Стыковка в Лазарусе

Michaël Van Canneyt
29 марта 2014г.
(оригинал на английском языке находится здесь, исходные коды примеров к статье можно скачать отсюда)

перевод Zoltanleo, aka Док

Аннотация

Стыковка - это функция, доступная во многих приложениях. По сути, это означает, что части приложений (обычно панели инструментов или меню) можно перемещать в другие места в окне или даже перемещать в отдельное окно. В этой статье обсуждаются механизмы обеспечения стыковки в приложении LCL.

1. Введение

Традиционно приложение показывает доступные панели инструментов в верхней части главного окна. Однако, когда доступно много панелей инструментов, верхняя часть окна становится переполненной. Вдобавок ко всему, у каждого пользователя есть свои предпочтения, когда дело доходит до размещения панели инструментов: например, панель инструментов форматирования справа, панель инструментов выравнивания слева и панель инструментов рисования в нижней части окна или даже наличие различных панелей инструментов. На экране могут быть размещены не только панели инструментов, но и другие вспомогательные средства при работе с документами или данными.

Чтобы приспособиться ко всему этому, была изобретена стыковка: эта технология позволяет пользователю перетаскивать определенные части пользовательского интерфейса в другое место на экране. В зависимости от того, где он был брошен, он хорошо интегрируется в то окно, на которое его бросили.

LCL также позволяет это сделать, и в этой статье... (*прим.перев*: в оригинале фраза оборвана).

2. Разрешение перетаскивания элементов управления

Поддержка механизма drag & dock аналогична поддержке механизма drag & drop: вместо перетаскивания содержимого элемента управления (перетаскивания элемента из списка, например, в дерево), сам элемент управления перетаскивается по экрану и бросается в другое место. Таким образом, начальная точка drag & dock аналогична начальной точке операций drag & drop.

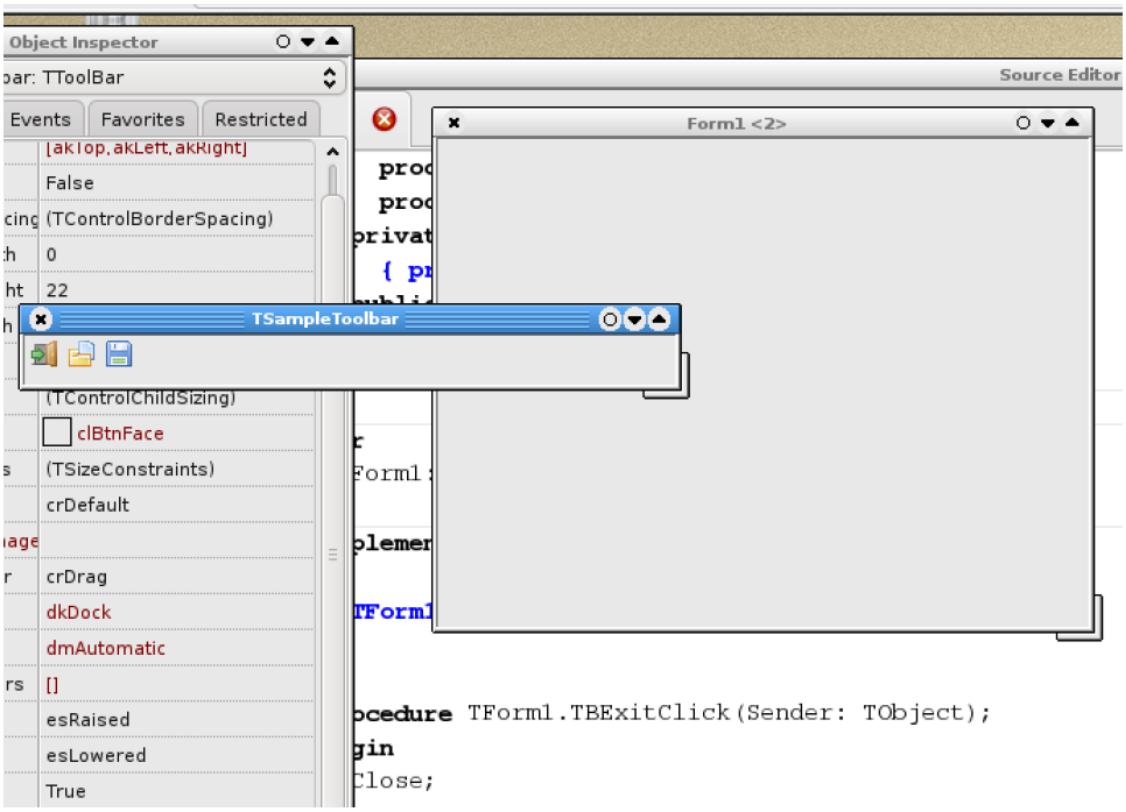
Чтобы сделать элемент управления пристыковываемым, важны 2 свойства: `DragKind` и `DragMode`. Первый по умолчанию задан как `dkDrop`, и должен быть задан как `dkDock`, чтобы указать, что при запуске операции перетаскивания должен перетаскиваться элемент управления, а не его содержимое (хотя можно сделать и то, и другое - это будет объяснено ниже). Свойство `DragMode` должно быть задано как `dmAutomatic`. Это означает, что LCL автоматически начнет операцию перетаскивания, как только увидит, что мышью щелкнули и потащили элемент управления.

Вот и все. Чтобы продемонстрировать это, можно создать небольшое демонстрационное приложение: простую форму с панелью инструментов на ней, содержащую 3 кнопки: одну для выхода из приложения (TBExit), две другие (TBOpen, TBSave), которые просто показывают сообщение:

```
procedure TForm1.TBExitClick(Sender: TObject);
begin
  Close;
end;

procedure TForm1.TBOpenClick(Sender: TObject);
begin
  ShowMessage('Вы нажали кнопку '+(Sender as TComponent).Name);
end;
```

Рисунок 1: Простое перетаскивание панели инструментов



Вышеупомянутые свойства могут быть установлены на панели инструментов, и приложение может быть запущено. Панель инструментов теперь можно перетащить за пределы основной формы, как показано на рисунке 1. Обратите внимание, что окно, в котором отображается панель инструментов, имеет имя компонента панели инструментов. Это вернется позже.

Когда плавающее окно, в котором находится панель инструментов, закрывается, пользователь оказывается без панели инструментов. Это можно легко исправить с помощью popup-меню или меню «Просмотр», в котором панель инструментов может быть отображена или скрыта. Для простой демонстрации в форму помещается кнопка со следующим кодом в обработчике `OnClick`:

```
procedure TForm1.BShowToolbarClick(Sender: TObject);
begin
  TSampleToolbar.Parent:=Self;
  TSampleToolbar.Visible:=True;
end;
```

Щелчок по кнопке после закрытия окна плавающей панели инструментов снова поместит панель инструментов в верхнюю часть окна. В реальном приложении, конечно, этот пункт меню будет отключен, пока отображается панель инструментов.

3. Пристыковка элемента управления к другому элементу управления

Это довольно просто - позволять элементу управления перетаскиваться за пределы формы и перемещаться в окне. Итак, как пристыковать его вдоль одной из сторон формы или в любом месте формы? Для этого многие потомки TWinControl можно сделать стыковочным узлом (DockSite): это означает, что к нему можно прикрепить элемент управления. По умолчанию элементы управления не являются стыковочными узлами. Их можно сделать таковыми, установив для свойства `DockSite` значение `True`.

Сама основная форма не должна быть стыковочным узлом: это позволит пользователю бросать панель инструментов в любое место формы, что, скорее всего, не является намерением программиста.

Обычно панели инструментов располагаются по краям главного окна. Для этого на форму помещаются 4 панели, выровненные по верхнему, левому, нижнему и правому краям формы: для них свойство `Docksite` будет установлено в `True`, а их свойство `AutoSize` будет установлено в `True`: это даст уверенность, что они адаптируют свой размер к размеру прикрепленного к ним элемента управления. Обратите внимание, что в результате панели в дизайнере исчезнут.

В настоящее время это поведение нормально, но отличается от Delphi. Чтобы панели не исчезали в дизайнере, оставьте для свойства `AutoSize` значение `False` и установите для него значение `True` в событии `OnCreate` формы:

```
procedure TForm1.FormCreate(Sender: TObject);
begin
  PTop.AutoSize:=True;
  PBottom.AutoSize:=True;
  PLeft.AutoSize:=True;
  PRight.AutoSize:=True;
end;
```

Теперь панель инструментов можно перетаскивать с одного края формы на другой. Чтобы убедиться, что панель инструментов адаптирует расположение кнопок к той стороне формы, к которой она пристыкована, свойство `align` панели инструментов устанавливается в соответствии с выравниванием панели, на которой она размещена. Это можно сделать в событии `OnDockDrop`, которое выполняется, когда элемент управления пристыкован:

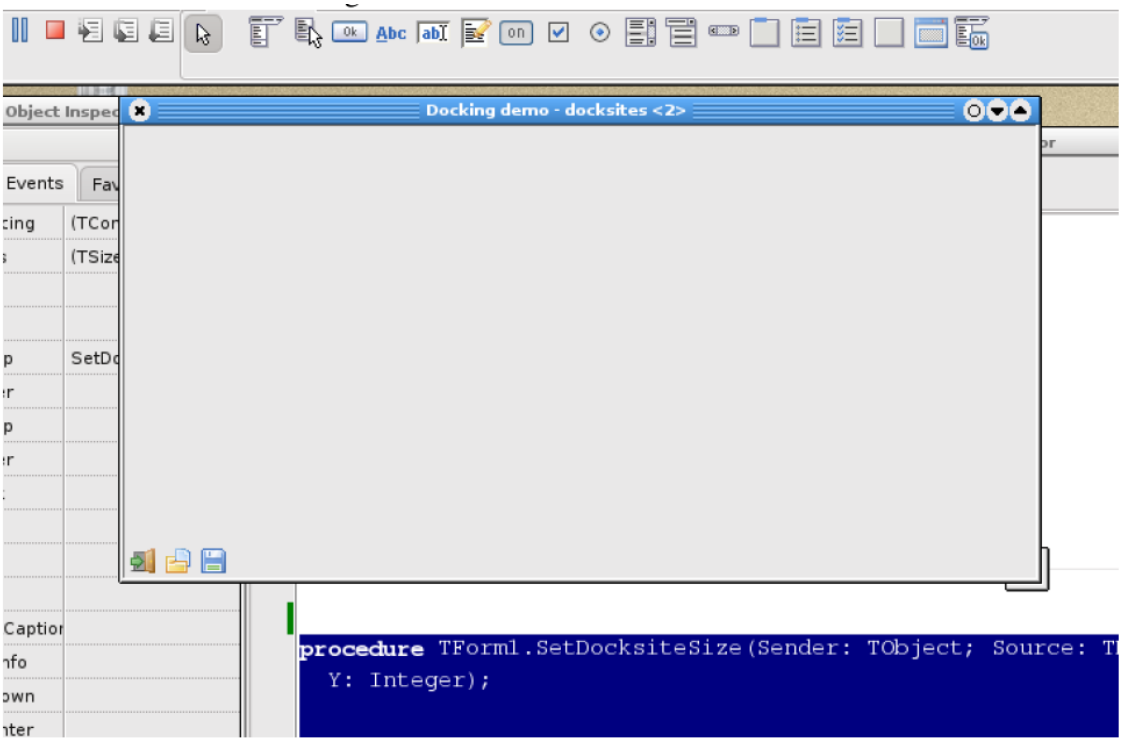
```
procedure TForm1.SetDocksiteSize(Sender: TObject; Source:TDragDockObject; X,Y:
Integer);
Var
  C: TControl;
begin
  C:=(Sender as TControl);
  if Source.Control is TToolbar then
    Source.Control.Align:=C.Align
  else
    Source.Control.Align:=alNone;
end;
```

Для других элементов управления свойству `Align` задано значение `alNone`, поэтому они сохраняют свой естественный размер.
Обратите внимание на параметр `Source` для этого события: он имеет тип `TDragDockObject` и описывает операцию `drag&dock`. Он имеет множество свойств, из которых следующие наиболее интересны для операций стыковки:

- **Control** Перетаскиваемый элемент управления.
- **DragPos** Начальная точка операции перетаскивания.
- **DragTarget** - целевой элемент управления, который мы сейчас перетаскиваем.
- **DockRect** Прямоугольник, указывающий область, в которой будет пристыкован элемент управления, если отпустить мышь.
- **DropAlign** выравнивание для использования при пристыковке элемента управления относительно `DropOnControl`
- **DropOnControl** Элемент управления, уже пристыкованный в стыковочном узле, относительно которого будет пристыкован перетаскиваемый элемент управления.
- **Floating** плавающий в настоящее время элемент управления

Результат показан на рисунке 2:

Рисунок 2: Пристыкованная снизу панель инструментов



4. Предоставление обратной связи пользователю

При упражнениях с примером реализации стыковочного узла можно заметить раздражающий побочный эффект свойства `Autosize`: стыковочный прямоугольник, который обычно является контурами доксайта, имеет нулевую ширину или высоту, потому что у стыковочного узла нет ширины или высоты (в зависит от той стороны формы, по которой стыковочный узел выравнивается).

Это можно исправить в событии `OnDockOver`:

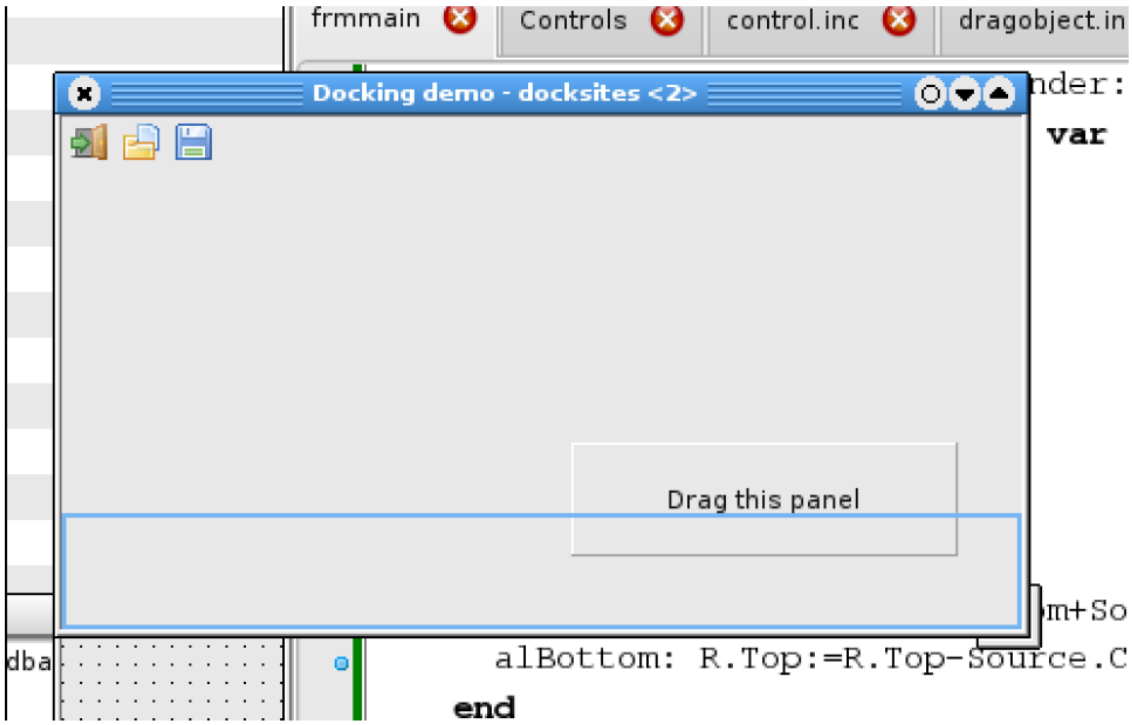
```
procedure TForm1.PTopDockOver(Sender: TObject; Source: TDragDockObject; X,Y: Integer;
State: TDragState; var Accept: Boolean);
Var
  R: TRect;
  C: TControl;
begin
  R:=Source.DockRect;
  C:=Sender as TControl;
  if (R.Bottom-R.Top)<=1 then
    case C.Align of
     alTop: R.Bottom:=R.Bottom+Source.Control.Height;
  alBottom: R.Top:=R.Top-Source.Control.Height;
    end
  else if (R.Right-R.Left)<=1 then
    case C.Align of
    alLeft: R.Right:=R.Right+Source.Control.Width;
   alRight: R.Left:=R.Left-Source.Control.Width;
    end;
  Source.DockRect:=R;
```

```
  end;
```

Свойство `DockRect` объекта `Source` можно изменить, чтобы предоставить пользователю обратную связь в области, где будет пристыкован элемент управления. Приведенный выше код просто увеличивает прямоугольник, чтобы он имел ненулевой размер.

Панель инструментов имеет ширину формы, поэтому прямоугольник будет увеличиваться до размера формы при наведении курсора на левый или правый край. для демонстрации эффекта приведенного выше кода, панель кидается на форму и делается перетаскиваемой посредством установки свойств `DragKind` и `DragMode`. При перетаскивании панели за один из краев (левой или правый) создается прямоугольник, как показано на рисунке 3:

Рисунок 3: Регулировка размера DockRect



Может быть предоставлена дополнительная обратная связь: например, можно повысить чувствительность механизма стыковки: по умолчанию, если мышь перемещается в пределах 10 пикселей от зоны стыковки (это жестко запрограммированное значение), срабатывает обратная связь стыковки. Эта зона может быть увеличена (или уменьшена), а то и удалена вообще: например, можно указать, что стыковочный узел не принимает панели. Это можно сделать в событии `OnGetSiteInfo`, которое запускается при перетаскивании элемента управления. Для панелей по краям формы событие может быть реализовано следующим образом:

```
procedure TForm1.PLeftGetSiteInfo(Sender: TObject; DockClient: TControl; var
InfluenceRect: TRect; MousePos: TPoint; var CanDock: Boolean);
begin
  CanDock:=DockClient is TToolbar;
  if CanDock then
    case (Sender as TControl).Align of
  alLeft,alRight: InflateRect(InfluenceRect,20,0);
  alBottom,alTop: InflateRect(InfluenceRect,0,20);
    end;
end;
```

Параметр `CanDock` можно использовать, чтобы указать, что стыковочный узел примет перетаскиваемый элемент управления. `InfluenceRect` - это ограничивающий прямоугольник стыковочного узла, увеличенный на 10 пикселей. Приведенный выше код добавляет к этому еще 20 пикселей. Если параметр `CanDock` имеет значение `True` (его значение при входе) и элемент управления перетаскивается в `InfluenceRect`, будет показан прямоугольник стыковки.

Даже если параметр `CanDock` имеет значение `true` и отображается прямоугольник пристыковки, все же можно отклонить стыковку элемента управления в событии `OnDockOver`: если для параметра `Accept` этого события задано значение `False`, стыковка будет отклонена. Это можно использовать для точной настройки стыковки: там, где `OnGetSiteInfo` просто указывает, принимает ли стыковочный узел элемент управления для стыковки, событие `OnDockOver` может ограничить стыковку определенными областями стыковочного узла (как это происходит, например, в IDE Delphi, когда кто-то пытается закрепить несколько окон инструментов друг над другом).

5. Управление началом операции перетаскивания

Игра с перетаскиваемой панелью на форме быстро покажет, что щелчок по панели приведет к ее всплытию. Это происходит потому, что, когда `DragMode` установлен в `dmAutomatic`, событие нажатия клавиши мыши запускает операцию перетаскивания, и это довольно неприятно. К счастью, это можно исправить с помощью 2 свойств глобального объекта Mouse:

- `DragImmediate` Если установлено значение `True`, событие нажатия клавиши мыши на любом перетаскиваемом элементе управления запустит операцию перетаскивания. Если установлено значение `False`, операция перетаскивания элемента управления начнется только после того, когда мышь будет перетащена на количество пикселей, указанных в свойстве `DragTreshold`.
- `DragTreshold` Расстояние, на которое пользователь должен перетащить мышь для начала операции перетаскивания, когда `DragImmediate` имеет значение `False`. По умолчанию это 5 пикселей.

Чтобы снова сделать панель кликабельной, достаточно следующего кода в событии формы `OnCreate`:

```
procedure TForm1.FormCreate(Sender: TObject);
begin
  Mouse.DragImmediate:=False;
  Mouse.DragThreshold:=50;
end;
```

6. Ручные перетаскивание и пристыковка

Есть еще один способ сделать панель интерактивной, не вмешиваясь в объект `Mouse`. Это тоже более мощный механизм. Свойству `DragMode` можно присвоить значение `dmManual`, это означает, что операции перетаскивания будут запускаться вручную с использованием метода `BeginDrag` `TControl`. Этот способ выглядит следующим образом:

```
procedure BeginDrag(Immediate: Boolean; Threshold: Integer = -1);
```

При вызове этот метод начнет операцию перетаскивания, если свойство `Immediate` имеет значение `True`. Если `Immediate` имеет значение `False`, то операция перетаскивания начнется после того, как мышь будет перетащена на количество пикселей, указанных в свойстве `Treshold`. Если `Treshold` равен -1, то используется значение `Mouse.DragTreshold`.

Теперь этот метод можно использовать для запуска операции перетаскивания в событии `OnMouseDown` панели:

```
procedure TForm1.Panel1MouseDown(Sender: TObject; Button: TMouseButton;
Shift: TShiftState;X, Y: Integer);
begin
  if (Button=mbLeft) and (ssCtrl in Shift) then
  Panel1.BeginDrag(False,10);
end;
```

Таким образом, нет большой разницы с предыдущей ситуацией, когда `DragMode=dmAutomatic`.

Это становится интересным только в том случае, если кто-то хочет переключиться, например, между операцией `Drag&Drop` и операцией `Drag&Dock`: следующий код запускает операцию `Drag&Dock`, только если при перетаскивании мыши нажимается клавиша `Ctrl`. Если клавиша `Ctrl` не нажата, запускается операция `Drag&Drop`:

```
procedure TForm1.Panel1MouseDown(Sender: TObject; Button: TMouseButton;
Shift: TShiftState; X, Y: Integer);
begin
  if (Button=mbLeft) then
  begin
    if (ssCtrl in Shift) then
      Panel1.DragKind:=dkDock
    else
      Panel1.DragKind:=dkDrag;
    Panel1.BeginDrag(False,10);
  end;
end;
```

Очевидно, что там для панели нечего перетаскивать, но код, подобный приведенному выше, может быть интересен в гридах, listboxe, listview или treeview, где пользователь может просто перетаскивать элементы или может перетаскивать и пристыковывать элемент управления с помощью клавиши `Ctrl`.

Можно не только запускать операцию перетаскивания вручную, но также можно вручную сделать элемент управление плавающим с помощью операции `ManualFloat`:

```
function ManualFloat(TheScreenRect: TRect; KeepDockSiteSize: Boolean): Boolean;
```

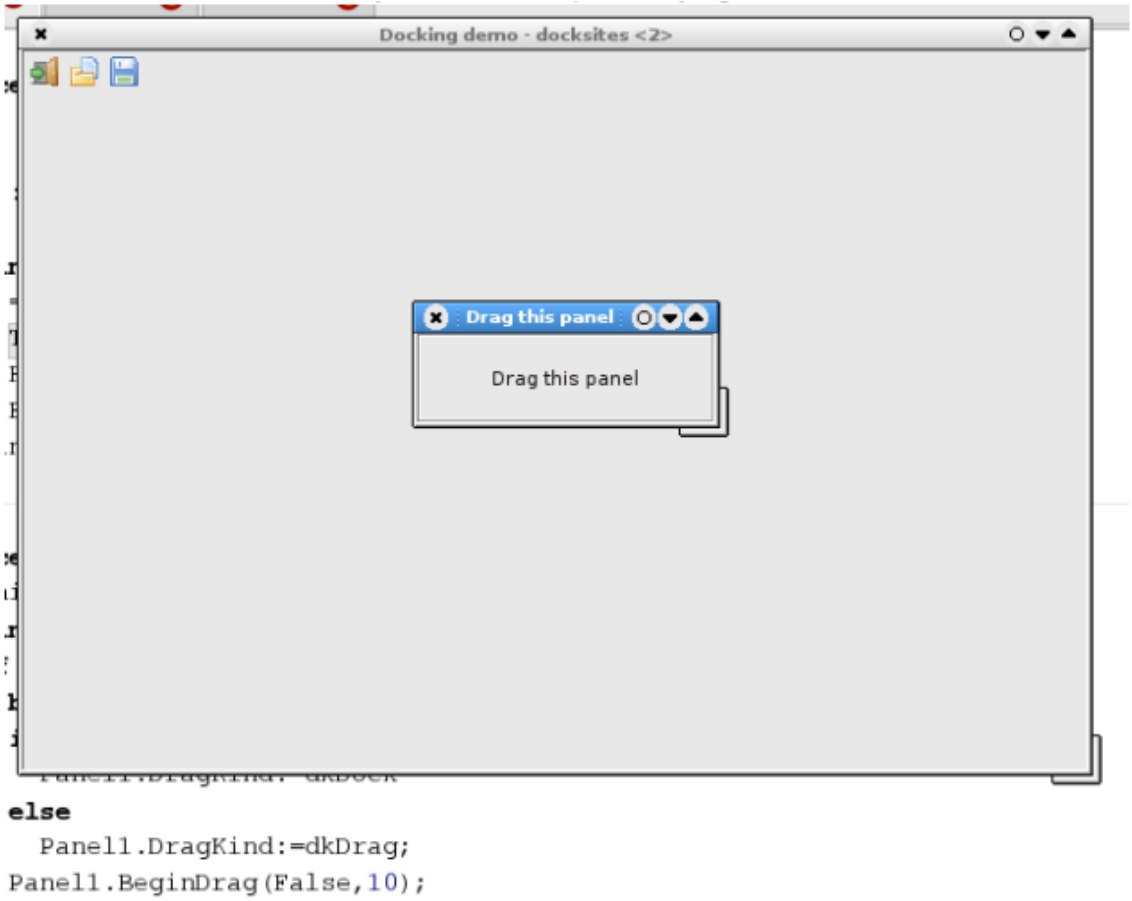
Параметр `TheScreenRect` - это ограничивающий прямоугольник (относительно экрана) для элемента управления, когда он перемещается. Параметр (необязательный) `KeepDockSiteSize` определяет, нужно ли изменять размер текущего стыковочного узла или нет (нет - значение по умолчанию).

Этот метод можно использовать, например, в событии double-click для панели:

```
procedure TForm1.Panel1DblClick(Sender: TObject);
var
  R: TRect;
begin
  R:=Panel1.BoundsRect;
  R.TopLeft:=ClientToScreen(R.TopLeft);
  R.Right:=R.Left+Panel1.Width;
  R.Bottom:=R.Top+Panel1.Height;
  Panel1.ManualFloat(R);
end;
```

Большая часть этого кода служит для вычисления размера и положения ограничивающего прямоугольника панели относительно экрана. Результат этого кода показан на рисунке 4:

Рисунок 4: Плавающая панель вручную



Наконец, также можно вручную пристыковать элемент управления к стыковочному узлу. Это можно сделать с помощью метода `ManualDock` элемента управления:

```
function ManualDock(NewDockSite: TWinControl; DropControl: TControl = nil; ControlSide:
TAlign = alNone; KeepDockSiteSize: Boolean = true): Boolean;
```

Параметр `NewDockSite` сообщает LCL, к какому элементе управления стыковочного узла должен быть пристыкован элемент управления, и является единственным обязательным параметром этого вызова. Параметры `DropControl` и `ControlSide` являются необязательными: если на стыковочном узлу уже есть некоторые элементы управления, пристыкованные к нему, то эти два параметра могут использоваться для указания относительного положения: `DropControl` - это элемент управления, относительно которого будет размещен новый элемент управления, а `ControlSide` определяет, где именно будет пристыкован элемент управления. Эти параметры соответствуют свойствам `DropOnControl` и `DropAlign` объекта `TDragDockObject`, описанного ранее.

Наконец, `KeepDockSiteSize` можно использовать для определения, будет ли механизм стыковки изменять размер стыковочного узла или нет. По умолчанию он сохраняет размер стыковочного узла.

Чтобы продемонстрировать использование этого, панель инструментов может быть вручную пристыкована на верхней панели примера программы: изначально панель инструментов вообще не пристыкована, а просто помещается над верхней панелью. Следующий код в вызове `FormCreate` пристыкует панель инструментов на верхней панели:

```
Toolbar1.ManualDock(PTop);
```

7. Некоторые события стыковки

Есть еще одна веская причина для ручной стыковки панели инструментов: каждый пристыковываемый элемент имеет событие `OnUndock`: это событие запускается при запуске операции перетаскивания и стыковки, оно запускается со стыковочного узла `TWinControl`, где элемент управления в настоящее время был прикреплен. Его можно использовать, например, для предотвращения операции перетаскивания и стыковки, как в следующем примере:

```
procedure TForm1.PTopUnDock(Sender: TObject; Client: TControl; NewTarget: TWinControl;
var Allow: Boolean);
begin
  Allow:=(NewTarget<>PBottom)
end;
```

Параметр `Allow` сообщает LCL, должна ли быть разрешена или запрещена операция отстыковки, а параметр `NewTarget` - это новый стыковочный узел, к которому пользователь хочет пристыовать элемент управления. Обратите внимание, что это событие запускается только тогда, когда пользователь фактически куда-то бросает элемент управления, то есть когда операция перетаскивания и стыковки завершена, и LCL пытается фактически пристыковать элемент управления.

Поскольку панель инструментов изначально не пристыкована, событие `OnUnDock` не будет инициировано, когда она будет пристыкована в первый раз. Пристыковка панели к верхней панели вручную гарантирует, что событие `OnUnDock` также будет запущено, когда панель инструментов будет повторно перепристыкована в первый раз.

Когда элемент управления где-то пристыкован, запускается событие `OnEndDock`. Например, это можно использовать для обратной связи с пользователем:

```
procedure TForm1Toolbar1EndDock(Sender, Target: TObject; X, Y: Integer);
begin
  Caption:=’Toolbar docked on ’+TComponent(Target).Name;
end;
```

Или можно сохранить новое местоположение стыковочного узла в файле .ini и восстановить его в новом сеансе программы с помощью `ManualDock`.

8. DockManager

Играя с программой-образцом, можно быстро заметить ограничение механизма drag & dock по умолчанию: на стыковочном узле виден только один элемент управления: LCL просто помещает один элемент управления поверх следующего на стыковочном узле, и поэтому последний брошенный на него элемент управления обычно будет единственным видимым.

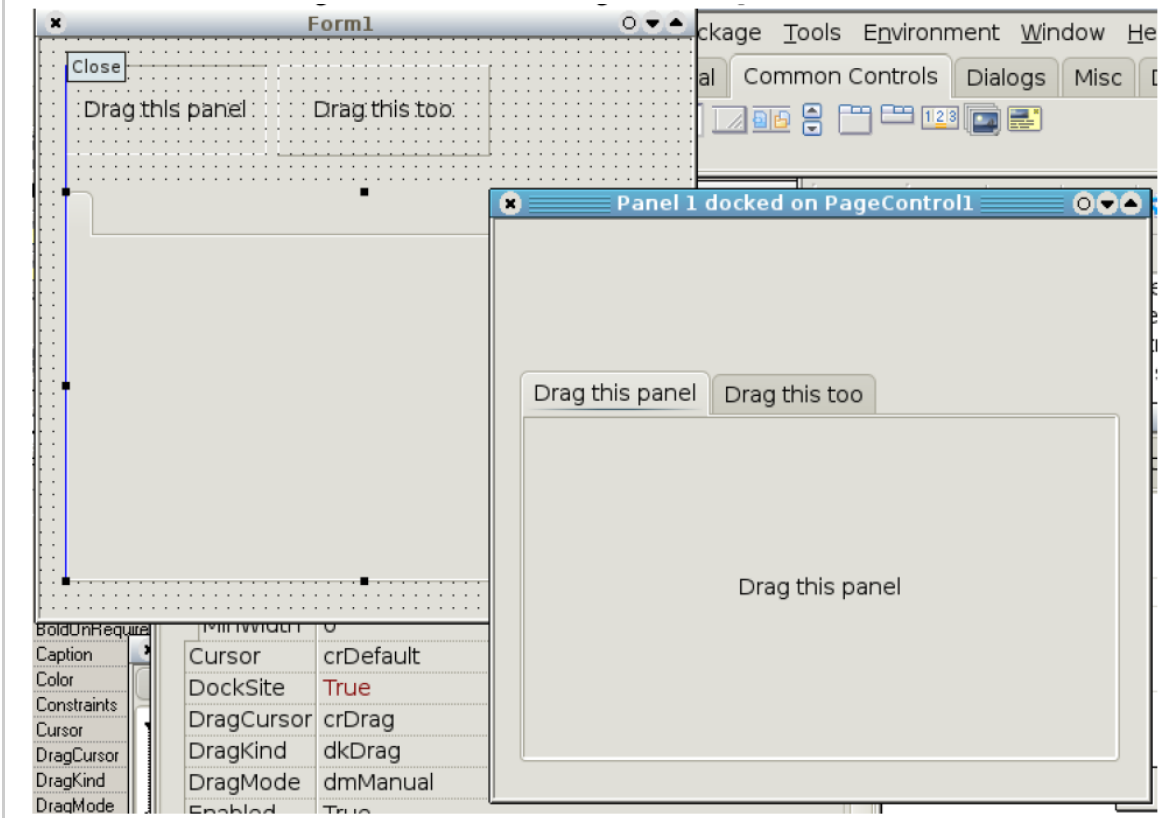
Когда `TWinControl` является стыковочным узлом и элемент управления пристыкован к нему, он использует менеджер пристыковки (экземпляр абстрактного класса `TDockManager` для управления позиционированием пристыкованных элементов управления. Это поведение можно отключить с помощью свойства `UseDockManager`: если оно установлено в значение `False`, диспетчер стыковки не используется, а пристыкованный элемент управления просто становится дочерним элементом стыковочного узла.

Если свойство `UseDockManager` имеет значение `True` (значение по умолчанию для всех элементов управления, кроме `TForm`), то экземпляр `dockmanager` создается сразу после пристыковки элемента управления (если его еще нет). По умолчанию фактический класс экземпляра `dockmanager` определяется глобальной переменной `DefaultDockManagerClass` в элементах управления модулем:

```
var
  DefaultDockManagerClass: TDockManagerClass;
```

Стандартная реализация `dockmanager` (в классе `TDockTree`) не содержит никакой логики для перемещения элементов управления на стыковочном узле. Некоторые потомки `TControl`, такие как `TPageControl`, реализуют собственный класс `dockmanager` для обеспечения настраиваемого поведения: в случае `TPageControl` диспетчер стыковочного узла стыкует каждый брошенный на него элемент управления на новой странице компонента `TPageControl`. Это можно легко продемонстрировать, создав форму с двумя пристыковываемыми панелями на ней и экземпляром `TPageControl`, который будет являться стыковочным узлом. После пристыковки обеих панелей на стыковочной форме будет создана ситуация, подобная рисунку 5:

Рисунок 5: Менеджер пристыковки элемента управления PageControl

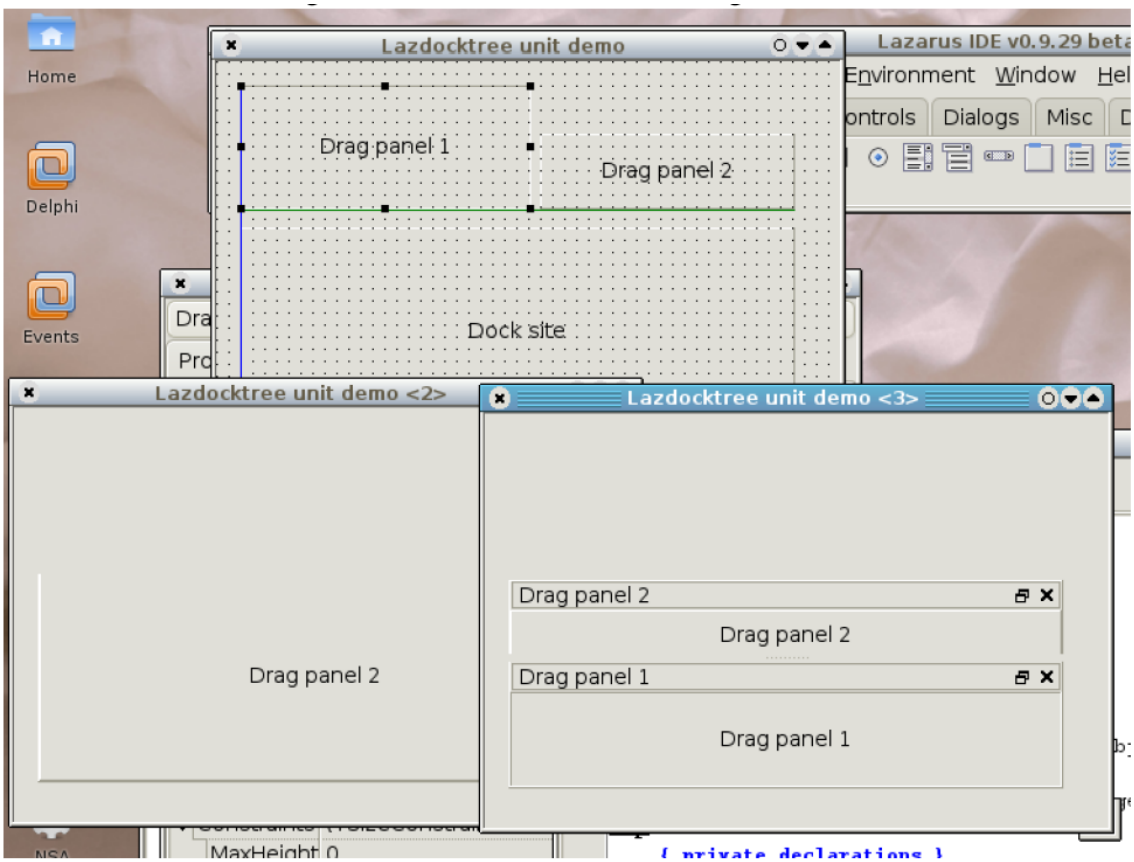


Оригинальная форма, как в конструкторе, также видна, чтобы показать разницу.

Стандартный класс стыковки не делает ничего особенного с элементами управления, состыкованными на стыковочном узле. Однако модуль `ldocktree` реализует класс `dockmanager` (и устанавливает его как `DefaultDockManagerClass`), который делает гораздо больше, чем просто родительские элементы пристыкованных элементов управления на стыковочном узле.

Чтобы продемонстрировать это, создадим небольшой проект с двумя перетаскиваемыми панелями и третьей панелью, которая является зоной стыковки. Панели разные по размеру и имеют разные названия. Программа компилируется и запускается (запускается вне IDE). Затем исходник модифицируется: модуль `ldocktree` добавляется в предложение `uses`, и программа снова запускается. Результат виден на рисунке 6:

Рисунок 6: Эффект работы менеджера стыковки модуля ldocktree



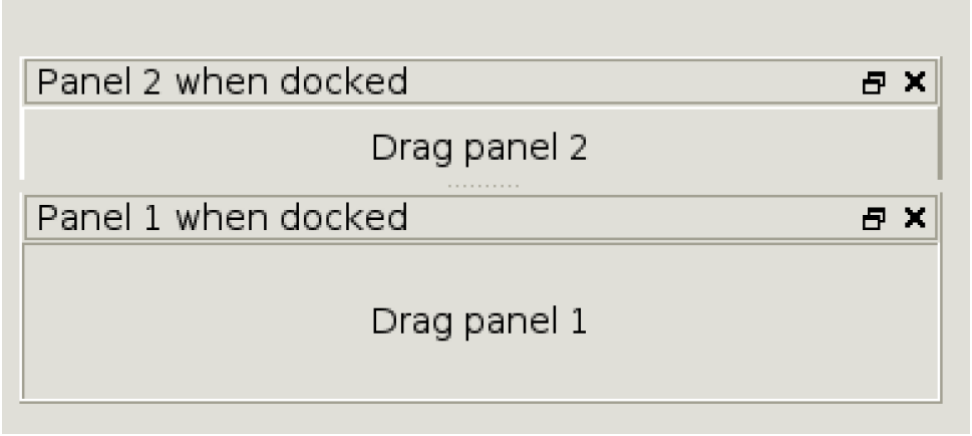
Он показывает 3 раза одну и ту же форму: *вверху* находится форма в режиме дизайна. *Левая нижняя форма* - это форма с двумя пристыкованными панелями, но без установленного менеджера стыковки: панель 2 была пристыкована последней и является единственной видимой. *Правая нижняя форма* относится к приложению, которое использует модуль `ldocktree`. Выглядит это заметно иначе: диспетчер стыковочного узла поместил панели в какое-то мини-окно и поместил их в стыковочный узел.

Заголовок, используемый при создании мини-окна стыковки, получается через свойство 'Text' пристыкованного элемента управления, но это можно настроить: событие `OnGetDockCaption` на стыковочном узле можно использовать для создания настраиваемых заголовков:

```
procedure TMainForm.CreateCaption(Sender: TObject; AControl: TControl; var ACaption:
String);
begin
  if AControl=DragPanel1 then
    ACaption:=’Panel 1 when docked’
  else if AControl=DragPanel2 then
         ACaption:=’Panel 2 when docked’;
end;
```

Эффект от этого можно увидеть на рисунке 7:

Рисунок 7: Пользовательские заголовки для диспетчера стыковки



Конечно, можно установить другие менеджеры стыковки: каталог *examples/dockmanager* в дереве исходных текстов Lazarus содержит реализацию менеджера стыковки, который реализует даже другие эффекты. Он поставляется с некоторыми демонстрационными программами, которые можно скомпилировать с разными директивами компилятора, чтобы показать различия. Он также содержит исследование по добавлению поддержки стыковки в саму Lazarus IDE. Заинтересованный читатель обязательно должен попробовать эти примеры, из них можно многое узнать о стыковках и диспетчерах стыковок.

9. Выводы

В этой статье сделана попытка показать, что стыковка в Lazarus легко реализуется: простая стыковка почти не требует кода, в то время как более сложные конфигурации возможны, но потребуют некоторого написания кода: особенно внимания требует размер стыкованных элементов управления. Восстановление макета стыковки еще не рассматривалось: это оставлено на будущее. Хотя Lazarus уже довольно давно поддерживает стыковку, механизмы определения размеров и стыковки подвергались серьезной реконструкции: часть приведенных здесь примеров была скомпилирована с использованием последних исходных кодов Lazarus, которые будут доступны на DVD, сопровождающем эту проблему.