Что нужно сделать, чтобы добавить кнопку в заголовок формы?

- Нам нужна функция для рисования кнопки
- Нужно перерисовывать кнопку, когда кнопка видима/невидима, когда форма активизируется, изменяет размеры или перемещается
- Нужно иметь событие щелчка мыши на кнопке

Рисуем TRect как кнопку

Вы не можете помещать объекты **VCL** в неклиентскую область окна, но можно нарисовать его и моделировать вид кнопки. Чтобы выполнить рисование в области заголовка окна, нужно сделать три важных вещи:

1. Нужно получить текущие координаты окна и размер изображения, чтобы знать в какой области рисовать и какого размера будет прямоугольник
2. Затем нужно определить структуру **TRect** с определенным размером и позицией в пределах области заголовка
3. Наконец, нужно нарисовать **TRect** как кнопку, затем добавить любое изображение или текст, которое должно быть на поверхности кнопки

Все это выполняется в одном вызове процедуры **DrawTitleButton**, который приведен ниже:

```
procedure TTitleBtnForm.DrawTitleButton;  var    bmap : TBitmap; { точечный рисунок - 16 X 16 : 16 цветов }    XFrame,  { размеры X и Y Sizeable области рамки окна }    YFrame, XTtlBit, {размеры X и Y изображений в заголовке }    YTtlBit  : Integer;  begin    {Получить размер рамки формы и изображений в области заголовка }    XFrame  := GetSystemMetrics(SM_CXFRAME);    YFrame  := GetSystemMetrics(SM_CYFRAME); XTtlBit := GetSystemMetrics(SM_CXSIZE);    YTtlBit := GetSystemMetrics(SM_CYSIZE); {$IFNDEF WIN32}      TitleButton := Bounds(Width - (3 * XTtlBit) - ((XTtlBit div 2) - 2),        YFrame - 1,                    XTtlBit + 2,                    YTtlBit + 2);     {$ELSE}     { для Delphi 2.0 }      if (GetVerInfo = VER_PLATFORM_WIN32_NT) then       TitleButton := Bounds(Width - (3 * XTtlBit) - ((XTtlBit div 2) - 2),                    YFrame - 1,   XTtlBit + 2,                    YTtlBit + 2)     else       TitleButton := Bounds(Width - XFrame - 4*XTtlBit + 2,                    XFrame + 2,                    XTtlBit + 2, YTtlBit + 2);    {$ENDIF}       Canvas.Handle := GetWindowDC(Self.Handle);    { Получить контекстное устройство для рисования }    try      { нарисовать кнопку в TRect } DrawButtonFace(Canvas, TitleButton, 1, bsAutoDetect, False, False, False);      bmap :=
```

TBitmap.Create; bmap.LoadFromFile('help.bmp'); with TitleButton do {$IFNDEF WIN32} Canvas.Draw(Left + 2, Top + 2, bmap); {$ELSE} if (GetVerInfo = VER_PLATFORM_WIN32_NT) then Canvas.Draw(Left + 2, Top + 2, bmap) else Canvas.StretchDraw(TitleButton, bmap); {$ENDIF} finally ReleaseDC(Self.Handle, Canvas.Handle); bmap.Free; Canvas.Handle := 0; end; end;

Шаг первый выполняется, делая четыре вызова функций *WinAPI* **GetSystemMetrics**, запрашивая систему о ширине и высоте окна (
SM_CXRFAME
и
SM_CYFRAME
) и размер точечных рисунков, содержащихся в области заголовка (
SM_CXSIZE
и
SM_CYSIZE
).

В шаге 2 выполняется функция **Bounds**, которая возвращает **TRect** с размерами и параметрами позиции. Заметьте, что здесь используются некоторые условные директивы компилятора. Это потому, что размер кнопок в заголовках Windows 95 и Windows 3.1 отличаются, так что и размер должен устанавливаться по-другому. Теперь это можно использовать в любой версии Windows. Однако, так как Windows NT UI то же самое, что и Windows 3.1, нужно определить, является ли это Windows NT и если так, нужно определить **T**
Rect
точно так же, как и для Windows 3.1.

Чтобы выполнить шаг 3, делаем вызов **DrawButtonFace** модуля *Buttons*, чтобы нарисовать кнопки в пределах
TRect
, который мы уже определили. Как только добавим, нужно включить код для рисования точечного рисунка на кнопке. Здесь тоже применяется условная директива компилятора для различных версий Windows. Потому что точечный рисунок 16 X 16 пикселей может быть слишком большим для Windows 95. Так что здесь используется
StretchDraw
, чтобы растянуть рисунок по размеру кнопки.

Ловушки событий Drawing и Painting

Вы должны удостовериться, что кнопка будет видима каждый раз после перерисовки

формы. Перерисовка происходит в ответ на активацию и изменение размеров. Если не будет обновления кнопки, она будет теряться, когда происходить перерисовка формы. Нужно написать обработчик события, которые выполнят эти действия и будут обновлять кнопку. Следующие четыре процедуры обрабатывают события нажатия кнопки и обновления:

```
{Paint triggering events}  procedure TForm1.WMNCActivate(var Msg : TWMNCActivate);
begin  Inherited;  DrawTitleButton;  end;  procedure TForm1.FormResize(Sender: TObject);
begin  Perform(WM_NCACTIVATE, Word(Active), 0);  end;  {Painting events}  procedure
TForm1.WMNCPaint(var Msg : TWMNCPaint);  begin  Inherited;  DrawTitleButton;  end;
procedure TForm1.WMSetText(var Msg : TWMSetText);  begin  Inherited;  DrawTitleButton;
end;
```

Каждый раз, когда происходит одно из этих событий, оно в свою очередь вызывает процедуру **DrawTitleButton**. Это будет гарантировать, что кнопка будет всегда видима в области заголовка. Здесь используется обработчик события формы **OnResize**, чтобы вынудить выполняться **WM_NCACTIVATE**.

Обработка щелчков мыши

Теперь, когда получен код, который рисует кнопку, нужно обработать щелчки мыши на кнопке. Это делается двумя процедурами. Первая процедура проверяет, был ли щелчок мыши в области кнопки, а вторая фактически исполняет выполнение кода, связанное с нажатием кнопки:

```
{ мыше-зависимые процедуры }  procedure TForm1.WMNCHitTest(var Msg :
TWMNCHitTest);  begin  Inherited;  { Проверяем, нажата ли кнопка в области заголовка }
  with Msg do  if PtInRect(TitleButton, Point(XPos - Left, YPos - Top)) then  Result :=
htTitleBtn;  end;  procedure TForm1.WMNCLButtonDown(var Msg : TWMNCLButtonDown);
begin  inherited;  if (Msg.HitTest = htTitleBtn) then  ShowMessage('Вы нажали новую
кнопку');  end;
```

Первая процедура **WMNCHitTest** сообщение нажатия, чтобы определить, где была нажата мышь в неклиентской области. В этой процедуре проверяется, что точка нажатия была в пределах нашего **TRect**, используя функцию **PtInRect**.
Если щелчок мыши был выполнен в пределах **TRect**, то результат сообщения устанавливается в *htTitleBtn*, которая является константой и объявлена как

htTitleBtn + 1

.

htSizeLast

- константа проверки нажатия, чтобы проверить где было последнее нажатие.

А теперь объединим все вместе

Теперь давайте посмотрим на полный код в форме:

```
unit Capbtn; interface uses SysUtils, WinTypes, WinProcs, Messages, Classes, Graphics, Controls, Forms, Dialogs, Buttons; type TTitleBtnForm = class(TForm) procedure FormResize(Sender: TObject); private TitleButton : TRect; procedure DrawTitleButton; {Paint-related messages} procedure WMSetText(var Msg : TWMSetText); message WM_SETTEXT; procedure WMNCPaint(var Msg : TWMNCPaint); message WM_NCPAINT; procedure WMNCActivate(var Msg : TWMNCActivate); message WM_NCACTIVATE; {Mouse down-related messages} procedure WMNCHitTest(var Msg : TWMNCHitTest); message WM_NCHITTEST; procedure WMNCLButtonDown(var Msg : TWMNCLButtonDown); message WM_NCLBUTTONDOWN; function GetVerInfo : DWORD; end; var TitleBtnForm: TTitleBtnForm; const htTitleBtn = htSizeLast + 1; implementation {$R *.DFM} procedure TTitleBtnForm.DrawTitleButton; var bmap : TBitmap; {Bitmap to be drawn - 16 X 16 : 16 Colors} XFrame, {X and Y size of Sizeable area of Frame} YFrame, XTtlBit, {X and Y size of Bitmaps in caption} YTtlBit : Integer; begin {Get size of form frame and bitmaps in title bar} XFrame := GetSystemMetrics(SM_CXFRAME); YFrame := GetSystemMetrics(SM_CYFRAME); XTtlBit := GetSystemMetrics(SM_CXSIZE); YTtlBit := GetSystemMetrics(SM_CYSIZE); {$IFNDEF WIN32} TitleButton := Bounds(Width - (3 * XTtlBit) - ((XTtlBit div 2) - 2), YFrame - 1, XTtlBit + 2, YTtlBit + 2); {$ELSE} {Delphi 2.0 positioning} if (GetVerInfo = VER_PLATFORM_WIN32_NT) then TitleButton := Bounds(Width - (3 * XTtlBit) - ((XTtlBit div 2) - 2), YFrame - 1, XTtlBit + 2, YTtlBit + 2) else TitleButton := Bounds(Width - XFrame - 4*XTtlBit + 2, XFrame + 2, XTtlBit + 2, YTtlBit + 2); {$ENDIF} Canvas.Handle := GetWindowDC(Self.Handle); {Get Device context for drawing} try {Draw a button face on the TRect} DrawButtonFace(Canvas, TitleButton, 1, bsAutoDetect, False, False, False); bmap := TBitmap.Create; bmap.LoadFromFile('help.bmp'); with TitleButton do {$IFNDEF WIN32} Canvas.Draw(Left + 2, Top + 2, bmap); {$ELSE} if (GetVerInfo = VER_PLATFORM_WIN32_NT) then Canvas.Draw(Left + 2, Top + 2, bmap) else Canvas.StretchDraw(TitleButton, bmap); {$ENDIF} finally ReleaseDC(Self.Handle, Canvas.Handle); bmap.Free; Canvas.Handle := 0; end; end; {Paint triggering events} procedure TTitleBtnForm.WMNCActivate(var Msg : TWMNCActivate); begin Inherited; DrawTitleButton; end; procedure TTitleBtnForm.FormResize(Sender: TObject); begin Perform(WM_NCACTIVATE, Word(Active), 0); end; {Painting events} procedure TTitleBtnForm.WMNCPaint(var Msg :
```

```
TWMNCPaint); begin Inherited; DrawTitleButton; end; procedure
TTitleBtnForm.WMSetText(var Msg : TWMSetText); begin Inherited; DrawTitleButton; end;
  {Mouse-related procedures} procedure TTitleBtnForm.WMNCHitTest(var Msg :
TWMNCHitTest); begin Inherited; {Check to see if the mouse was clicked in the area of the
button} with Msg do if PtInRect(TitleButton, Point(XPos - Left, YPos - Top)) then
Result := htTitleBtn; end; procedure TTitleBtnForm.WMNCLButtonDown(var Msg :
TWMNCLButtonDown); begin inherited; if (Msg.HitTest = htTitleBtn) then
ShowMessage('Вы нажали новую кнопку'); end; function TTitleBtnForm.GetVerInfo :
DWORD; var verInfo : TOSVERSIONINFO; begin verInfo.dwOSVersionInfoSize :=
SizeOf(TOSVersionInfo); if GetVersionEx(verInfo) then Result := verInfo.dwPlatformID;
{Returns: VER_PLATFORM_WIN32s Win32s on Windows 3.1
VER_PLATFORM_WIN32_WINDOWS Win32 on Windows 95
VER_PLATFORM_WIN32_NT Windows NT } end; end.
```