроется окно редактора с пустым рисунком. Перенесем рисунок из Paint (Сtrl+A, Ctrl+C) (можно и нарисовать свой рисунок в Image Editor) в окно Image Editor (Сtrl+V). Закроем окно редактора рисунка и Image Editor, на вопрос "Save Changes..." кликнем "Yes".

Создадим обработчик нажатия для кнопки и запишем несколько вариантов кода, позволяющего отображать и сохранять рисунок:

//Вариант 1 - загрузка функцией LoadFromResourceName()

Image1->Picture->Bitmap->LoadFromResourceName

((int)HInstance,"Bitmap1");

//Вариант 2 - загрузка функцией LoadImage()

HBITMAP hBmp=LoadImage(GetModuleHandle(NULL),

"Bitmap1",IMAGE\_BITMAP,0,0,

LR\_COPYFROMRESOURCE | LR\_DEFAULTCOLOR |

LR\_CREATEDIBSECTION);

if(hBmp == 0) ShowMessage("Картинка не загружена");

else Image1->Picture->Bitmap->Handle=hBmp;

//Вариант 3 - через использование TBitmap

Graphics::TBitmap\* ptBitmap;

ptBitmap = new Graphics::TBitmap();

ptBitmap->LoadFromResourceName((int)HInstance,"Bitmap1");

Image1->Width=ptBitmap->Width;

Image1->Height=ptBitmap->Height;

//И отобразить несколькими способами

// a).

Image1->Picture->Bitmap->Assign((TPersistent\*)ptBitmap);

// b).

BitBlt(Image1->Canvas->Handle,0,0,ptBitmap->Width,ptBitmap->Height,

ptBitmap->Canvas->Handle,0,0,SRCCOPY);

// c).

Image1->Canvas->Draw(0,0,ptBitmap);

}

Откомпилируем и загрузим приложение, при нажатии кнопки 2 изображение появится в компоненте TImage.

Для рассмотрения использования отдельного файла ресурса в проект внесем некоторые изменения.

В обработчиках событий ButtonClick запишем коды, приведенные ниже. После компиляции и запуска приложения нажимаем первую кнопку и закрываем приложение. Добавляем к проекту файл "a.rc". Вновь компилируем и запускаем приложение и нажимаем вторую кнопку - картинка отображается в компоненте Image.

void \_\_fastcall

TForm1::Button1Click(TObject \*Sender) {

TStringList\* ptSList = new TStringList();

ptSList->Clear();

ptSList->Add("MYBMP RT\_BITMAP \"a.bmp\"");

ptSList->SaveToFile("a.rc");

delete ptSList;

}

void \_\_fastcall

TForm1::Button2Click(TObject \*Sender) {

TResourceStream\* ptRes =

new TResourceStream((int)HInstance,"MYBMP","RT\_BITMAP");

ptRes->SaveToFile("a.bmp");

Image1->Picture->LoadFromFile("b.bmp");

delete ptRes;

}

Пример полностью аналогичен предыдущему примеру, за исключением того, что для формирования файла .rc использован класс TStringList, а тип ресурса стал "RT\_BITMAP".

Отображение рисунка возможно выполнить и немного по другому:

void \_\_fastcall

TForm1::Button2Click(TObject \*Sender) {

Graphics::TBitmap \*ptBitmap;

ptBitmap=new Graphics::TBitmap();

char \*pchResType = "RT\_BITMAP";

TResourceStream\* ptRes =

new TResourceStream((int)HInstance, "MYBMP",pchResType);

ptBitmap->LoadFromStream(ptRes);

Image1->Picture->Assign((TPersistent\*)ptBitmap);

delete ptRes;

delete ptBitmap;

}

**Редактирование картинок из ресурсов и отображение отредактированных картинок.**

Следующие примеры показывают как можно извлекать ресурс картинки и внести в него коррективы. Зная структуру файла картинки, например, таким образом можно сделать надписи на картинках или наложить на рисунок из ресурса другой рисунок и т.п.

void \_\_fastcall

TForm1::Button2Click(TObject \*Sender) {

//Тип ресурса

char \*pchResType = "RT\_BITMAP";

//Поиск ресурса

HRSRC hSrsrc=FindResource(NULL,"MYBMP",pchResType);

if(!hSrsrc)

{ShowMessage("Ресурс не найден"); return;}

//Загрузка ресурса в память

HGLOBAL hGlobal=LoadResource(NULL,hSrsrc);

if(!hGlobal)

{ShowMessage("Ресурс не загружен"); return;}

//Фиксация ресурса или получения указателя на ресурс

void \*pvResPtr=LockResource(hGlobal);

if(!pvResPtr)

{ShowMessage("Ресурс не зафиксирован"); return;}

TMemoryStream \* ptMStream=new TMemoryStream();

//Размер ресурса

unsigned long vulResourceSize=SizeofResource(NULL,hSrsrc);

//Поток бит помещаем в массив по умолчанию для Stream

ptMStream->WriteBuffer(pvResPtr,vulResourceSize);

//На этом этапе можно сохранить ресурс в памяти

ptMStream->SaveToFile("a0.bmp");

//Буфер для маневра байтами

unsigned char \* puchResourceBuffer =

new unsigned char[vulResourceSize];

//Переносим весь поток в буфер

memcpy(puchResourceBuffer, pvResPtr, vulResourceSize);

//Замена битов рисунка - для примера просто чуть чуть

//испортим картинку. Учитывая, что в файле 256 цветной .bmp

//биты рисунка начинаются с 0x436(1078-го) байта, испортим

//в картинке несколько байт

for(int i=1078; i < 1000; i++) puchResourceBuffer[i]=i;

//Поток на начало

ptMStream->Seek(0,0);

//Сохранить в поток все с изменениями

ptMStream->Write(puchResourceBuffer,vulResourceSize);

//Поток на начало

ptMStream->Position=0;

//Опять сохраним и посмотрим, что получилось

//ptMStream->SaveToFile("a1.bmp");

Image1->Picture->Bitmap->LoadFromStream(ptMStream);

delete Stream;

delete puchResourceBuffer;

}

Аналогично можно поступить и с картинкой из файла project1.res:

void \_\_fastcall

TForm1::Button2Click(TObject \*Sender) {

Graphics::TBitmap\* ptBitmap;

ptBitmap = new Graphics::TBitmap();

//Загружаем ресурс

ptBitmap->LoadFromResourceName((int)HInstance,"Bitmap1");

Image1->Width=ptBitmap->Width;

Image1->Height=ptBitmap->Height;

//Сдесь можно отобразить изображение

//Image1->Picture->Bitmap->Assign(Bitmap);

TMemoryStream \* ptMStream=new TMemoryStream();

ptBitmap->SaveToStream(ptMStream);

//Можно сохранить то, что в потоке

ptMStream->SaveToFile("a1.bmp");

//Поток на начало

ptMStream->Seek(0,0);

//Буфер для манипуляции битами

puchResourceBuffer =

new unsigned char[ptMStream->Size];

ptMStream->Read(puchResourceBuffer,ptMStream->Size);

//Замена битов рисунка - портим рисунок

for(int i=1078; i < 1000; i++) puchResourceBuffer[i]=i;

//Поток на начало

ptMStream->Seek(0,0);

//Сохранить в поток все с изменениями

ptMStream->Write(puchResourceBuffer,ptMStream->Size);

//Поток на начало

ptMStream->Position=0;

//Можно посмотреть, что получилось

Stream->SaveToFile("a1.bmp");

//Отобразить

Image1->Picture->Bitmap->LoadFromStream(Stream);

delete Bitmap;

delete Stream;

delete puchResourceBuffer;

}

## Иконки в ресурсах

Хранение иконок в ресурсах ничем не отличается от хранения рисунков.

Используем иконку, поместив ее в ресурс через файл projeсt1.res, а затем с использованием файла a.res.

Создадим новый проект, на форму которого для данного примера достаточно поместить два компонента Button. Не компилируя проект, откроем файл projeсt1.res в Image Editor. Там уже есть ветка Icon с именем иконки MAINICON. Воспользовавшись контекстным меню пункт New/Icon, создаем новую иконку и переименуем ее в MAINICON1. Двойным кликом на имене иконки откроем графический редактор и нарисуем любой рисунок. Помним, что цвет левого нижнего пиксела рисунка по умолчанию является цветом прозрачности. Закроем редактор, сохранив изменения в файле projeсt1.res.

Создадим обработчик нажатия для кнопки 2 и запишем следующий код:

void \_\_fastcall

TForm1::Button2Click(TObject \*Sender)

{

HWND h=LoadIcon(HInstance,"MAINICON1");

if(h == 0) ShowMessage("Иконка не загружена");

else

{

Application->Icon->Handle=h;

Application->Icon->SaveToFile("a.ico");

}

}

Запускаем приложение. При нажатии кнопки 2 иконка меняется не только на форме окна приложения, но и в панели задач.

Следующий код показывает как можно получить информацию об иконке и как создается новая иконка, на основании структуры HICON. Кроме того здесь также показано как скопировать иконку в буфер обмена.

\_ICONINFO tIconInfo;

//Загружаем нашу иконку

HICON hIcon=LoadIcon(HInstance,"MAINICON1");

//Копируем информацию из нее в структуру HICON

GetIconInfo(hIcon,&tIconInfo);

//По информации в структуре создаем новую иконку

HICON hIcon1=CreateIconIndirect(&tIconInfo);

//Записываем иконку в буфер

BOOL fOpen = ::OpenClipboard(NULL);

if(fOpen)

{

::EmptyClipboard();

//Как ни странно а указатель hbmColor на биты иконки

::SetClipboardData(CF\_BITMAP, tIconInfo.hbmColor);

::CloseClipboard();

}

//Обязательно удаляем что создано CreateIconIndirect()

DestroyCursor(hIcon1);

Структура ICONINFO общая для иконок и курсоров.

typedef struct \_ICONINFO

{

BOOL fIcon;

DWORD xHotspot;

DWORD yHotspot;

HBITMAP hbmMask;

HBITMAP hbmColor;

}ICONINFO;

Для рассмотрения использования отдельного файла ресурса для работы с иконками в проект внесем некоторые изменения. В обработчиках событий ButtonClick запишем коды, приведенные ниже. После компиляции и запуска приложения нажимаем первую кнопку и закрываем приложение. Добавляем к проекту файл "a.rc". Вновь компилируем и запускаем приложение и нажимаем вторую кнопку - иконка меняется на нарисованную ранее (сохранена на предыдущем шаге) и в форме окна приложения, и в панели задач.

void \_\_fastcall

TForm1::Button1Click(TObject \*Sender) {

TStringList\* ptSList = new TStringList();

ptSList->Clear();

ptSList->Add("MYICON RT\_ICON \"a.ico\"");

ptSList->SaveToFile("a.rc");

delete ptSList;

}

void \_\_fastcall

TForm1::Button2Click(TObject \*Sender) {

char \*pchResType = "RT\_ICON";

TResourceStream\* ptRes =

new TResourceStream((int)HInstance, "MYICON",pchResType);

//Можно и не сохранять

ptRes->SaveToFile("a.ico");

Form1->Icon->LoadFromStream(ptRes);

Application->Icon=Form1->Icon;

delete ptRes;

}

## Курсоры в ресурсах

Использование ресурсов для манипуляций с курсорами во многом аналогично работе с иконками, но здесь есть и существенное отличие, речь о котором пойдет ниже.

Также как и ранее, сначала создадим курсор в файле project1.res, а затем в файле a.res. Создадим новый проект на которой для данного примера поместим шесть компонент Button. Не компилируя проект, откроем projeсt1.res в Image Editor. По аналогии с созданием иконки в предыдущем примере, создадим курсор с именем MAINCURSOR и нарисуем его (любой рисунок, например черную стрелку). Закроем редактор сохранив изменения в файле projeсt1.res.

В секции private файла Unit1.h определим функцию vLoadCursor(). Эта функция необходима только для того, чтобы не повторять один и тотже код.

private:

void \_\_fastcall vLoadCursor(HCURSOR hCursor);

Тело функции опишем в файле "UNIT1.cpp" следующим образом:

void \_\_fastcall

TForm1::vLoadCursor(HCURSOR hCursor) {

if(hCursor == 0) ShowMessage("Курсор не загружен");

else {

Screen->Cursors[1] = hCursor;

Form1->Cursor =(TCursor)1;

}

}

Этот код показывает, что если удалось получить хэндл курсора, то его можно и подключить через массив Cursors свойств Screen.

Создадим обработчик нажатия для кнопки 2 и запишем следующий код:

void \_\_fastcall

TForm1::Button2Click(TObject \*Sender) {

HWND hCursor=LoadCursor(GetModuleHandle(NULL),"MAINCURSOR");

vLoadCursor(hCursor);

}

Откомпилируем и запустим приложение - нарисованный курсор появляется при нажатии на кнопку 2.

Для кнопки 3 создадим другой способ загрузки курсора.

void \_\_fastcall TForm1::Button3Click(TObject \*Sender) {

HCURSOR hCursor=LoadImage

(GetModuleHandle(NULL),

"MAINCURSOR",IMAGE\_CURSOR,0,0,

LR\_COPYFROMRESOURCE | LR\_CREATEDIBSECTION);

vLoadCursor(hCursor);

}

Курсор загружается как и при нажатии на кнопку 2. О функции LoadImage() речь также уже шла.

Перейдем к использованию отдельного .res файла.

Для кнопки 1 напишем уже знакомый (новый только идентификатор ресурса RT\_CURSOR) код:

void \_\_fastcall TForm1::Button1Click(TObject \*Sender) {

TStringList\* ptSList = new TStringList();

ptSList->Clear();

ptSList->Add("MAINCURSOR1 RT\_CURSOR \"Hand-i.cur\"");

ptSList->SaveToFile("a.rc");

delete ptSList;

}

Выполним этот код и в директории проекта будет создан файл "a.rc", содержащий строку:

MyCur RT\_CURSOR "Hand-i.cur"

На данном этапе отличия от предыдущего примера нет.

Скопируем в директорию проекта файл "Hand-i.cur" из директории "C:\Windows\Cursors\" и добавим к проекту файл "a.rc" (Project/Add to Project), а в обработчиках Button4Click напишем код:

void \_\_fastcall

TForm1::Button4Click(TObject \*Sender) {

char \*pchResType = "RT\_CURSOR";

TResourceStream\* ptRes =

new TResourceStream((int)HInstance, "MAINCURSOR1",pchResType);

//Можно сохранить курсор

ptRes->SaveToFile("b.cur");

delete ptRes;

//Можно загрузить из файла

HCURSOR hCursor = LoadImage(0,"b.cur",

IMAGE\_CURSOR,0,0,LR\_LOADFROMFILE);

//Актуализируем курсор

vLoadCursor(hCursor);

}

При запуске приложения все работает, курсор формы сменился со стрелки на руку.

Как загрузить курсор не сохраняя его в файле на диске? До этого и иконки, и картинки можно было загрузить и использовать прямо из потока. Для курсоров такой возможности нет.

Остается один выход - либо хранить курсоры только в Файле ресурсов приложения, либо пойти по достаточно сложному пути, который описан в обработчике события нажатия кноки пять (Button5Click).

void \_\_fastcall

TForm1::Button5Click(TObject \*Sender) {

////////////////////////////////////////////////////////////

// В Ы Б О Р Р Е С У Р С А В Б У Ф Е Р

////////////////////////////////////////////////////////////

char \*pchResType = "RT\_CURSOR";

//Поиск ресурса

HRSRC hSrsrc=FindResource(NULL,"MAINCURSOR1",pchResType);

if(!hSrsrc)

{ShowMessage("Ресурс не найден"); return;}

//Загрузка ресурса в память

HGLOBAL hGlobal=LoadResource(NULL,hSrsrc);

if(!hGlobal)

{ShowMessage("Ресурс не загружен"); return;}

//Фиксация ресурса или получения указателя на ресурс

void \*pvResPtr=LockResource(hGlobal);

if(!pvResPtr)

{ShowMessage("Ресурс не зафиксирован"); return;}

////////////////////////////////////////////////////////////

// К О П И Р У Е М П О Т О К В С В О Й Б У Ф Е Р

////////////////////////////////////////////////////////////

//Так как указатели на void не поддерживают арифметики

//приходится копировать биты ресурса в поток TMemoryStream

//а оттуда в свой буфер

TMemoryStream \* Stream=new TMemoryStream();

unsigned long vulResourceSize=SizeofResource(NULL,hSrsrc);

//Поток бит помещаем в массив по умолчанию для Stream

Stream->WriteBuffer(pvResPtr,vulResourceSize);

//На этом этапе можно сохранить курсор и затем

//загрузить его если надо из файла.

Stream->SaveToFile("w.cur");

//Но цель сейчас обойтись без файла

// Определяем буферы байт для всего потока ресурса.

//Там всего как в любом курсоре 32\*32 - 326 байт

char vrgchResourceBuffer[326];

//Можно объявить и так

//unsigned char \* puchResourceBuffer =

// new unsigned char[ResourceSize];

memcpy(vrgchResourceBuffer, pvResPtr, vuliResourceSize);

////////////////////////////////////////////////////////////

// Ф О Р М И Р У Е М Г О Р Я Ч У Ю Т О Ч К У

////////////////////////////////////////////////////////////

unsigned int viHotSpootX=vrgchResourceBuffer[10]+

10\*vrgchResourceBuffer[11];

unsigned int viHotSpootY=vrgchResourceBuffer[12]+

10\*vrgchResourceBuffer[13];

////////////////////////////////////////////////////////////

// Ф О Р М И Р У Е М М А С С И В Ы М А С О К

////////////////////////////////////////////////////////////

// Маски начинаются с адреса 70 и 198 и содержат по 128 байт

char vrgchResourceBuffer1[128];

char vrgchResourceBuffer2[128];

for(int i=0;i < 32;i++)

{

for(int j=0;j < 4;j++)

{

vrgchResourceBuffer1[j+i\*4]=vrgchResourceBuffer[198-4-i\*4+j];

vrgchResourceBuffer2[j+i\*4]=vrgchResourceBuffer[326-4-i\*4+j];

}

}

//Можго пойти по друглму пути c применением указателей

/\*

void \*pvBuf=&vrgchResourceBuffer[70];

memcpy(vrgchResourceBuffer1,pvBuf,128);

pvBuf=&vrgchResourceBuffer[198];

memcpy(vrgchResourceBuffer2,pvBuf,128);

// Переворачиваем маски - хранятся по 4 байта строка

//первая строка маски есть последняя строка массива

for(int i=0;i < 16;i++)

{

for(int j=0;j < 4;j++)

{

char ch1=vrgchResourceBuffer1[j+i\*4];

char ch2=vrgchResourceBuffer2[j+i\*4];

vrgchResourceBuffer1[j+i\*4]=vrgchResourceBuffer1[128-4-i\*4+j];

vrgchResourceBuffer2[j+i\*4]=vrgchResourceBuffer2[128-4-i\*4+j];

vrgchResourceBuffer1[128-4-i\*4+j]=ch1;

vrgchResourceBuffer2[128-4-i\*4+j]=ch2;

}

}

\*/

////////////////////////////////////////////////////////////

// С О З Д А Е М И З А Г Р У Ж А Е М К У Р С О Р

////////////////////////////////////////////////////////////

HCURSOR hCursor=CreateCursor(NULL,viHotSpootX,viHotSpootY,32,32,

vrgchResourceBuffer2,vrgchResourceBuffer1);

//Наконец то загрузили

vLoadCursor(hCursor);

}

## Строки символов в ресурсах

По аналогии с предыдущими примерами, поместим на форме компонент Memo и два компонента Button. В обработчиках ButtonClick напишем код:

void \_\_fastcall

TForm1::Button1Click(TObject \*Sender) {

TStringList\* ptSList = new TStringList();

ptSList->Clear();

ptSList->Add("MyString RT\_STRING Begin \"Моя строка\" End");

ptSList->SaveToFile("a.rc");

delete ptSList;

}

void \_\_fastcall

TForm1::Button2Click(TObject \*Sender) {

char \*pchResType = "RT\_STRING";

TtResourceStream\* ptRes =

new TtResourceStream((int)HInstance, "MyString",pchResType);

Memo1->Lines->LoadFromStream(ptRes);

delete ptRes;

}

Выполнив ту же последовательность действий как и в предыдущих примерах получим в компоненте Memo при нажатии кнопки два строку:

"Моя строка"

## HTML документы в ресурсах

Помещение HTML документов (файлов .html, .htm ...) в ресурсы полностью аналогично с примером помещения в ресурсы и использования .exe файлов. Поэтому здесь приводятся лишь коды приложения, форма которого имеет компонент TMemo и два компонента TButton, и которое выводит в браузере документ "a.html", а текст кода документа в компоненте Memo.

void \_\_fastcall

TForm1::Button1Click(TObject \*Sender) {

TStringList\* ptSList = new TStringList();

ptSList->Clear();

ptSList->Add("MyHTML RT\_HTML \"a.html\"");

ptSList->SaveToFile("a.rc");

delete ptSList;

}

void \_\_fastcall

TForm1::Button2Click(TObject \*Sender) {

char \*pchResType = "RT\_HTML";

TtResourceStream\* ptRes =

new TtResourceStream((int)HInstance, "MyHTML",pchResType);

Memo1->Lines->LoadFromStream(ptRes);

Memo1->Lines->SaveToFile("b.html");

ShellExecute(0,NULL,"b.html", NULL,NULL,SW\_tResTORE);

delete ptRes;

}

## .wav и другие музыкальные файлы в ресурсах

По аналогии с предыдущими примерами, оформим создание файла a.rc для формирования a.res в приложении, поместив на форме компонент TButton. Откомпилируем и выполним приложение, предварительно поместив в директорию приложения любой музыкальный файл, например a.wav.

void \_\_fastcall

TForm1::Button1Click(TObject \*Sender) {

TStringList\* ptSList = new TStringList();

ptSList->Clear();

ptSList->Add("MYWAV SOUND \"mywav.wav\"");

ptSList->SaveToFile("a.rc");

delete ptSList;

}

Заметим, что идентификатора ресурса SOUND нет в списке идентификаторов ресурсов. Это говорит о том, что в файл ресурсов можно помещать и произвольную информацию - вопрос лишь в том, как ее потом оттуда извлечь и использовать.

Добавим на форму еще один компонент TButton и запишем код:

void \_\_fastcall

TForm1::Button2Click(TObject \*Sender) {

sndPlaySound("MyWav", SND\_SYNC | SND\_RESOURCE);

PlaySound("MyWav",NULL, SND\_SYNC | SND\_RESOURCE);

}

Содержимое файла, помещенное в ресурс, воспроизводится дважды (для обеспечения этой возможности добавлен флаг SND\_SYNC). Однако, если повторить все тоже, но заменив тип ресурса "SOUND", например на "SOUND1", эти функции откажутся воспроизводить ресурс. Таким образом, хотя "SOUND" не определен как тип ресурса в файле winuser.h как некоторая константа (например MAKEINTRESOURCE(x) - где x некоторое число), функции считают, что в исполняемом файле находится ресурс именно с типом "SOUND".

Для приведенных ниже кодов ресурс может иметь любой идентификатор отличный от "SOUND". Здесь используются уже знакомые по предыдущим параграфам потоки, которые позволяют найти практически произвольный ресурс и извлечь его из исполняемого модуля как двоичный код.

void \_\_fastcall

TForm1::Button1Click(TObject \*Sender) {

TStringList\* ptSList = new TStringList();

ptSList->Clear();

//Заменили тип ресурса SOUND на SOUND1

ptSList->Add("MYWAV SOUND1 \"mywav.wav\"");

ptSList->SaveToFile("a.rc");

delete ptSList;

}

Используем TResourceStream.

void \_\_fastcall

TForm1::Button2Click(TObject \*Sender) {

char \*pchResType = "SOUND1";

TResourceStream\* ptRes =

new TResourceStream((int)HInstance, "MyWav",pchResType);

//На это этапе можно сохранить

ptRes->SaveToFile("MyWav1.wav");

//Здесь возможно запустить системный плеер

ShellExecute(Handle,"open","MyWav1.wav",NULL,NULL,SW\_SHOWNORMAL);

//Можно воспроизвести из памяти

unsigned char \* puchResourceBuffer =

new unsigned char[ptRes->Size];

ptRes->Read(puchResourceBuffer,ptRes->Size);

sndPlaySound(puchResourceBuffer, SND\_SYNC | SND\_MEMORY);

delete ptRes;

}

Используем TMemoryStream и функции поиска и фиксации ресурса.

void \_\_fastcall

TForm1::Button2Click(TObject \*Sender)

{

HRSRC hRsrc=FindResource(NULL,"MyWav","SOUND1");

if(!hRsrc)

{ShowMessage("Ресурс не найден"); return;}

//Загрузка ресурса в память

HGLOBAL hGlobal=LoadResource(NULL,hRsrc);

if(!hGlobal)

{ShowMessage("Ресурс не загружен"); return;}

//Фиксация ресурса или получения указателя на ресурс

void \*pvResPtr=LockResource(hGlobal);

if(!pvResPtr)

{ShowMessage("Ресурс не зафиксирован"); return;}

TMemoryStream \* ptMStream=new TMemoryStream();

//Размер ресурса

unsigned long vulResourceSize=SizeofResource(NULL,hRsrc);

//Поток бит помещаем в массив по умолчанию для Stream

ptMStream->WriteBuffer(pvResPtr,vulResourceSize);

//На этом этапе можно сохранить

ptMStream->SaveToFile("mywav2.wav");

//Здесь возможно запустить системный плеер

ShellExecute(Handle,"open","MyWav2.wav",NULL,NULL,SW\_SHOWNORMAL);

//Воспроизвести из файла

sndPlaySound("mywav2.wav", SND\_SYNC | SND\_FILENAME);

unsigned char \* puchResourceBuffer =

new unsigned char[vulResourceSize];

//Переносим весь поток в буфер

memcpy(puchResourceBuffer, pvResPtr, vulResourceSize);

//Воспроизвести из памяти

sndPlaySound(puchResourceBuffer, SND\_SYNC | SND\_MEMORY);

}

Аналогично .wav файлам можно манипулировать и с любыми другими музыкальными файлами.

**Замечания**:

1. В некоторых версиях C++ Builder для того, чтобы обновить файл ресурса, его нужно сначала удалить из проекта (Project, Remove from Project), а затем добавить снова (Project, Add to Project). Просто при добавлении нового ресурса в файл он не линкуется в проекте и при запуске программы возникает исключение EResNotFound (“ресурс не найден”).

2. Также полезными могут оказаться следующие функции для работы с ресурсами (только список):

* FindResource - Ф-ция для поиска ресурса.
* SizeofResource - Ф-ция для получения размера ресурса.
* LoadResource - Ф-ция для загрузки ресурса.
* LockResource - Ф-ция для фиксирования ресурса в памяти.
* EnumResourceTypes - Ф-ция для получения списка типов ресурсов.
* EnumResourceNames - Ф-ция для получения списка имён ресурсов.
* BeginUpdateResource - Ф-ция для обновления ресурсов.
* UpdateResource - Ф-ция для замены ресурсов.
* EndUpdateResource - Ф-ция для завершения обновления ресурсов.

Например, здесь мы выполняем поиск ресурса по имени с его последующей загрузкой:

HRSRC hRes = FindResource(NULL, "Bitmap1", RT\_BITMAP);

HGLOBAL hResLoad = LoadResource(NULL, hRes);