TMS MultiTouch SDK
DEVELOPERS GUIDE

September 2011

Web: http://www.tmssoftware.com
Email: info@tmssoftware.com

Index

Introduction ... 4
Availability ... 4
Hardware ... 4
Online references ... 5
TMultiTouchRegion ... 6
TMultiTouchRegion description ... 6
TMultiTouchRegion features ... 7
TMultiTouchRegion architecture ... 8
TMultiTouchRegion use ... 9
TMultiTouchRegion properties ... 24
TMultiTouchRegion methods & functions ... 27
TMultiTouchRegion events ... 30
TMultiTouchRegion mouse support ... 33
TMultiTouchItem properties ... 34
TDefaultDisplayInfo properties ... 35
TMultiTouchTopLayerItem properties ... 36
TMultiTouchRegionItemAppearance properties ... 37
TMultiTouchRegion sample code ... 39
TMultiTouchRegionVisualizer ... 42
TMultiTouchRegionVisualizer description ... 42
TMultiTouchRegionVisualizer features ... 42
TMultiTouchRegionVisualizer use ... 42
TMultiTouchRegionVisualizer properties ... 42
TMultiTouchRegionVisualizer Events ... 43
TMultiTouchRegionPreviewVisualizer ... 44
TMultiTouchRegionPreviewVisualizer description ... 44
TMultiTouchRegionPreviewVisualizer features ... 44
TMultiTouchRegionPreviewVisualizer use ... 44
TMultiTouchRegionPreviewVisualizer properties ... 44
TMultiTouchRegionPreviewVisualizer events ... 45
TMultiTouchRegionPDFVisualizer ... 45

TMultiTouchRegionPDFVisualizer description 45
TMultiTouchRegionPDFVisualizer features 45
TMultiTouchRegionPDFVisualizer use 45
TMultiTouchRegionPDFVisualizer properties 46
TMultiTouchRegionPDFVisualizer events 46
Advanced techniques 47
TMultiTouch Samples 56
Electronic Board Demo description 56
Electronic Board Demo features 56
Electronic Board Demo screenshots 56
Music Browser Demo description 58
Music Browser Demo features 58
Music Browser Demo Screenshots 58
Photo Album Demo description 60
Photo Album Demo features 60
Photo Album Demo screenshots 60
QuickPDF Demo description 62
QuickPDF Demo features 62
QuickPDF Demo Screenshots 62
TMS Products Viewer Demo description 65
TMS Products Viewer Demo features 65
TMS Products Viewer Demo Screenshots 65

Introduction

The TMS MultiTouch SDK is designed to give your applications immersive graphical user interfaces including versatile multi-touch capabilities like item swipe, move, rotate and resize.

The core component is the TMultiTouchRegion component handling multitouch input and abstracting all difficulties of item manipulation with multiple fingers. In addition, the TMS MultiTouch SDK includes the entire range of TMS Smooth Controls with iPhone/iPad style thumbnail list, calendar, touch keyboard, popup, buttons, trackbars and much more.

The TMS MultiTouch SDK is designed for Windows 7 32bit or 64bit with single touch or multi-touch enabled screens. The number of touch points is limited to the number supported by the hardware you use. With the TMS SmoothTable products, offering a POS type wall-mount or table type multitouch screen, this ranges from 6 touch points for the smallest 32" version and up to 32 for the 60" version. Contact TMS software sales for more information about our multitouch hardware offerings.

In this document you will find an overview of the TMultiTouchRegion component and its features, code snippets to quickly start using the components and overviews of properties, methods and events. A separate developers guide is available for the TMS Smooth Controls.

Availability

The TMS Multitouch SDK consists of VCL components for Win32 or Win64 application development. The TMS MultiTouch SDK is available for Embarcadero™ 2010, XE, XE2 & C++Builder 2010, XE, XE2.

Hardware

The TMS Multitouch SDK will work with multi-touch enabled screens designed for Windows 7, like for example the Dell SX2210T. tmssoftware.com offers in addition to the TMS MultiTouch SDK a standard 32" 1920x1080 multitouch screen in black or white aluminium cases & glass cover for POS applications and can contract designs for multitouch screens up to 60". Standard solutions start from 3750 EUR. Contact TMS software sales for more details on our multitouch hardware offerings.

List of included components

- TMultiTouchRegion: core multitouch handling component
- TMultiTouchRegionVisualizer: base class for item visualizers
- TMultiTouchRegionPreviewVisualizer: Windows shell preview based item visualizer
- TMultiTouchRegionPDFVisualizer: QuickPDF based item visualizer for pages from a PDF file

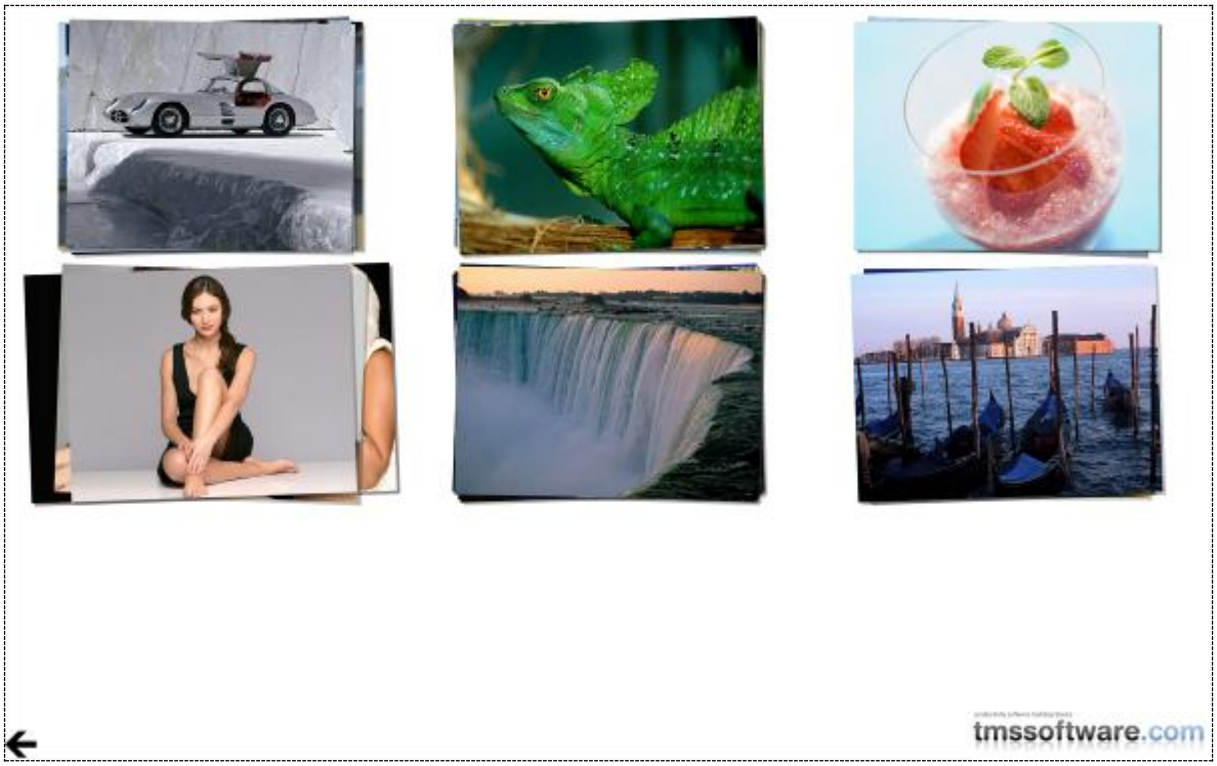
Online references

TMS software website:
http://www.tmssoftware.com

TMS MultiTouch SDK page:
http://www.tmssoftware.com/site/multitouchsdk.asp

TMS Smooth Controls Pack page:
http://www.tmssoftware.com/site/advsmoothcontrols.asp

TMultiTouchRegion



TMultiTouchRegion description

TMultiTouchRegion is the core component of the TMS MultiTouch SDK. The component is designed to visually display, manipulate, and interact with items in various ways: item linking, rotation, scale, zoom, selection, multiselect, flip and much more.
The component offers different display modes such as Cascade, Grid, Position, Random, Stacked and Transform. You can consider TMultiTouchRegion as a component that has 4 layers:

1) Background: can be a texture, a gradient, an image and more. When enabled, the background itself can be manipulated: scale, rotate, move

2) Static background items: items that can be positioned anywhere on the background and that cannot be manipulated but that are sensitive to a touch click.

3) Normal items: these items can be manipulated with touch and within this items layer, the item last clicked has the highest z-index. Normal items have content and optionally a detail. Controls can be associated with items to execute specific actions on item level.

4) Static top layer items: these items are identical to items in the background layer but remain always on top of normal items. The items cannot be manipulated but are touch click sensitive. These could be used to offer an exit button, open a menu etc… in the application.

TMultiTouchRegion features

- Multitouch manipulations of items: rotate, scale, pan, zoom, move…
- Display modes for items: Cascade, Grid, Position, Random, Stacked and Transform
- Items can have an image, text or combined
- Items can have associated controls
- Items have default content & detail content with flip animation between default & detail
- Visualizer concept to allow the display any kind of file type (includes visualizer for Windows shell preview & PDF pages (via QuickPDF lib))
- Custom drawing on items
- Manipulation of region background (pan, zoom, scale)
- Static control items on region
- Built-in methods to perform screen rotation (0°,90°,180°,270°) to automatically change the orientation of the screen towards the position of the user.
- Different states for each item : normal, selected
- Animated transitions
- Automatic paging in grid mode
- Automatic stack to other display mode transitions with animation
- Configurable number of rows / columns in grid mode
- Complex fills (texture, gradients, image) on items, control items

TMultiTouchRegion architecture

- Background / region: surface on which both static items & items that can be manipulated in the UI can be presented. Optionally, the background region can be manipulated, causing all items in the region to move/scale/rotate along with the background. The background is drawn with a fill as defined via TMultiTouchRegion.Fill.
- Items: is a collection of TMultiTouchItem instances. These are items that can be manipulated (move, scale, resize, rotated) in the UI.
- TMultiTouchControlItems: is a collection of TMultiTouchTopLayerItem instances. These are items that cannot be manipulated, only a click event is triggered for these items. These items can be below normal items or on top of normal items.
- Scroll indicators: as the background region can be larger than the control size, scroll indicators can show both vertically & horizontally the position of the region in view within the total region size.
- Visualizer: a visualizer is a component that can be linked to the TMultiTouchRegion, and is responsible for the drawing of the item. The concept of a separate visualizer class allows extending the component to draw different kinds of resources in items, for example a page of a PDF document, a shell preview image of a file, etc…
- TDefaultDisplayItem: class that holds default settings that will be applied when new items are created
- TMultiTouchRegionItemAppearance: class that holds various color settings of how an item looks in normal & selected state

TMultiTouchRegion use

First use

Dropping a TMultiTouchRegion on the form displays a container control in a default style.

1) Adding or removing Items can be done programmatically or by using the TMultiTouchRegion.Items collection property of the control at design time.

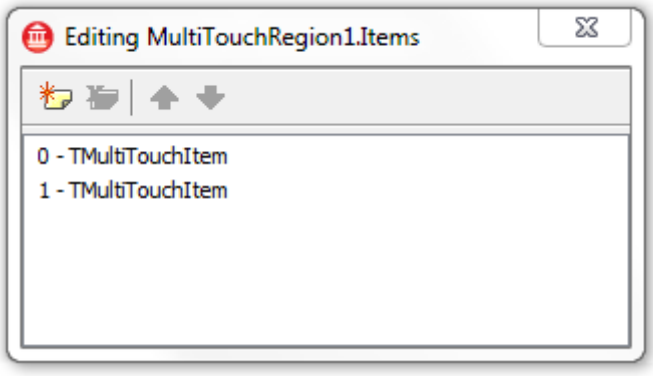
2) The class property TMultiTouchRegion.DefaultItem holds the settings how an item added to the collection looks and behaves by default. With this class property it can be avoided to write a lot of code to initialize new created items to a specific setting.

3) The appearance of the items is set via TMultiTouchRegion.ItemAppearance. This defines the settings for the appearance of an item in normal state and selected state.

4) Static items that are positioned on the region either below or on top of normal items and that are just touch click sensitive are added via the TMultiTouchRegion.ControlItems collection.

Adding /Removing Items

Adding or removing items can be done either programmatically or at design-time. Double-click the Items property and the default collection editor will popup, allowing you to add new or remove existing Items.



To add a new or remove an existing item programmatically use the code below.

Add an item:

```
MultiTouchRegion.BeginUpdate;
with MultiTouchRegion.Items.Add do
begin
  MainItem.FileName := 'sample.png';
  MainItem.Caption := 'sample';
end;
MultiTouchRegion.EndUpdate;
```

Remove an item:

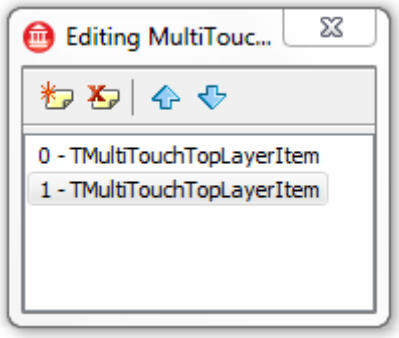
```
MultiTouchRegion.BeginUpdate;
MultiTouchRegion.Items.Delete(ItemIndex);
MultiTouchRegion.EndUpdate;
```

The behavior of newly created items can be defined via its properties. The following sample enables horizontal movement and resizing of the items, and disables the possibility to rotate and move vertically:

```
for idx := 0 to 2 do
begin
  MultiTouchRegion1.Items[idx].CanRotate := False;
  MultiTouchRegion1.Items[idx].CanMoveHorizontal := True;
  MultiTouchRegion1.Items[idx].CanMoveVertical := False;
  MultiTouchRegion1.Items[idx].Resizable := False;
end;
```

Adding /Removing ControlItems

Adding or removing controlitems is done in a similar way as items, either programmatically or at design-time. Double-click the TMultiTouchControlItems property and the default collection editor will popup, allowing you to add new or remove existing controlItems.



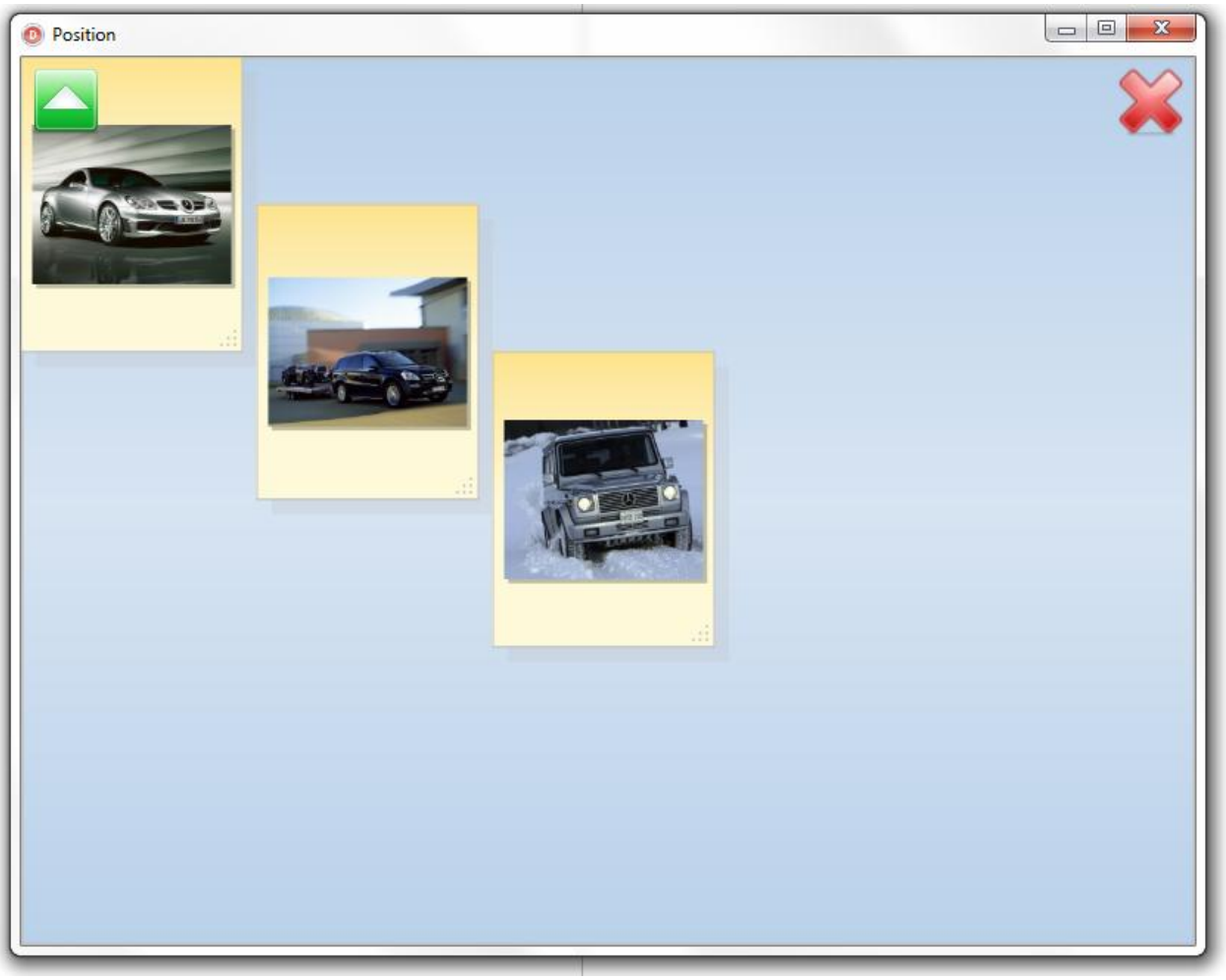
To add a new or remove an existing controlitem programmatically use the code below.

Add a controlitem: this code generates the left top button in Sample2.

```
MultiTouchRegion.BeginUpdate;
with MultiTouchRegion.ControlItems.Add do
begin
  Top:=6;
  Left:=6;
  Height := 48;
  Width := 48;
  Fldr := ICONSFOLDER;
  Fill.Picture.LoadFromFile(Fldr + 'up_128.png');
  Location := ilTopLeft;
  Fill.PictureSize := psCustom;
  Fill.PictureHeight := MultiTouchRegion1.ControlItems.Items[0].Height;
  Fill.PictureWidth := MultiTouchRegion1.ControlItems.Items[0].Width;
  Fill.PictureAspectMode := pmStretch;
  Fill.Color := clNone;
  FillDown.Color := clNone;
end;
MultiTouchRegion.EndUpdate;
```

Remove a controlitem:

```
MultiTouchRegion.BeginUpdate;
MultiTouchRegion.ControlItems.Delete(ItemIndex);
MultiTouchRegion.EndUpdate;
```



The code snippet below illustrates how the click on a controlitem can be used to change the display mode of normal items in the TMultiTouchRegion and how another one is used to exit.

The left button changes the default DisplayMode to GridMode and back. The right button closes the form.

```
procedure TForm1.MultiTouchRegion1ControlItemClick(Sender: TObject;
  ControlItem: TMultiTouchCustomItem);
begin
  case ControlItem.Index of
  0:
  begin
    if MultiTouchRegion1.DisplayMode = dmPosition then
      MultiTouchRegion1.DisplayMode := dmGrid
    Else
      MultiTouchRegion1.DisplayMode := dmPosition;

    MultiTouchRegion1.BeginUpdate;
    MultiTouchRegion1.ResetItems;
```

```
      MultiTouchRegion1.EndUpdate;
    end;
    1:  Form1.Close;
    end;
  end;
```

Display Modes

The DisplayMode determines how the items are by default displayed in the TMultiTouchRegion control. The DisplayMode property is set by default to Random. This means the items will be added at a random position and with a random rotation on the region. TMultiTouchRegion.RandomAngle sets the range of angles (in degrees from - TMultiTouchRegion.RandomAngle degrees to +TMultiTouchRegion.RandomAngle) that are randomly applied when items are positioned on the region.

The TMultiTouchRegion.DefaultItem class property provides options like CanMoveHorizontal, CanMoveVertical, CanRotate, CanScale, Resizable and EnabeleFlip. These default settings will be automatically applied when an item is created.

Appearance of the item in normal state and selected state can be set via the TMultiTouchRegion.ItemAppearance.Fill and TMultiTouchRegion.ItemAppearance.FillSelected properties. This is a common property for all items, meaning that all items will have the same appearance for normal state and selected state. If there is a need to have a specific appearance for a specific item, this can always be customized via the event OnItemFill.

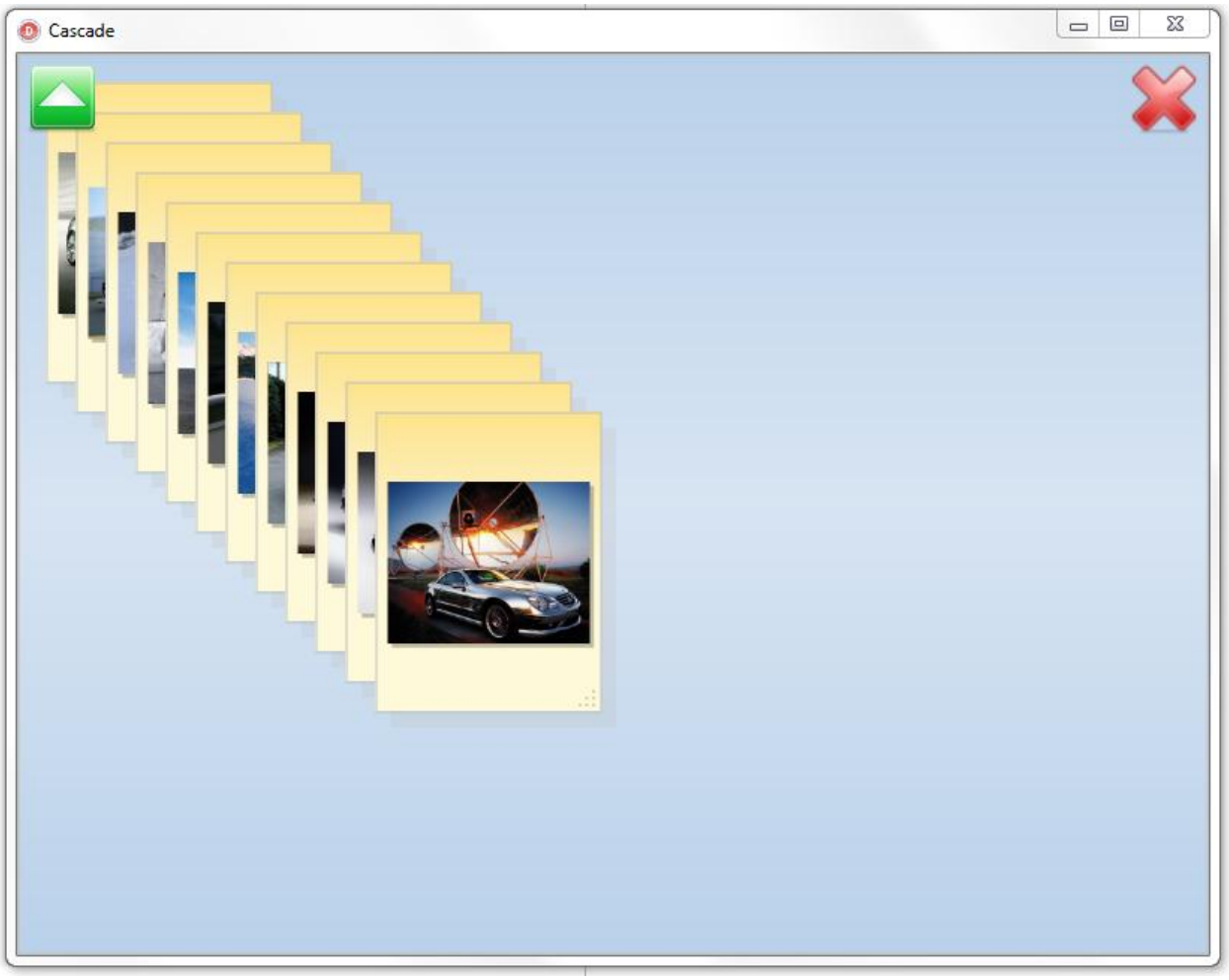
Sample:

```
procedure TForm1.MultiTouchRegion1ItemFill(Sender: TObject;
  Item: TMultiTouchItem; AFill: TmtouchFill);
begin
  // apply a specific color only for items where Tag = 1
  if Item.Tag = 1 then
  begin
    // in selected state, set item's color to red
    if Item.Selected then
      AFill.Color := clRed
    else
    // otherwise, set it to yellow
      AFill.Color := clYellow;
  end;
end;
```

The different display modes are:

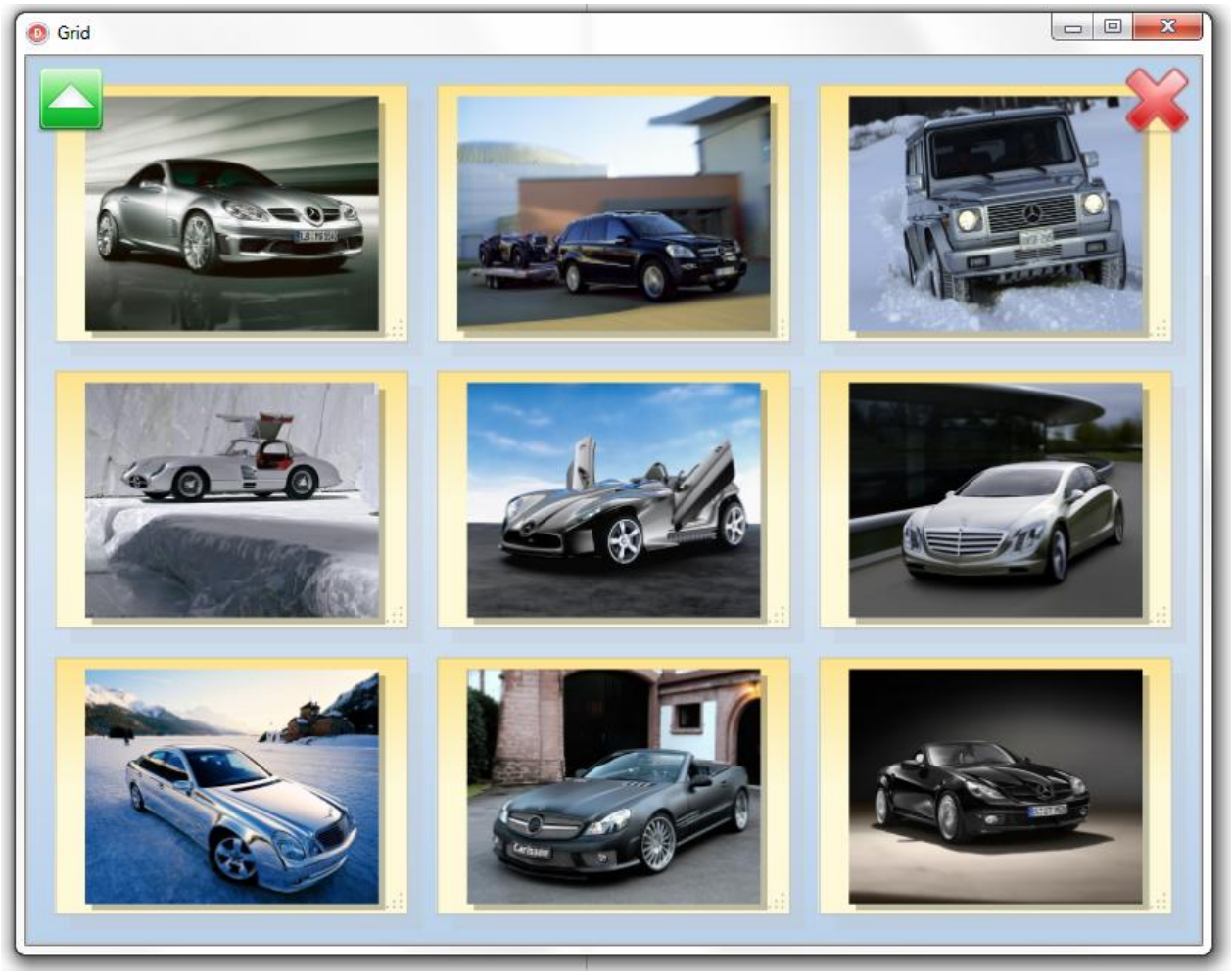
- **dmCascade**: Items are displayed from left/top to right/bottom. As each item is added, it is shown in the region with a small horizontal & vertical offset from the previously added item. The horizontal and vertical offset that is applied in cascade mode can be set with TMultiTouchRegion.HorizontalSpacing, TMultiTouchRegion.VerticalSpacing properties.

```
TMultiTouchRegion.DisplayMode := dmCascade;
```



- **dmGrid**: as items are added, these are automatically positioned in a grid layout. The properties TMultiTouchRegion.Columns & TMultiTouchRegion.Rows determine how many columns and rows are used in this grid layout. The distance between rows and columns is set with TMultiTouchRegion.HorizontalSpacing and TMultiTouchRegion.VerticalSpacing. When there are more items than can fit in a grid layout of Columns x Rows items, a new page is added and paging between series of items is automatically performed when the user swipes horizontally over the screen.

```
TMultiTouchRegion.DisplayMode := dmgrid;
```



- **dmPosition**: In this mode, the position of the item is controlled by the properties with X,Y position of the item and the preset item's width and height:

```
TMultiTouchItem.ItemX: integer;
TMultiTouchItem.ItemY: integer;
```

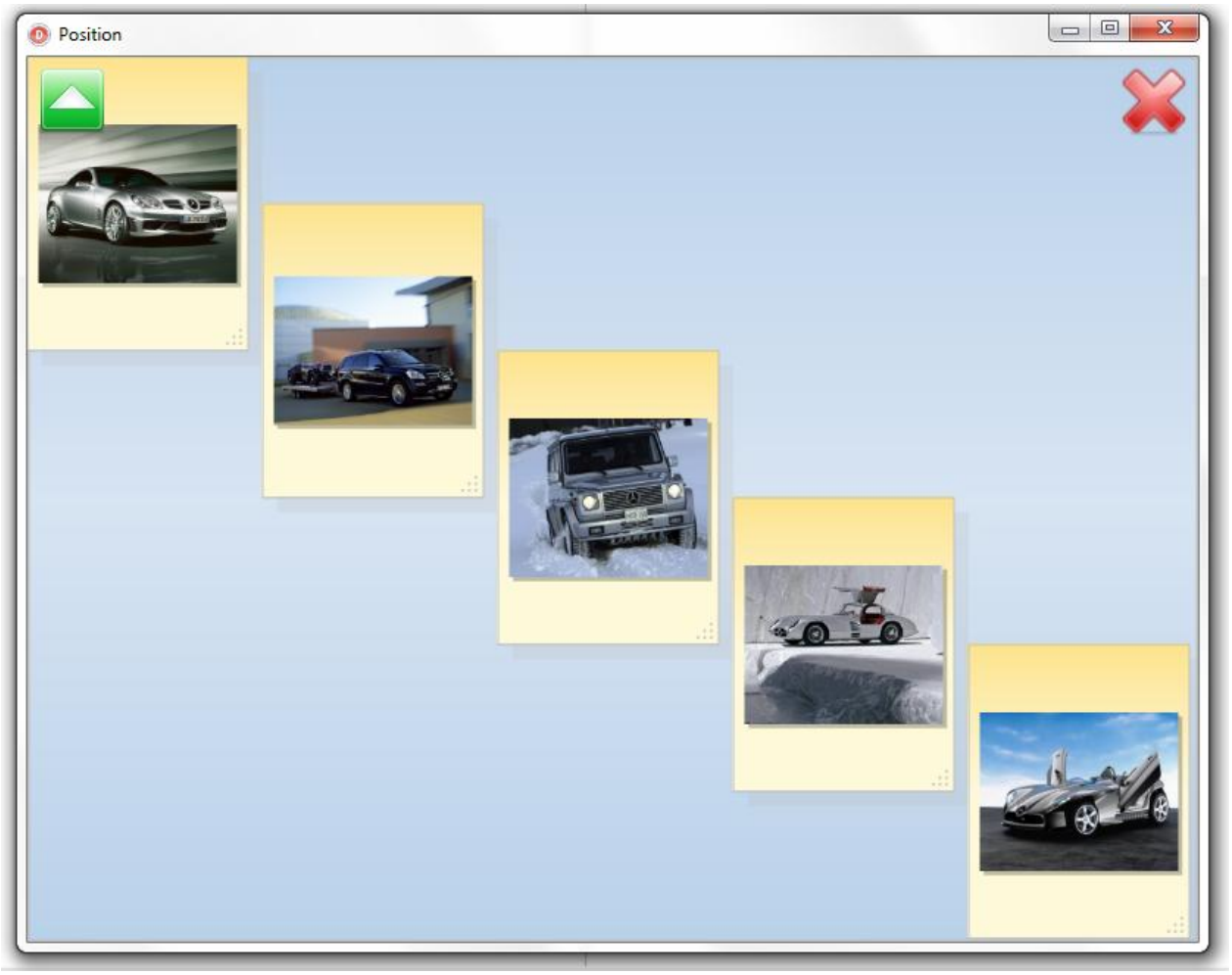
The size of the item is set with:

```
TMultiTouchItem.ItemWidth: integer;
TMultiTouchItem.ItemHeight: integer;
```

Example:

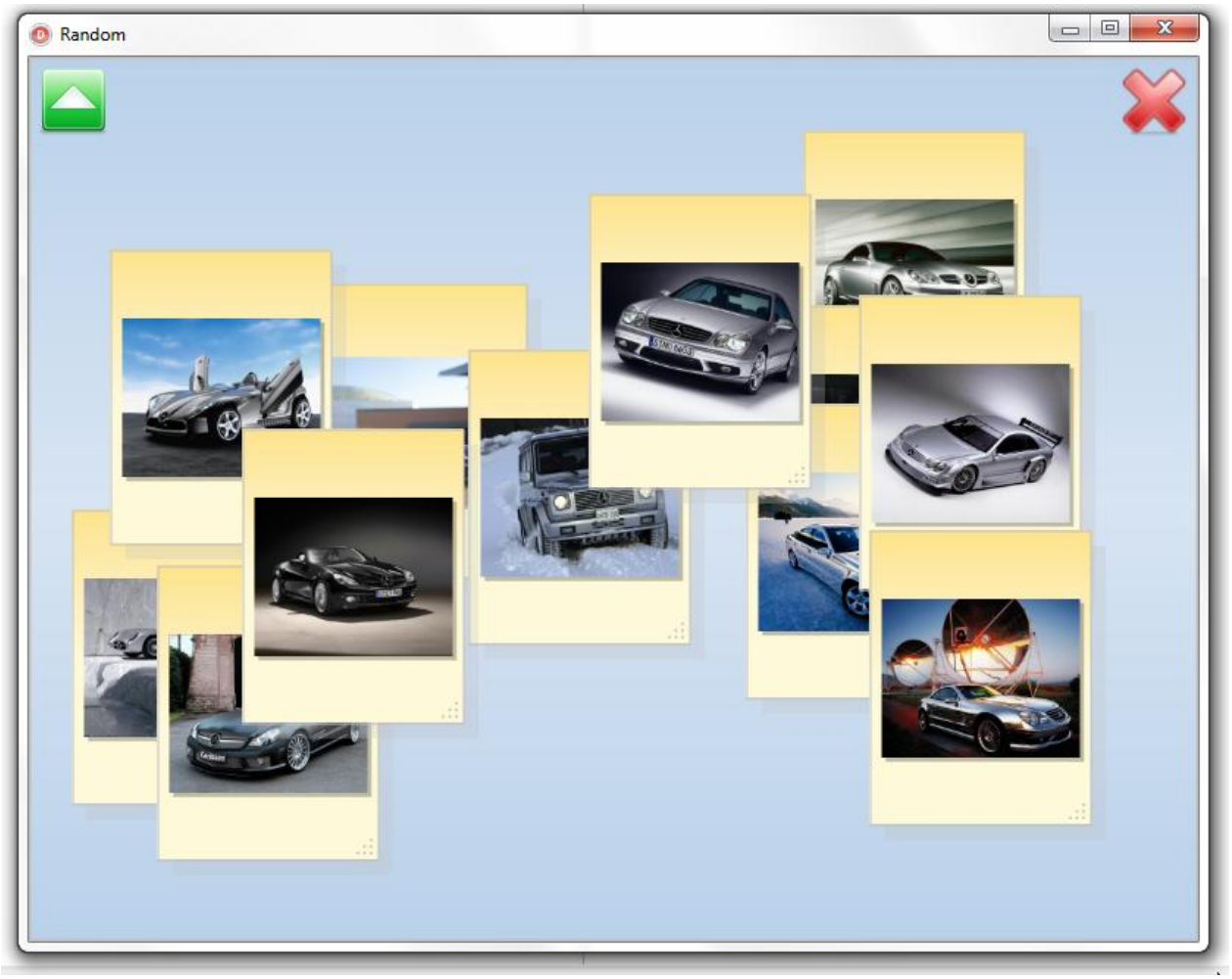
```
MultiTouchRegion1.DisplayMode := dmPosition;
var
  mti: TMultiTouchItem;
for i := 0 to 2 do
```

```
begin
    mti := MultTouchRegion1.Items.Add;
    mti.ItemX := idx * 200;
    mti.ItemY := idx * 100;
end;
```



- **dmRandom**: The position of the items is randomly chosen and a random angle can be applied. The range of the random angle that is applied is set via TMultiTouchRegion.RandomAngle: integer. When this is a set to X, the angle can vary between –X and +X degrees. Note that in random mode, the size of items is defined by MultiTouchItem.ItemWidth / MultiTouchItem.ItemHeight.

```
MultiTouchRegion1.DisplayMode := dmRandom;
```



- **dmStacked**: In this mode, items that have the same StackIndex property are displayed as a stack. This means that items are initialized with a position on top of each other with a small random angle applied to make it look like a stack.

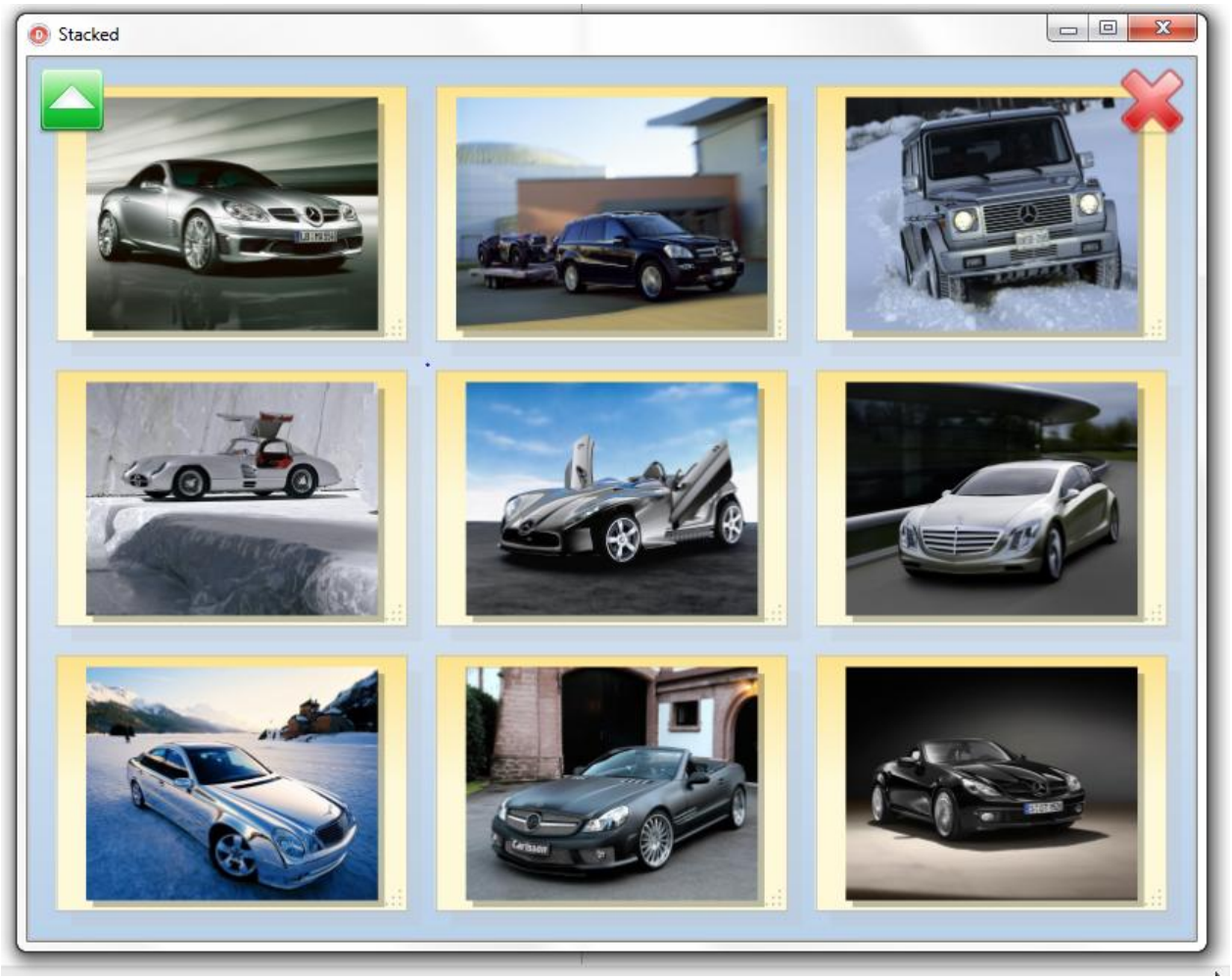
  The code below creates a stack of images by loading the images from a folder and setting the StackIndex of all these items to 1. A second stack could be added by calling the same code for a different folder and with a different stack index. Note that stacks are always displayed in a grid layout.

```
var
 AFolder: string;

begin
  MultiTouchRegion1.DisplayMode := dmStacked;
  AFolder := DEMOFILESFOLDER + '';
  // 2nd parameter sets the stack index to 1
  MultiTouchRegion1.AddFileLocationsFromFolder(AFolder +'.jpg', 1,
      False);
```

```
end;
```

After a touch on the stack, it expands to a new view containing all items in this stack. The view mode is set with TMultiTouchRegion.OpenStackMode. If this would be set to dmRandom for example, this means that after touching the stack, it will be expanded with an animation effect to a random positioning of all items in this stack. Note that the speed of this animation is controlled by TMultiTouchRegion.AnimationFactor. The lower the animation factor is set, the higher the speed of this animation.



- **dmTransform**: In the dmTransform mode, the initialization of the items on the region is controlled by its transforms. The transform is a matrix that controls position, scale and rotation of an item. This matrix holds the transform needed to position an item with ItemWidth/ItemHeight at coordinates 0,0 and with 0,0 degrees rotation at its final desired position/scale/rotation. The item consists of a background and content (content means an image or text positioned on this background). As such, there are two transforms, one for the background and one for the content. As an item can have a main side and detail side, each side has these 2 transforms. The transforms can be accessed by public properties.

```
item.MainItem.ContentTransform: TD2DMatrix3x2F;
item.MainItem.BackGroundTransform: TD2DMatrix3x2F;
```

Either this matrix can be set directly or record helper functions for this matrix can be called:

```
uses
  D2D1;

// apply a rotation of deg degrees around point X,Y:
```

```
Item.MainItem.ContentTransform := Item.MainItem.ContentTransform *
    TD2DMatrix3x2F.Rotation(deg, x, y);
```

Following snippet results in the transform example below:

```
Item.MainItem.ContentTransform := Item.MainItem.ContentTransform *
TD2DMatrix3x2F.Rotation(Item.Index * 120, 200, 200);

Item.MainItem.BackGroundTransform :=
Item.MainItem.BackGroundTransform *
TD2DMatrix3x2F.Rotation(Item.Index * 120, 200, 200);
```

dmTransform allows extensive fine-tuning, which is demonstrated by the following sample

```
MultiTouchRegion1.Items[I].MainItem.BackGroundTransform :=
TD2DMatrix3x2F.Translation(100, 100);
```

```
        MultiTouchRegion1.Items[I].ItemWidth := 500;

        MultiTouchRegion1.Items[I].ItemHeight := 400;

        MultiTouchRegion1.Items[I].MainItem.ContentTransform :=
TD2DMatrix3x2F.Translation(100, 100);

        MultiTouchRegion1.Items[I].MainItem.ContentWidth := 400;

        MultiTouchRegion1.Items[I].MainItem.ContentHeight := 300;

        MultiTouchRegion1.Items[I].MainItem.ContentTransform :=
TD2DMatrix3x2F.Translation(150, 150);

        MultiTouchRegion1.Items[I].MainItem.BackGroundTransform :=
MultiTouchRegion1.Items[I].MainItem.BackGroundTransform *
TD2DMatrix3x2F.Rotation(45, D2D1PointF(100 + 250, 100 + 200));

        MultiTouchRegion1.Items[I].MainItem.ContentTransform :=
MultiTouchRegion1.Items[I].MainItem.ContentTransform *
TD2DMatrix3x2F.Rotation(45, D2D1PointF(150 + 200, 150 + 150));
```

Paging

When using the TMultiTouchRegion in dmGrid and dmStacked mode, the existing items are automatically positioned in a grid layout, depending on the number of columns and rows set.

When there are more items than the page can handle, a new page is automatically created which holds the remaining items.

The PageIndex property can be used to switch programmatically between different pages or read the currently selected page. The PageCount property returns the total number of available pages in dmGrid mode.

TMultiTouchRegion properties

- **AnimationFactor**: The factor used control the speed of built-in animations. The lower the factor is set, the faster the animation.
- **Columns**: The number of columns per page for display mode dmGrid and dmStacked.
- **ControlAppearance:** This holds settings that will be applied when control items are created
  - DefaultDownFill: default fill used for control items in the down state
  - DefaultFill: default fill used for control items in normal state
  - Font: sets the default font that will be used for region control items
  - ItemFont: sets the default font that will be used for item controls items
- **ControlItems:** A collection of items that are statically positioned on the TMultiTouchRegion. TMultiTouchControlItems cannot be moved, sized, scaled, rotated and are only sensitive to a single touch. A control item can display a picture and/or text. Control items can be used to add fixed controls in the region like an exit button for example. Control items are either positioned below regular items or above regular items. This is controlled by the property TMultiTouchTopLayerItem.BackgroundLayer. Set this to false if the control item is at all times above regular items. The full list of properties for the control item class can be found in the paragraph about TMultiTouchTopLayerItem.
- **DefaultItem:** A set of properties that can be used to define the default appearance and behavior for all created items. The full list of properties for the control item class can be found in the paragraph about TMultiTouchItem.
- **DisplayMode:** This display mode can be set to the values: dmCascade, dmGrid , dmPosition, dmRandom, dmStacked, dmTransform.
- **EnableInertia:** When true, inertia is applied to the physical movements of items. The parameters that control the inertia effect can be dynamically set via the event: OnSetInertiaParameters. Detailed information about the parameters for the inertia processing can be found at: http://msdn.microsoft.com/en-us/library/dd562169(v=vs.85).aspx
- **EnableInteraction:** When true, enables interaction with the items in the control.
- **EnableManipulation:** When EnableInteraction is true and EnableManipulation is true, manipulation of items in the control is possible. When not enabled, items cannot be moved, sized, scaled or rotated. Touch events in this case will manipulate the background region (when enabled).
- **Fill:** A set of properties to define the appearance of the TMultiTouchRegion. The full list of properties for the control item class can be found in the paragraph about TMultiTouchRegionItemAppearance.
- **HorizontalSpacing:** Defines the horizontal spacing between the onscreen items when display mode is dmGrid or dmStacked or the horizontal offset in dmCascade mode.
- **ItemAppearance**:
  - **Fill:** A set of properties to specifiy the appearance of items. The full list of properties of this class can be found in the paragraph about MultiTouchRegionItemAppearance.

  - **FillSelected:** This is the same as the Fill property but specifies the appearance of items in selected state.
  - **Font:** Sets the font for text that can be associated with an item.
- **Items:** A collection of items that are dynamically positioned on the TMultiTouchRegion. Items can be moved, sized, scaled, rotated and are sensitive to different types of touch. An item can by default display a picture and/or text. The full list of properties for the item class can be found in the paragraph about TMultiTouchItems.
- **MultiSelect:** When set to true, multiple items can be selected simultaneously. Note that when multiselection is enabled, all selected items can be manipulated at once as a group of items.
- **OpenStackedMode:** Sets the display mode that will be used when the user clicks on a stack to open it. This OpenStackedMode can be any of the values: dmCascade, dmGrid , dmPosition, dmRandom, dmStacked, dmTransform.
- **RandomAngle:** Sets the range of the randomly applied angle to items in dmRandom mode. The randomly applied angle is between –RandomAngle degrees and +RandomAngle degrees.
- **RegionCanMoveHorizontal:** When set to true, allows the region to move horizontally. This is useful for using a region that is larger than the control size. Scroll indicators can show the horizontal position of the actual region in view, within the total region size.
- **RegionCanMoveVertical:** When set to true, allows the region to move vertically. This is useful when using a region that is larger than the control size. Scroll indicators can show the vertical position of the actual region in view, within the total region size.
- **RegionCanRotate:** When set to true, allows the region to rotate.
- **RegionCanScale:** When set to true, allows the possibility to scale
- **RegionZoomMaximum:** Defines the maximum zoom-in ratio of the background region.
- **RegionZoomMinimum:** Defines the minimum zoom-out ratio of the background region.
- **ResizeControls:** When set to true, scales the TMultiTouchControlItems controls associated with the item while scaling the item itself.
- **Rows:** The number of rows per page in display modes dmGrid and dmStacked.
- **StackedMaximumVisible:** Defines the maximum number of items that will be effectively painted in dmStacked display mode. Limiting this value improves performance.
- **TouchEvents:** Different reactions can be defined for the different touch events.
  - **DblTouch:** action to perform on a double-touch:
    - **dmZoomInOut:** Perform zoom-in, zoom-out of the touched item.
    - **dmDetail:** Shows detail information (backside of the item).
    - **dmNone:** No action is taken.
  - **LeftTouch:** action to perform on a single touch:
    - **mDrag:** Starts dragging an item.
    - **mDetail:** Shows detail information (backside of the item).
    - **mNone:** No action is taken.
    - **mTouch:** Selects the item.
    - **mZoomInOut:** Perform zoom-in, zoom-out of the touched item.
  - **RightTouch:** action to perform on a mouse right-click or long touch:

  - Settings are identical to LeftTouch.

- **Visualizer:** Sets the visualizer class instance via which the content of the items is rendered. Standard following visualizer classes are provided: TMultiTouchRegionVisualizer, TMultiTouchRegionPreviewVisualizer, TMultiTouchRegionPDFVisualizer.

TMultiTouchRegion methods & functions

- **BeginUpdate and EndUpdate**

  To optimize performance, redrawing of items can be postponed till multiple items or added or till multiple item properties are changed. This is achieved via surrounding the code that performs multiple settings between BeginUpdate/EndUpdate calls.

```
MultiTouchRegion1.BeginUpdate;
   // add multiple items here or change several properties
MultiTouchRegion1.EndUpdate;
```

- **StopThread and StartThread**

  When using methods of TMultiTouchRegion that internally use threading to load the images (like AddFileLocationsFromFolder) you can interrupt the thread by calling StopThread. To start or resume loading the images again, call StartThread.

- **AddFileLocationsFromFolder(AFolder: String; AStackIndex: integer = - 1; UseFileNameCaption: Boolean = True)**

  Adds files matching a specific filter from a folder as items. The 2nd parameter is the StackIndex and this is a default parameter with default value -1. As such, when not used, items are not added in a stack. When the 2nd parameter is >= 0, it is added to a stack with this index. Stacks are displayed on the region starting from stack 0 in the upper left corner, stack 1 on the right and so on. When the last parameter is set to true, the filename of the file retrieved is automatically set as caption for the item.

  Sample:

```
MultiTouchRegion1.AddFileLocationsFromFolder(AFolder +'\*.jpg',
AStackIndex, False);
```

- **SaveToFile(AFileName: string) and LoadFromFile(AFileName: string)**

  SaveToFile saves the full state of items to a file. This means its properties such as CanMoveVertically, CanMoveHorizontally, CanRotate, CanScale but also its transforms. This full information enables to persist the state of the items and restore it later. The LoadFromFile method loads previously saved items state from a file.

- **ChangeDisplayOrientation(DisplayOrientation: TDisplayOrientation) and GetDisplayOrientation: TDisplayOrientation**

ChangeDisplayOrientation can be used to change the orientation of the screen. The rotation can be: doDefault, do90, do180, do270. This provides the capability to orient the screen in the direction at which the user looks at the screen.

To flip the screen for example, use:

```
MultiTouchRegion1.ChangeDisplayOrientation(do180)
```

The GetDisplayOrientation function retrieves the actual display orientation.

- **SaveRegionToFile(AFileName: string) and LoadRegionFromFile(AFileName: string)**

  SaveRegionToFile saves the state of the background region (its transforms) to a file.

  LoadFromFile loads previously saved transforms of the background from a file.

- **ZoomInOut(AItemIndex: Integer)**

  ZoomInOut toggles the zoom state of the item with index AItemIndex in the Items collection.

- **ZoomIn(AItemIndex: Integer)**

  ZoomIn puts the item with index AItemIndex in zoomed state

- **ZoomOut(AItemIndex: Integer)**

  ZoomOut puts the item with index AItemIndex back in normal state

- **function Zoomed: Boolean**

  Returns true when the item is zoomed.

- **InitItems(AThread: TMultiTouchThread = nil) , UpdateItems, ResetItems**

  InitItems initializes all items for a TMultiTouchRegion, i.e. all transforms are reset to initial state.

  UpdateItems rebuilds the internal draw stack from items in the items collection. This will force an update of the displayed items when for example many items are added/removed/changed in the items collection.

  ResetItems resets all items to show the main content (and not detail content)

- **function ItemsPerPage: integer**

  Returns the number of items that are shown on a page.

- **PageIndexForItemIndex(Index: Integer): Integer**

Returns the page number for an item with ItemIndex.

- **function GetItemAtPoint(Point: TD2D1Point2F; CountInternalItem: Boolean = False): TMultiTouchItem**

  Returns the item a specific location Point or nil when no item is found.

- **function SelectedItem: TMultiTouchItem**

  Returns the selected item, nil if none is selected.

- **function RegionTransform: TD2D1Matrix3x2F**

  Returns the transform that is applied to the background region when it is enabled for moving, rotating or scaling. This transform is also automatically applied to all items in this region to make the items move along with the background region.

- **GetSTItem(Index: Integer): TMultiTouchItem**

  Retrieves an item from the draw stack at a certain index

```
stItemMain := GetSTItem(i).MainItem;

stItemDetail := GetSTItem(i).DetailItem;
```

- **function CountNeedInitialization: Integer**

  Counts the number of items that (still) need to be initialized. This can be used to track the progress of the background thread to initialize items. When result is zero, the background thread has finished processing.

- **XYToItemControlItem(X, Y: Integer): TItemControlRecord**

  Returns a record containing the control item instance and the item instance when a control item is found at position X,Y.

- **XYToControlItem(X, Y: integer; r:TD2DRectF; Controls: TMultiTouchControlItems; BackGround: Boolean): TMultiTouchCustomItem**

  Returns an instance of a control item when found at position X,Y on the background region, otherwise it returns nil.

TMultiTouchRegion events

- **OnAfterCloseStack:** Event occurs just after the stack is closed. The event returns the top item of the stack clicked and the stack index.
- **OnAfterOpenStack:** Event occurs just after the stack is opened. The event returns the top item of the stack clicked and the stack index.
- **OnBeforeCloseStack:** Event occurs just before the stack is closed. The event returns the top item of the stack clicked and the stack index. With the Allow parameter, setting it to false will stop the stack close operation.
- **OnBeforeOpenStack:** Event occurs just before the stack is opened. The event returns the top item of the stack clicked and the stack index. With the Allow parameter, setting it to false will stop the stack open operation.
- **OnControlItemClick**: Event occurs when a control item is clicked. The event returns the control item clicked.
- **OnDrawControlItem:** Event occurs when a control item is drawn. It allows performing custom drawing of the control item via the D2DCanvas parameter with the rectangle R of the item. When the AllowDraw parameter is set to false, the built-in control item drawing is no longer executed.
- **OnDrawControlItemText:** Event occurs when a control item text is drawn. It allows performing custom drawing of the item's caption. When the AllowDraw parameter is set to false, the built-in control item's caption drawing is no longer executed.
- **OnStartDrawItem**: Event occurs when a normal item is about to be drawn. Performing drawing from this event will be done before the built-in item drawing is done.
- **OnEndDrawItem**: Event occurs after a normal item has been drawn. Performing drawing from this event will draw on top of the component's built-in drawing.
- **OnDrawItemText**: Event occurs before an item's text is drawn. It allows performing custom drawing of the item's text. When the AllowDraw parameter is set to false, the built-in item's text drawing will not be executed.
- **OnDrawRegion**: Event occurs when region is drawn. It allows performing custom drawing of the background region.
- **OnItemClick**: Event occurs when a normal item is clicked.
- **OnItemControlItemClick**: Event occurs when an item's control is clicked. It returns both the instance of the control item and the item with which this control item is associated.

- **OnItemCustomLoad**: Event occurs when the component's internal thread is loading items. It allows overriding the component's built-in loading of content. Via the APicture parameter for example, the image for the item can be set in the event handler instead of heaving the thread load it from file or via a visualizer.
- **OnItemDblClick**: Event occurs when an item is double clicked.
- **OnItemDetailCustomLoad**: Similar to the OnItemCustomLoad event but for the detail content of the item.
- **OnItemFill**: Event occurs before an item will be drawn. The event returns both the item and the fill with which it will be drawn. Default, the fill is equal to TMultiTouchRegion.ItemAppearance.Fill. From this event, the fill can be dynamically changed to set a specific fill for some specific items.
- **OnItemMouseDown**: Event occurs on mouse down on an item.
- **OnItemMouseMove**: Event occurs on mouse move over an item.
- **OnItemMouseUp**: Event occurs on mouse up on an item.
- **OnItemMove**: Event occurs when an item is moved via touch manipulation. The X, Y parameters return the new position the item will be moved to. Via these var parameters, the move position can be dynamically changed.
- **OnItemRotate**: Event occurs when an item is rotated via touch manipulation. The Rotation parameter is the angle of rotation and can be dynamically changed.
- **OnItemScale**: Event occurs when an item is scaled. The Scale var parameter returns the amount of scaling and can be dynamically changed.
- **OnItemSelectedChanged**: Event occurs when the selected item changes. It returns the new selected item.
- **OnMouseDown**: Event occurs on mouse down.
- **OnMouseMove**: Event occurs on mouse move.
- **OnMouseUp**: Event occurs on mouse up.
- **OnPageChange**: Event occurs when the page change via a slide touch manipulation. It returns the original page index and the new page index. Via the var Allow parameter, the page change can be stopped by setting this to false.
- **OnResize**: Event occurs when the TMultiTouchRegion control is resizing.

- **OnSetInertiaParameters**: Event occurs when the TMultiTouchRegion needs to initialize the inertia parameters. It passes an IInertiaProcessor interface instance. All parameters of the inertia can be changed via this interface.
- **OnThreadDone**: Event occurs when the TMultiTouchRegion internal thread has finished initializing all normal items.

TMultiTouchRegion mouse support

- **Mouse**

  The MultiTouch SDK is intended to be used with a touch-screen Mouse support is only integrated to enable developers who develop without a touch-screen to use and test the component. Most mouse operations are straightforward but using the mouse to perform rotation might not be so obvious. Perform a double click with the mouse on an item to set the rotation point and then click & drag the item to perform the rotation around this rotation point. Double click the item again to stop the rotation.

TMultiTouchItem properties

- **AspectRatio:** When set to true, keeps the height/width ratio of the item's image while resizing.
- **CanMoveHorizontal:** When set to true, an item can be moved horizontally.
- **CanMoveVertical:** When set to true, an item can be moved vertically.
- **CanRotate:** When set to true, the item can be rotated.
- **CanScale:** When set to true, the item can be scaled.
- **DetailItem:** An item has default content and detail content with flip animation between default and detail, when EnableFlip is set to true. The DetailItem property defines the detail content. A complete list of properties of this class can be found in the paragraph about TDefaultDisplayInfo.
- **EnableFlip:** When set to true, an item can flip (turn around its vertical axis) between default content and detail content.
- **ItemHeight:** Height of the item.
- **ItemWidth:** Width of the item.
- **ItemX:** Horizontal position in the region
- **ItemY:** Vertical position in the region
- **MainItem:** An item has default content and detail content with flip animation between default and detail, when EnableFlip is set to true. This property defines the default content. A complete list of properties of this class can be found in the paragraph TDefaultDisplayInfo.
- **Resizable:** When set to true, the item can be resized
- **ResizeHandleSize:** When TMultiTouchItem.Resizable is set to true, an icon appears in the right bottom corner of the item that indicates the item can be resized by dragging this corner. The size of this icon can be set with this property.
- **StackIndex:** This property defines the stack the item belongs to.

TDefaultDisplayInfo properties

- **AspectRatio:** When set to true, keeps the height/width ratio of the item's image while resizing.
- **BackGround:** When set to true, a background is displayed around the item's content.
- **BackGroundShadow:** When set to true, a background shadow is created for the item.
- **Caption:** Sets the optional caption text that can be displayed along an item's content.
- **ContentShadow:** When set to true, a shadow is displayed for the image / text content.
- **ContentType:** The content type can be any of the values: ctImage, ctImageText, ctText.
- **Controls:** Collection of controls that can be associated with a TMultiTouchItem.
- **DetectTransparentBackGround:** When set to true, will only detect as a click on an item when the click happened on the non-transparant part of the .PNG image. For other image formats, the click is detected when the item is clicked anywhere within its rectangle.
- **FileName:** Holds the name of the file associated with the item.
- **ImageHeight:** Height of the displayed item when ImageLocation is set to ilCustom.
- **ImageLeft:** Left position of the displayed item when ImageLocation is set to ilCustom.
- **ImageLocation:** A set of predefined image locations for the item. The image location can be any of the values: ilTopLeft, ilTopCenter, ilTopRight, ilCenterLeft, ilCenterCenter, ilCenterRight, ilBottomLeft, ilBottomCenter, ilBottomRight, ilCustom.
- **ImageMargin:** Margin around the image.
- **ImageTop:** Top position of the displayed item when ImageLocation is set to ilCustom.
- **ImageWidth:** Width of the displayed item when ImageLocation is set to ilCustom.
- **Text:** Holds the text to be displayed when the item's main content is shown.
- **TextLeft:** Text left position when TextLocation is set to ilCustom.
- **TextLocation:** A set of predefined text locations within the item. The text location can be any of the values: ilTopLeft, ilTopCenter, ilTopRight, ilCenterLeft, ilCenterCenter, ilCenterRight, ilBottomLeft, ilBottomCenter, ilBottomRight, ilCustom.
- **TextTop:** Text top position when TextLocation is set to ilCustom.

TMultiTouchTopLayerItem properties

- **Backgroundlayer:** When set to true, the control item is positioned on the background below normal items in the region. When it is false, it will always be displayed on top of normal items.
- **Fill:** A set of properties to define the look of the different TMultiTouchControlItems. The full list of properties for the control item class can be found in the paragraph about TMultiTouchRegionItemAppearance.
- **FillDown:** Identical to Fill, but defining the settings for the down state of a control item.
- **Height:** Height of the control item.
- **ItemObject:** Can be used to assign any object to a control item.
- **Left:** Left position of the displayed control item when Location is set to ilCustom.
- **Location:** The location of the picture can be any of the values: ilTopLeft, ilTopCenter, ilTopRight, ilCenterLeft, ilCenterCenter, ilCenterRight, ilBottomLeft, ilBottomCenter, ilBottomRight, ilCustom.
- **Tag:** Stores an Integer value. Tag is an Integer property that has no predefined meaning. It can be used for storing an additional integer value, or it can be typecast to any value such as a component reference or a pointer.
- **Text:** Text to be displayed on the control item.
- **TextLeft:** Text horizontal position offset, can be positive and negative.
- **TextLocation:** The text location can be any of the values: ilBottemLeft, ilBottomCenter, ilBottomRight, ilCenterLeft, ilCenterCenter, ilCenterRight, ilTopLeft, ilTopCenter, ilTopRight.
- **TextTop:** Text vertical position offset, can be positive and negative
- **Top:** Top position of the displayed control item when Location is set to ilCustom.
- **Visible:** When set to true, the control item is visible.
- **Width:** Width of the control item.

TMultiTouchRegionItemAppearance properties

- **Font:** Sets the font to use
- **Fill:** Set of properties to control the appearance of the item
  - **BackGroundPicture**: the background picture filename when GradientType gtTexture is chosen.
  - **BackGroundPictureAspectMode**: The backgroundpicture aspect mode can be any of the values: pmNormal, pmStretch.
    - **pmNormal**: The background picture will be positioned on the fill.
    - **pmStretch**: The background picture will be stretched on the fill.
  - **BackGroundPictureAspectRatio:** the height/width ratio is kept as the original item.
  - **BackGroundPictureLeft**: when the position is set to ppCustom the left position can be set with this property.
  - **BackGroundPictureMode:** The background picture mode can be any of the values: pmOutsideFill, pmInsideFill.
    - **pmOutsideFill:** When chosen, the fill will extend outside the borders of the control.
    - **pmInsideFill:** When chosen, the fill will be limited to the borders of the control.
  - **BackGroundPicturePosition**: The background picture position can be any of the values: ppTopLeft, ppTopCenter, ppTopRight, ppBottomLeft, ppBottomCenter, ppBottomRight, ppTiled, ppStretched, ppCenterLeft, ppCenterCenter, ppCenterRight, ppCustom.
  - **BackGroundPictureTop**: the top position of the background picture when the BackgroundPicturePosition is set to ppCustom.
  - **BorderColor:** the color of the border of the fill.
  - **BorderOpacity**: the opacity of the border of the fill.
  - **BorderWidth**: the width of the border of the fill.
  - **Color**: the start color of the top part gradient (if the GradientType is gtSolid, Color is the only property used).
  - **ColorMirror**: when ColorMirror is set to a color different from clNone the fill will be split up in 2 parts: the top part and the mirror bottom part. ColorMirror is the start color of the mirror bottom part.
  - **ColorMirrorTo**: the end color of the mirror bottom part.
  - **ColorTo**: the end color of the top part gradient.
  - **Glow:** The glow property can be any of the values: gmNone, gmGradient.
    - **gmNone:** When chosen, no glow effect is used.
    - **gmGradient**: When chosen, a gradient glow effect Is used
  - **GlowGradientColor:** Used color for the glow effect when gmGradient is chosen.
  - **GlowRadialColor:** Used Radial Color when gmGradient is chosen.
  - **GradientMirrorType**: the type of gradient of the mirror bottom part.

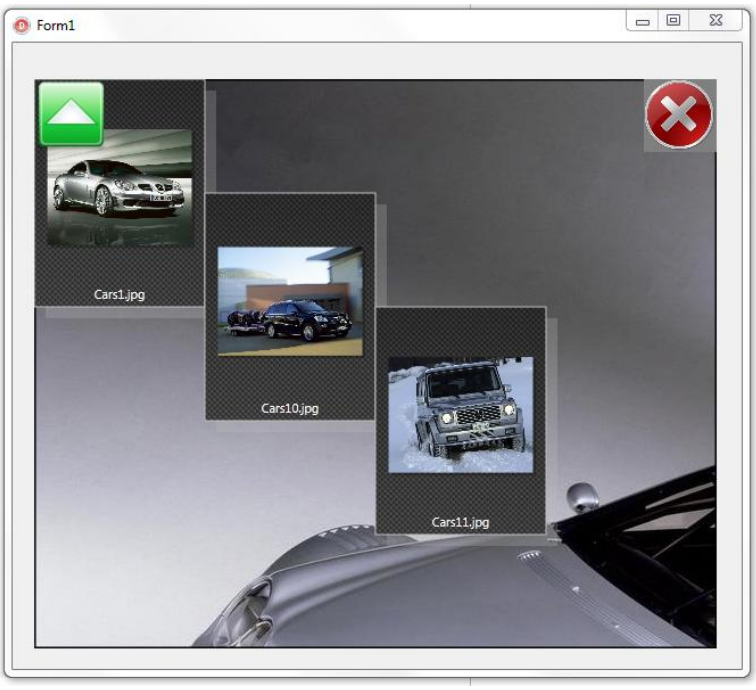
- **GradientType**: The gradient type for the top part (or full background if mirror colors are set to clNone) can be any of the values: gtSolid, gtRadial, gtVertical, gtHorizontal, gtForwardDiagonal, gtBackwardDiagonal, gtTexture, gtNone.
- **Opacity**: the opacity of the start color of the top part.
- **OpacityMirror**: the opacity of the start color of the mirror bottom part.
- **OpacityMirrorTo**: the opacity of the end color of the mirror bottom part.
- **OpacityTo**: the opacity of the end color of the top part.
- **Picture**: you can always set a picture that is not bound to the rectangle of the fill. In other words you can draw a picture which overlaps the fill.
- **PictureAspectMode:** The picture aspect mode can be any of the values: pmStretch, pmNormal.
  - **pmStretch:** The picture is stretched in order to fill the complete control.
  - **pmNormal:** The picture is shown in his original size.
- **PictureAspectRatio:** When pmStretch is selected, keep the picture in the original height/width ratio.
- **PictureHeight**: the height of the picture in case the picturesize is set to custom.
- **PictureLeft:** the left position of the picture in case pictureposition is set to custom.
- **PicturePosition**: The picture position can be any of the values: ppTopLeft, ppTopCenter, ppTopRight, ppBottomLeft, ppBottomCenter, ppBottomRight, ppTiled, ppStretched, ppCenterLeft, ppCenterCenter, ppCenterRight, ppCustom.
- **PictureSize**: it can be useful to resize the picture to a different size when it is too large. Set PictureSize to custom and use PictureWidth and PictureHeight to change the size of the picture.
- **PictureTop:** the top position of the picture in case PicturePosition is set to ppCustom.
- **PictureWidth**: the width of the picture in case the PictureSize is set to ppCustom.
- **Rounding:** the rounding of the fill, set Rounding = 0 to have a rectangular shape and a higher value to have more rounded corners.
- **RoundingType**: the type of rounding of the fill. In some cases it can be useful to only set the specific corners of the fill to be rounded. The rounding type can be any of the values: rtNone, rtTop, rtBottom, rtBoth, rtLeft, rtRight.
- **ShadowColor**: the color of the shadow of the fill.
- **ShadowOffset**: the offset of the shadow of the fill.
- **ShadowType**: The ShadowType can be any of the values: stRightBottom, stBottom or stSurround.
  - **stRightBottom:** shadow is drawn both at bottom side and right side of item
  - **stBottom:** shadow is drawn only on bottom side of item
  - **stSurround:** shadow is drawn on all sides of item

- **FillSelected :** a set of properties (identical to those of the above Fill), can be set

TMultiTouchRegion sample code

In the sample below, following properties have been set programmatically to control the appearance of items:

```
      MultiTouchRegion1.Fill.Borderwidth:=3;
      MultiTouchRegion1.Fill.BorderColor:=clblack;
      MultiTouchRegion1.Fill.BackGroundPicture.LoadFromFile(DemoFldr +
'Cars\Cars19.jpg');
      MultiTouchRegion1.Fill.BackGroundPictureAspectMode := pmStretch;
      MultiTouchRegion1.Fill.BackGroundPictureAspectratio := True;
      MultiTouchRegion1.Fill.BackGroundPictureMode := pmInsideFill;
      MultiTouchRegion1.Fill.BackGroundPicturePosition:= ppTopLeft;
      MultiTouchRegion1.ItemAppearance.Fill.BackGroundPicture.LoadFromFile(
IconFldr + 'bg.jpg');
      MultiTouchRegion1.ItemAppearance.Fill.BackGroundPictureAspectMode :=
pmStretch;
      MultiTouchRegion1.ItemAppearance.Fill.BackGroundPictureAspectratio :=
True;
      MultiTouchRegion1.ItemAppearance.Fill.BackGroundPictureMode :=
pmInsideFill;
      MultiTouchRegion1.ItemAppearance.Fill.BackGroundPicturePosition:=
ppCenterCenter;
      MultiTouchRegion1.ItemAppearance.Font.Name:='EuroStyle';
      MultiTouchRegion1.ItemAppearance.Font.Color:=clWhite;
```



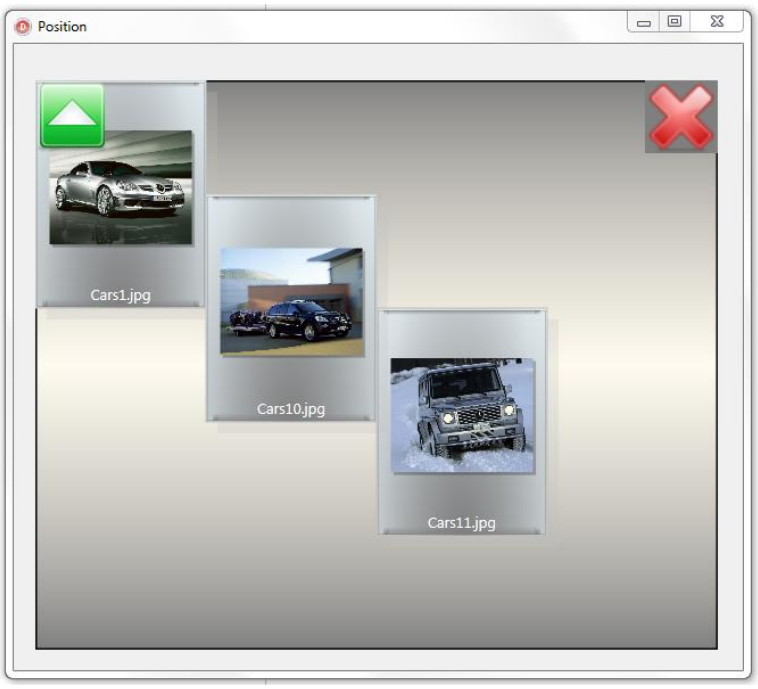
Another example, showing backgroundfill with gradients, items fill with glow and radial gradients

```
      MultiTouchRegion1.Fill.GradientType := gtVertical;
      MultiTouchRegion1.Fill.GradientMirrorType := gtVertical;
      MultiTouchRegion1.Fill.Color := clGray;
      MultiTouchRegion1.Fill.ColorTo := clCream;
```

```
MultiTouchRegion1.Fill.ColorMirror := clCream;
MultiTouchRegion1.Fill.ColorMirrorTo := clGray;

MultiTouchRegion1.ItemAppearance.Fill.Glow:= gmRadialGradient;
MultiTouchRegion1.ItemAppearance.Fill.GradientType:= gtradial;
MultiTouchRegion1.ItemAppearance.Fill.GradientMirrorType:= gtradial;

MultiTouchRegion1.ItemAppearance.Fill.GlowGradientColor:= clGray;
MultiTouchRegion1.ItemAppearance.Fill.GlowRadialColor:= clGray;
```



Visualizers

The visualizer concept allows to display any kind of file type, the default visualizer is set to TMultiTouchRegionVisualizer.

Currently there are 2 extra visualizers that are supported:

- **TMultiTouchRegionPreviewVisualizer:** Extends the default visualizer with preview support.
- **TMultiTouchRegionPDFVisualizer:** Extends the default visualizer with a PDF viewer.

TMultiTouchRegionVisualizer

TMultiTouchRegionVisualizer description

The TMultiTouchRegionVisualizer unit defines a base visualizer class from where other custom visualizer classes can derive to create new visualizers which draw the items in the page. TMultiTouchRegionVisualizer is designed to display and animate highly configurable items, with complex appearances.

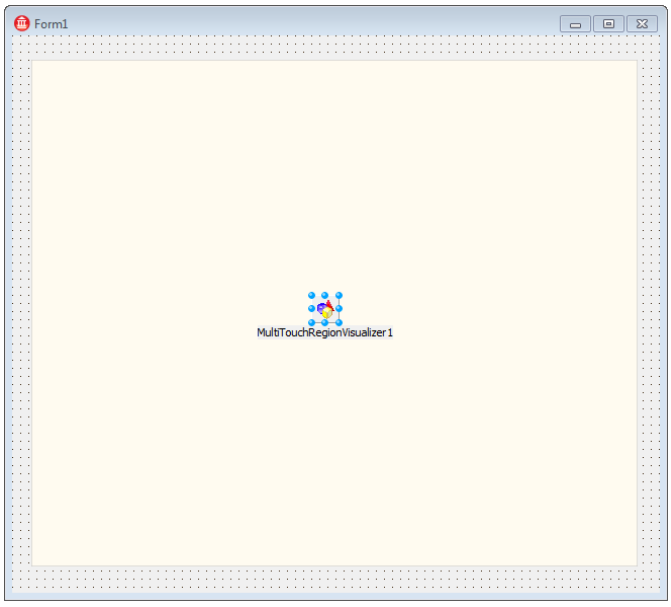
TMultiTouchRegionVisualizer features

TMultiTouchRegionVisualizer is the default visualizer. It is capable of displaying images of the following type: BMP,GIF,JPEG,PNG,ICO as well as optionally text.

TMultiTouchRegionVisualizer use

First use

Drop the TMultiTouchRegionVisualizer control on your TMultiTouchRegion component.



Then select your TMultiRegionVisualizer via the TMultiTouchRegion.Visualizer property. Note that by default TMultiTouchRegion internal already uses the TMultiTouchRegionVisualizer by default so it is not strictly necessary to assign this visualizer. When no visualizer is assigned it will use an internally created TMultiTouchRegionVisualizer.

TMultiTouchRegionVisualizer properties

- **Name:** Name of the control.

- **Tag:** Stores an Integer value. Tag is an Integer property that has no predefined meaning. It can be used for storing an additional integer value, or it can be typecast to any value such as a component reference or a pointer.

TMultiTouchRegionVisualizer Events

- **OnItemCustomize:** Event called when loading an item via threading. This event can be helpful to further customize the item properties.

TMultiTouchRegionPreviewVisualizer

TMultiTouchRegionPreviewVisualizer description

TMultiTouchRegionPreviewVisualizer is a class that draws file previews in different display modes. TMultiTouchRegionPreviewVisualizer is designed to visually display and animate highly configurable items, with complex appearances. The TMultiTouchRegionPreviewVisualizer is a derived class from the base class TMultiTouchRegionvisualizer.

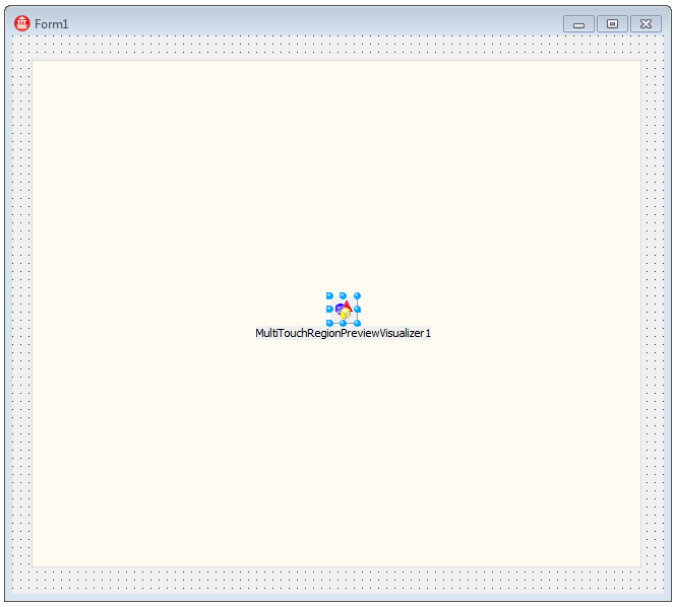
TMultiTouchRegionPreviewVisualizer features

The TMultiTouchRegionPreviewVisualizer is designed to show a Windows shell preview of files. It uses the shell preview API to retrieve and display file types like PDF, MS Word, MS Powerpoint etc… Note that the way the preview is shown is highly dependent on the preview handlers installed in the OS.

TMultiTouchRegionPreviewVisualizer use

First use

Drop the TMultiTouchRegionPreviewVisualizer control on your TMultiTouchRegion component.



Then select your TMultiTouchRegionPreviewVisualizer in the TMultiTouchRegion.Visualizer property.

TMultiTouchRegionPreviewVisualizer properties

- **Name:** Name of the control.
- **PreviewHeight:** Height of the shell preview image to retrieve
- **PreviewWidth:** Width of the shell preview image to retrieve

TMultiTouchRegionPreviewVisualizer events

- **OnItemCustomize:** Event called when loading an item via threading. This event can be helpful to further customize the item properties.

TMultiTouchRegionPDFVisualizer

TMultiTouchRegionPDFVisualizer description

TMultiTouchRegionPreviewVisualizer is a class that draws PDF-files in different display modes. TMultiTouchRegionPreviewVisualizer is designed to display and animate highly configurable items, with complex appearances. The TMultiTouchRegionPreviewVisualizer is a derived class from the base class TMultiTouchRegionvisualizer.

TMultiTouchRegionPDFVisualizer features

- Complete PDF-files can be loaded at once, or one page at a time.
- Contains a Zoom In / Zoom Out functionality

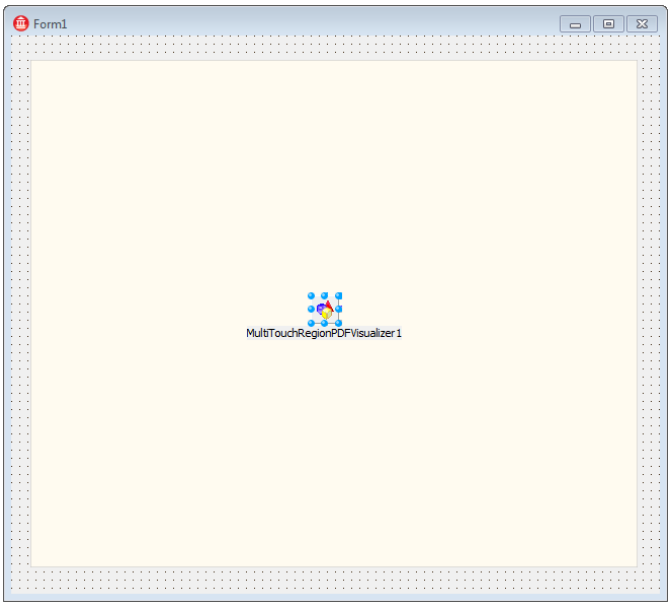
TMultiTouchRegionPDFVisualizer use

First use

TMultiTouchRegionPDFVisualizer uses the QuickPDF library, which has to be downloaded from http://www.quickpdf.org/.
Make sure to you use your own 25 character registration key, key will be provided by Denebu upon download of quickPDF.

```
const
  quickPdfKey = 'j6xxxxxxxxxxxxxxxxxxxxx8y';
```

Drop the TMultiTouchRegionPDFVisualizer control on your TMultiTouchRegion component.



Then select your TMultiTouchRegionPDFVisualizer in the TMultiTouchRegion.Visualizer property.

TMultiTouchRegionPDFVisualizer properties

- **AutoCreatePages:** When set to True, all pages from de PDF-File will be loaded (in a separate thread).
- **FileName:** Name of the PDF-File to load.
- **Name:** Name of the control
- **Tag:** Stores an Integer value. Tag is an Integer property that has no predefined meaning. It can be used for storing an additional integer value, or it can be typecast to any value such as a component reference or a pointer.
- **UnlockKey:** For proper functioning of QuickPDF, you have to provide your own registration key to this property.
- **Resolution:** The bitmap-resolution can be any of the values: prHigh, prMedium, prLow.

TMultiTouchRegionPDFVisualizer events

- **OnItemCustomize:** Event called when loading an item via threading. This event can be helpful to further customize the item properties.
- **OnPageLoaded :** Event occurs when a page is loaded.

Advanced techniques

In the TMS MultiTouch SDK some methods, properties and events are made public to give an extra touch to the application you are developing.

To explain some more advanced techniques that can be achieved with the TMS MultiTouch SDK, such as custom drawing and interaction, we need to take a look at the Music Browser demo. When browsing through the demo, you will notice that the albums page has an extra feature to display tracks.

Clicking on an album displays the tracks on the right side, which behave exactly as the album in terms of scaling, rotation and movement. Interaction with the album reflects on the track list as displayed below.

As the selected track plays ,the progress is displayed by painting a rectangle with the size of the text, and is calculated on the track position and length. Switching tracks can be done by simply clicking on a different track.

The tracklist is an array of tracklistitems that are linked to the DataObject property of the album. Whenever the album changes, the tracklist is rebuilt with the corresponding tracks. A tracklistitem holds the information to quickly access the correct track data.

```
TTrackListItem = class(TPersistent)
private
  FFileName: String;
  FName: String;
  FID: Integer;
public
  property Name: String read FName write FName;
  property FileName: String read FFileName write FFileName;
  property ID: Integer read FID write FID;
end;
```

Filling the tracklist is done when clicking on the album with the following code:

```
procedure TForm1.LoadTracks(AlbumID: Integer; AlbumItem:
TMultiTouchItem);
var
  I: integer;
  lst: TObjectList;
  trlstit: TTrackListItem;
  str,fname: String;

begin
  //Fetching the correct tracks from the Database.
  MultiTouchRegion1.BeginUpdate;
  ADOQuery1.SQL.Clear;
ADOQuery1.SQL.Add('SELECT * from tblTracks WHERE AlbumID = ' +
inttostr(AlbumID));
  ADOQuery1.ExecSQL;
  ADOQuery1.Open;
  ADOQuery1.First;

//Checking and cleaning up possible existing trackitem (which
holds the track data and is linked to the album).
  if Assigned(TrackItem) then
  begin
    if Assigned(TrackItem.MainItem.DataObject) then
    begin
      lst := TObjectList(TrackItem.MainItem.DataObject);
      lst.Free;
    end;
    TrackItem.Free;
    for I := 0 to MultiTouchRegion1.Items.Count - 1 do
    begin
      MultiTouchRegion1.Items[I].LinkedItem := nil;
    end;
  end;

//Creating a new TrackItem and setting appearance and data
properties.

  I := 1;
  TrackItem := MultiTouchRegion1.Items.Add;
  with TrackItem do
  begin
    MainItem.ContentType := ctText;
```

```
//Initializing the Track BackGroundTransform and
//ContentTransform with the Album BackGroundTransform and
//ContentTransform for later use.
MainItem.BackGroundTransform :=
AlbumItem.MainItem.BackGroundTransform;
    MainItem.ContentTransform := MainItem.BackGroundTransform;
//
    MainItem.TextLocation := ilCenterLeft;
    MainItem.BackGround := False;
    CanMoveVertical := False;
    MainItem.DetectBackGround := False;
    CanRotate := False;
    CanScale := False;
    CanMoveHorizontal := False;
    Resizable := False;
    EnableFlip := False;
    lst := TObjectList.Create;

    //Loading the tracks
    while not ADOQuery1.Eof do
    begin
str := inttostr(I + 1) + '. '+
ADOQuery1.FieldByName('TrackName').AsString;
      MainItem.Text := MainItem.Text + str;
      MainItem.Text := MainItem.Text + #13#10;
      trlstit := TTrackListItem.Create;
      trlstit.name := ADOQuery1.FieldByName('TrackName').AsString;

trlstit.FileName :=
ADOQuery1.FieldByName('TrackFileName').AsString;
      trlstit.ID := ADOQuery1.FieldByName('ID').AsInteger;
      lst.Add(trlstit);
      ADOQuery1.Next;
      Inc(I);
    end;

    MainItem.DataObject := lst;
  end;

  MultiTouchRegion1.EndUpdate;
  AlbumItem.LinkedItem := TrackItem;
```

```
//Playing the first track and displaying the top-layer control
item.

  if CurrentAlbum <> AlbumItem.MainItem.Tag then
  begin
    CurrentAlbum := AlbumItem.MainItem.Tag;
    TrackAlbum := CurrentAlbum;
    if Assigned(TrackItem) then
    begin
      lst := TObjectList(TrackItem.MainItem.DataObject);
      if Assigned(lst) then
      begin
        CurrentTrack := 0;
        HideTrack;
        if lst.Count > 0 then
        begin
          fname := GlobalPath + 'Tracks\' +
TTrackListItem(lst.Items[0]).FileName;

          if FileExists(fname) then
          begin
            MediaPlayer1.FileName := fname;
            MediaPlayer1.Open;
            MediaPlayer1.Play;
            mediaopened := true;
            ShowTrack;
          end;
        end;
      end;
    end;
  end;

  MultiTouchRegion1.Invalidate;
end;
```

The method LoadTracks takes care of loading the tracks from the Database and creating a ready to use item, which holds the data and the information for displaying the tracks. Since the matrix transforms hold the position, rotation and scale values for that item, we can now integrate this in a custom drawing event: OnDrawItemText.

```
procedure TForm1.MultiTouchRegion1DrawItemText(Sender: TObject;
  D2DCanvas: TDirect2DCanvas; Item: TMultiTouchItem; AText:
string;
  R: D2D_RECT_F; var AllowDraw: Boolean);
var
  lst: TObjectList;
  lstItem: TTrackListItem;
  I: integer;
  LItem: TMultiTouchItem;
  tw, th: Integer;
  str: String;
begin
//Disable the normal text drawing for this TrackItem.
  AllowDraw := False;

  LItem := GetLinkedItem(TrackItem);

//Get the linked album item to set the correct transform.

  if Assigned(LItem) then
  begin

D2DCanvas.RenderTarget.SetTransform(LItem.MainItem.BackGroundTrans
form);

//Get the tracklist and display them.

    if Assigned(Item.MainItem.DataObject) then
    begin
      lst := TObjectList(Item.MainItem.DataObject);
      for I := 0 to lst.Count - 1 do
      begin
        lstItem := TTrackListItem(lst.Items[i]);

        D2DCanvas.Font.Size := 12;
        D2DCanvas.Font.style := [fsBold];

        str := inttostr(I + 1) + '. '+ lstItem.Name;

        if i = CurrentTrack then
        begin
          D2DCanvas.Font.Color := clBlack;
          D2DCanvas.Brush.Style := bsSolid;
          tw := D2DCanvas.TextWidth(str);
          th := D2DCanvas.TextHeight(str);
```

```
          if mediaopened and (MediaPlayer1.Length > 0) then
            tw := Round(tw * MediaPlayer1.Position /
MediaPlayer1.Length)
          else
            tw := 0;

          D2DCanvas.Brush.Color := clWhite;
          D2DCanvas.FillRect(Bounds(Round(LItem.ItemWidth - 50),
Round(I  * 30), D2DCanvas.TextWidth(str), th));
          D2DCanvas.Brush.Color := clLime;
          D2DCanvas.FillRect(Bounds(Round(LItem.ItemWidth - 50),
Round(I  * 30), tw, th));

          D2DCanvas.Brush.Style := bsClear;
        end
        else
        begin
          D2DCanvas.Brush.Style := bsClear;
          D2DCanvas.Font.Color := clWhite;
        end;

        D2DCanvas.TextOut(Round(LItem.ItemWidth - 50), Round(I  *
30), str);
      end;
    end;
  end;
end;
```

When starting the application and browsing to the albums you will now see the tracks when selecting an album. However, there is no interaction when clicking on a track. The interaction must also be handled separately since the tracklist is a custom part of the application. The interaction is handled in the OnItemMouseUp event.

```
procedure TForm1.MultiTouchRegion1ItemMouseUp(Sender: TObject;
  Item: TMultiTouchItem; TouchPoints: array of TOUCHINPUT);
begin
  DetectTrack(TouchPoints[0]);
end;
```

The TouchPoints parameter is an array of points that are registered when using a touch screen application and releasing your touch on an item. For non-touch based applications the TouchPoints array only contains one point.

If we look at the DetectTrack method, which handles the interaction, you will see that we are using similar code as the custom track drawing. This is necessary for correct implementation.

```
procedure TForm1.DetectTrack(p: TOUCHINPUT);
var
  LItem: TMultiTouchItem;
  lst: TObjectList;
  I: integer;
  geo: ID2D1RectangleGeometry;
  lstItem: TTrackListItem;
  tw, th: Integer;
  rItem: TD2DRectF;
  str,fname: string;
  pos: TPoint;
  itemclick: Bool;
begin
  if Assigned(TrackItem) then
  begin
    LItem := GetLinkedItem(TrackItem);
    if Assigned(LItem) and assigned(TrackItem.MainItem.DataObject)
then
    begin
      lst := TObjectList(TrackItem.MainItem.DataObject);
      for I := 0 to lst.Count - 1 do
      begin
        lstItem := TTrackListItem(lst.Items[i]);

        MultiTouchRegion1.D2DCanvas.Font.Size := 12;
        MultiTouchRegion1.D2DCanvas.Font.style := [fsBold];

        str := inttostr(I + 1) + '. '+ lstItem.Name;

        tw := MultiTouchRegion1.D2DCanvas.TextWidth(str);
        th := MultiTouchRegion1.D2DCanvas.TextHeight(str);
        rItem :=  D2D1RectF(LItem.ItemWidth - 50, Round(I  * 30),
LItem.ItemWidth - 50 +  tw, Round(I * 30) + th);
        D2DFactory.CreateRectangleGeometry(rITem, Geo);
        pos := ScreenToClient(Mouse.CursorPos);

        geo.FillContainsPoint(Point(p.x div 100, p.y div 100),
LItem.MainItem.BackGroundTransform, 1.0, itemClick);
        if itemClick then
        begin
          CurrentTrack := Max(0, Min(I, lst.Count - 1));
          fname := GlobalPath + 'Tracks\'+ lstItem.FileName;
```

```
            if FileExists(fname) then
            begin
              MediaPlayer1.FileName := fname;
              MediaPlayer1.Open;
              MediaPlayer1.Play;
              ShowTrack;
            end;
            Break;
          end;
        end;
      end;
    end;
  end;
```

The most important section in this code is the detection if the touchpoint is located in a specific area determined by the backgroundtransform.

```
geo.FillContainsPoint(Point(p.x div 100, p.y div 100),
LItem.MainItem.BackGroundTransform, 1.0, itemClick);
```

The geo variable is declared as a RectangleGeometry based on the trackitem rectangle, which contains the appropriate methods and functions to use for interaction.

In the custom drawing code the BackgroundTransform is used to set a starting position to draw the trackitems. If a transform is applied to the Direct2DCanvas all drawing code is relative to the transform. Remember that BackgroundTransform contains the correct position of the rectangle and is therefore used in combination with the geo variable.

If the correct track is located, the MediaPlayer component will then play the correct track.

TMultiTouch Samples

Electronic Board Demo description

The Electronic Board demo allows the user to make a design with the items shown on the left bottom side. This design can be saved, and re-opened for further use.

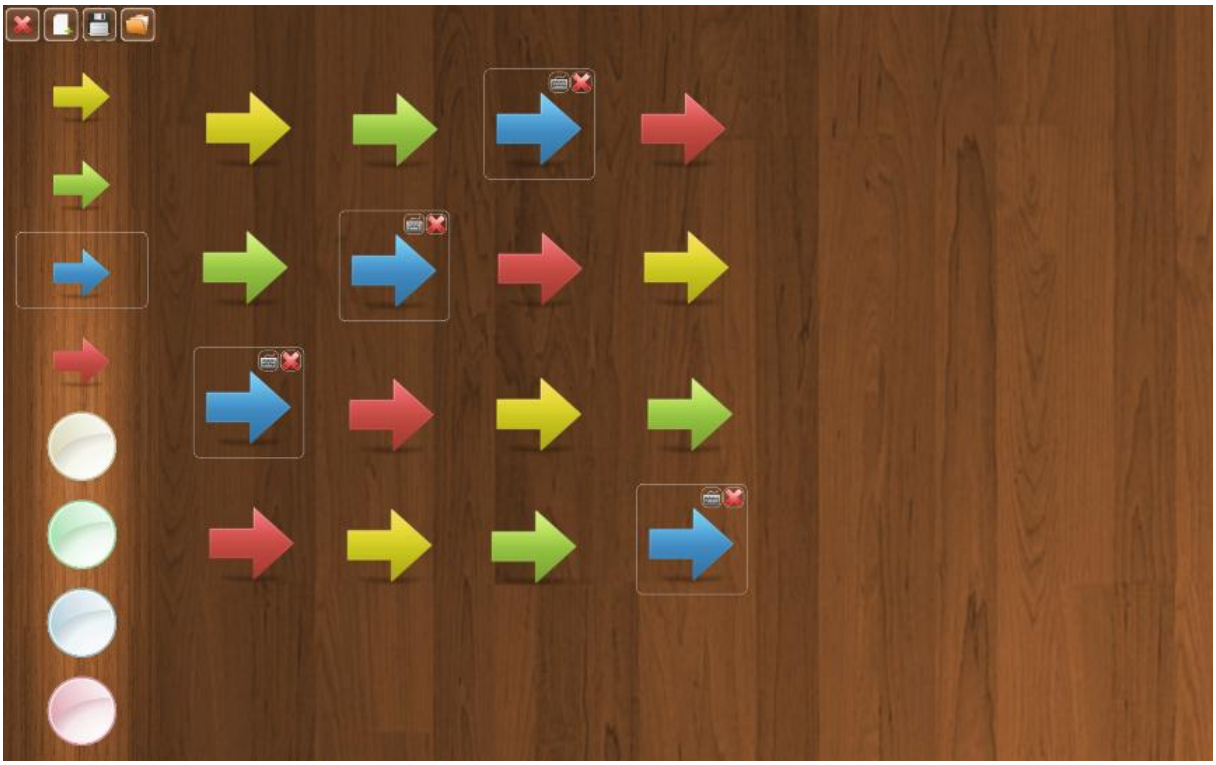
The items that are available for use in the design-mode, are found in a second TMultitouchRegion (leftmost list). Moving in between its items can be done with the touch of a finger, when tapping on the item in the left list, the item becomes selected. After this, a tap on the electronic drawing board, will create a new instance of the selected item and place it at touch position on the electronic board.

Electronic Board Demo features

The Electronic Board Demo gives an example of the possibility to save to file, and load from file. Selection of multiple items is demonstrated.
Notice that when selecting an item, movement is applied to that item. With a left-mouse-down on the region next to an item, the entire region is moved around. The position of the visible background within the entire region is displayed with scroll bars on the bottom and right side.

Electronic Board Demo screenshots

The main screen displays the different TMultiTouchRegion components and multiselect.



The second screenshot gives an example of a region that is moved in the control. Notice the scrollbars displaying the position in the control, and the bullets indicating out of sight items.

Music Browser Demo description

The Music Browser Demo program shows the different music genres to start with. When selecting one of them, a second screen is opened with artists belonging to the selected genre.

When the artist is clicked, all albums within the library from the artist are shown.

A click on the album loads the track-list for that album, and starts the playback of the album. For the clicked album, extra control items are being displayed to play a track or skip to a next or previous track.

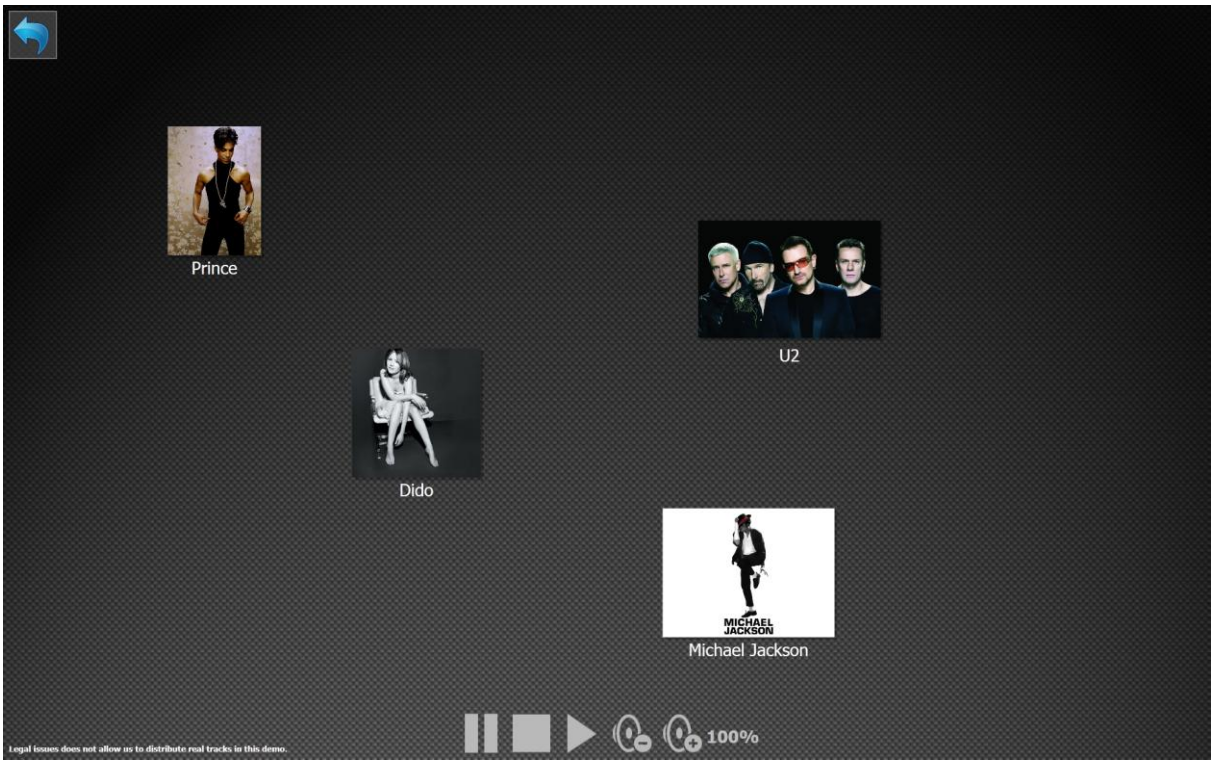
Music Browser Demo features

The Music Browser demo shows a variety of features of the TMultiTouchRegion at work. Control items are used to control the audio level or to exit the application. The genre items images are loaded from an Access database together with the transforms. The display mode is therefore dmTransform. This allows to retrieve the transform from the database and apply it. By saving the transform upon exit, this ensures that item location, size, rotation is persisted in the database.
The Music Browser demo gives an example on how to use the TMultiTouchControlItems, as a control to close the demo, and as back button.

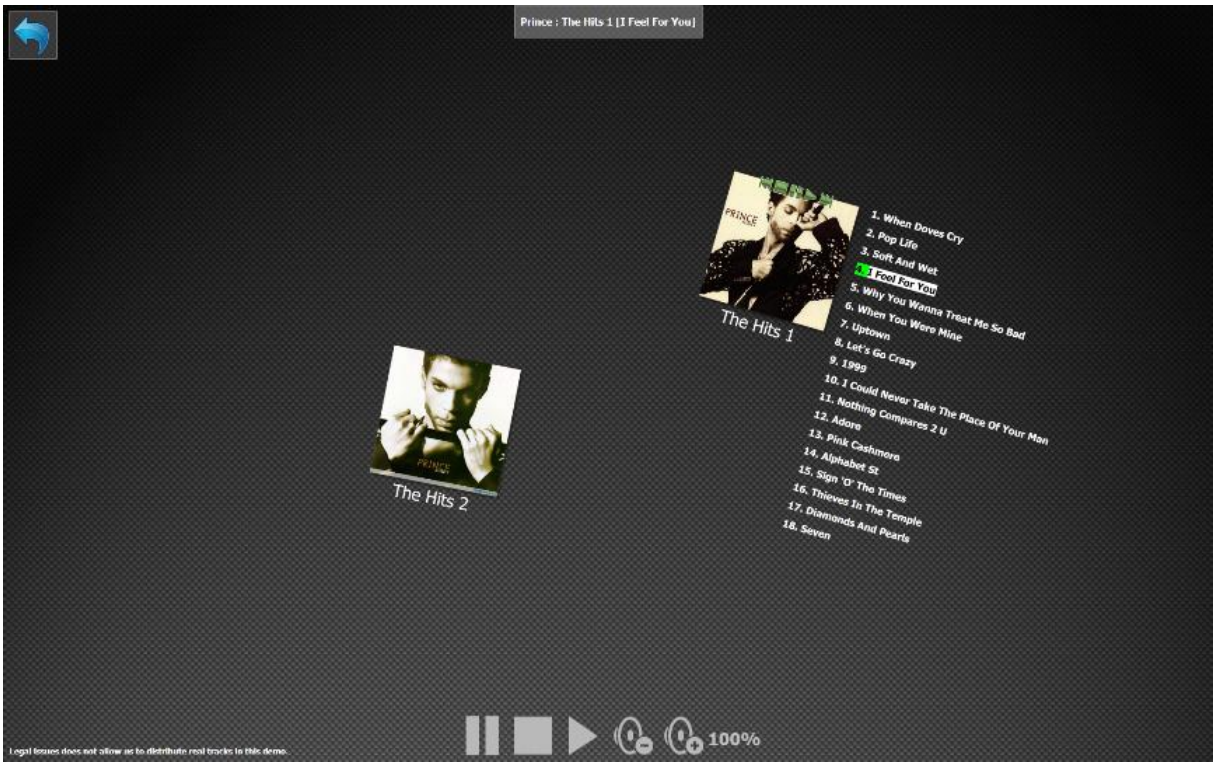
Music Browser Demo Screenshots

The main screen displays the different genres in random mode.

The genres screen displays the different artists in random mode.



When an item in the albums screen is clicked, it loads the tracks and displays the tracks next to the item. Via a custom drawn background for the track text, the progress of playing the track is shown.



Finally, the track titles are sensitive to clicks, meaning that it is sufficient to click on a track title to start playing it.

Photo Album Demo description

The Photo Album Demo program shows the pictures from the Windows Pictures Folder in a stack. The possibility is given to add more directories, when the '+' icon is clicked, a folder dialog is opened. When a stack is clicked, all pictures of this stack are shown in grid mode. Selecting one of these pictures will expand it to full-screen.

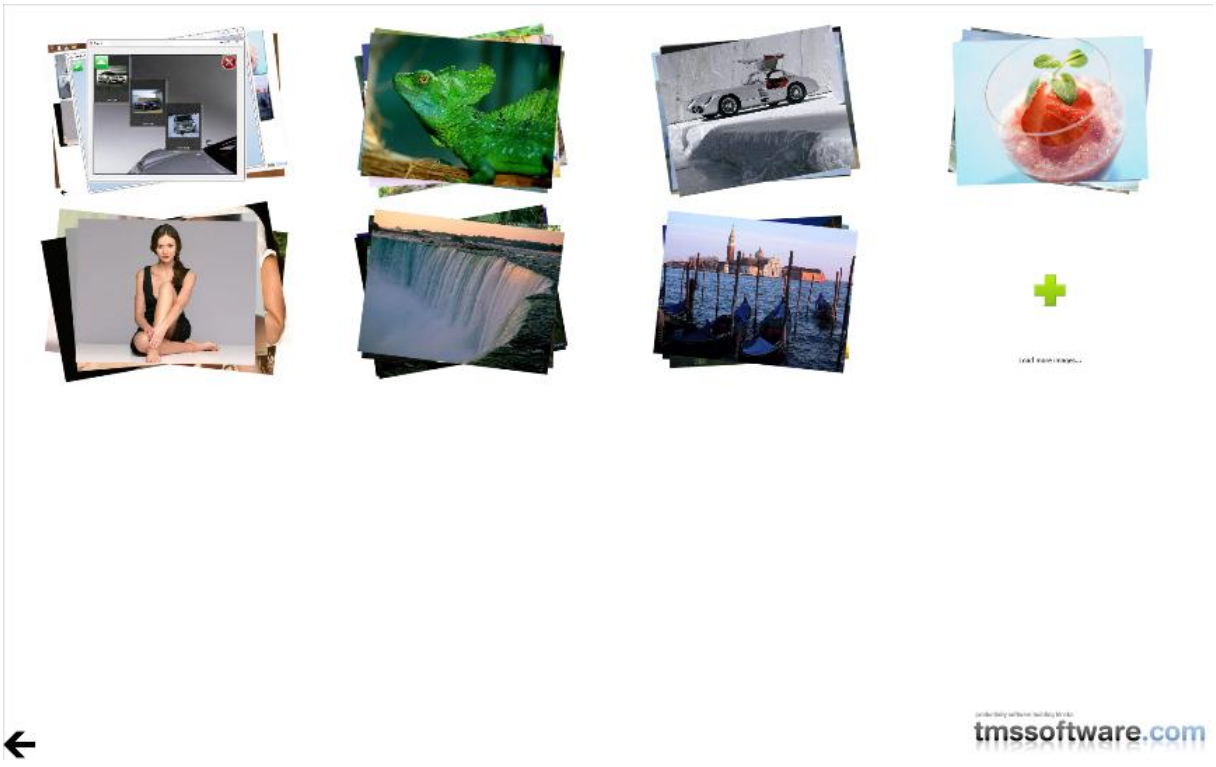
The Photo Album Demo displays the multiple touch features to scale and rotate the items.

Photo Album Demo features

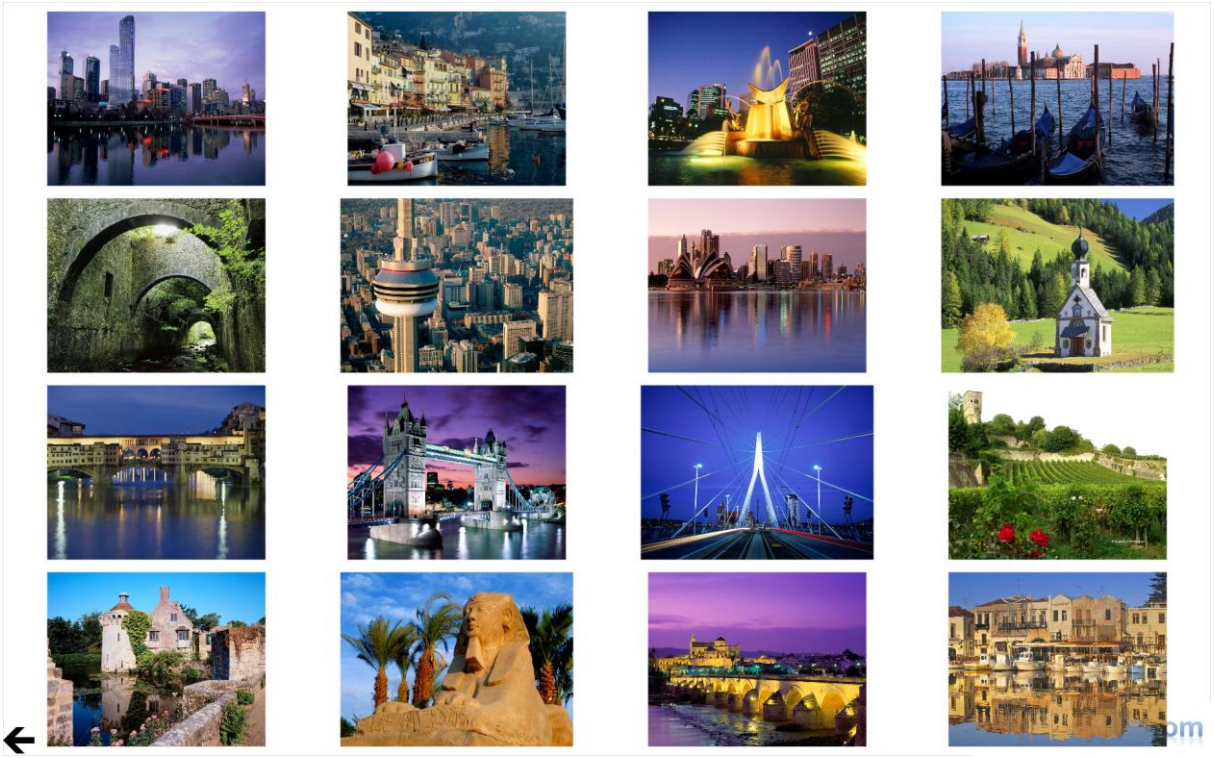
The Photo Album Demo explains how to create items in stacked-mode with the LoadImageStack method. The 'load more images' item gives an example on how to create an item manually.
The Photo Album Demo shows how to implement a TMultiTouchRegionControlItem.

Photo Album Demo screenshots

The main screen displays different folders in stacked mode.



When a stack is clicked, the Photo Album Demo opens the stack in grid mode.



A picture is shown in full screen when selected.

QuickPDF Demo description

The QuickPDF Demo program displays the different PDF files from the PDFFiles directory to start with. When selecting one of them, a second screen is opened with a double page detail from that file. One can swipe to the right to see the next pages, and swipe to the left to go to the previous pages.

When one of the pages is clicked, a detail of that page is shown. Detail pages can also be swiped.

When displaying pages and detail, a thumbnail view can be activated when clicking the left control.

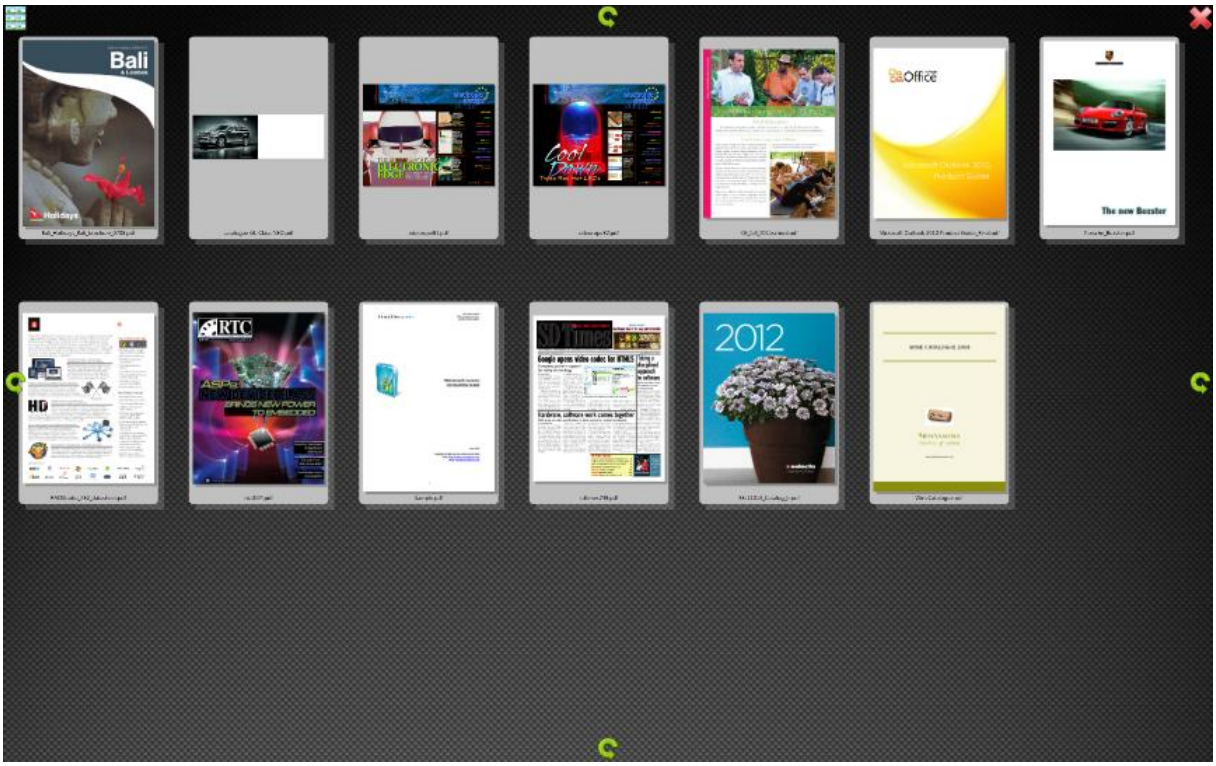
QuickPDF Demo features

The QuickPDF Demo demonstrates the use of the TMultiTouchRegionPDFVisualizer.
The QuickPDF Demo gives an example on how to use different TMultiTouchRegions in the same application. A second TMultiTouchRegion is used to display the thumbnails.
The demo explains the use of TMultiTouchControlItems. The left control on the PDF files screen toggles between the default random mode and position mode. The right control closes the demo.
The left control on the pages screen activates a horizontal thumbnail viewer on the first click, a second click changes this to a vertical thumbnail viewer. The right control returns to the PDF files screen. The centered green arrows display the ChangeDisplayOrientation() method.

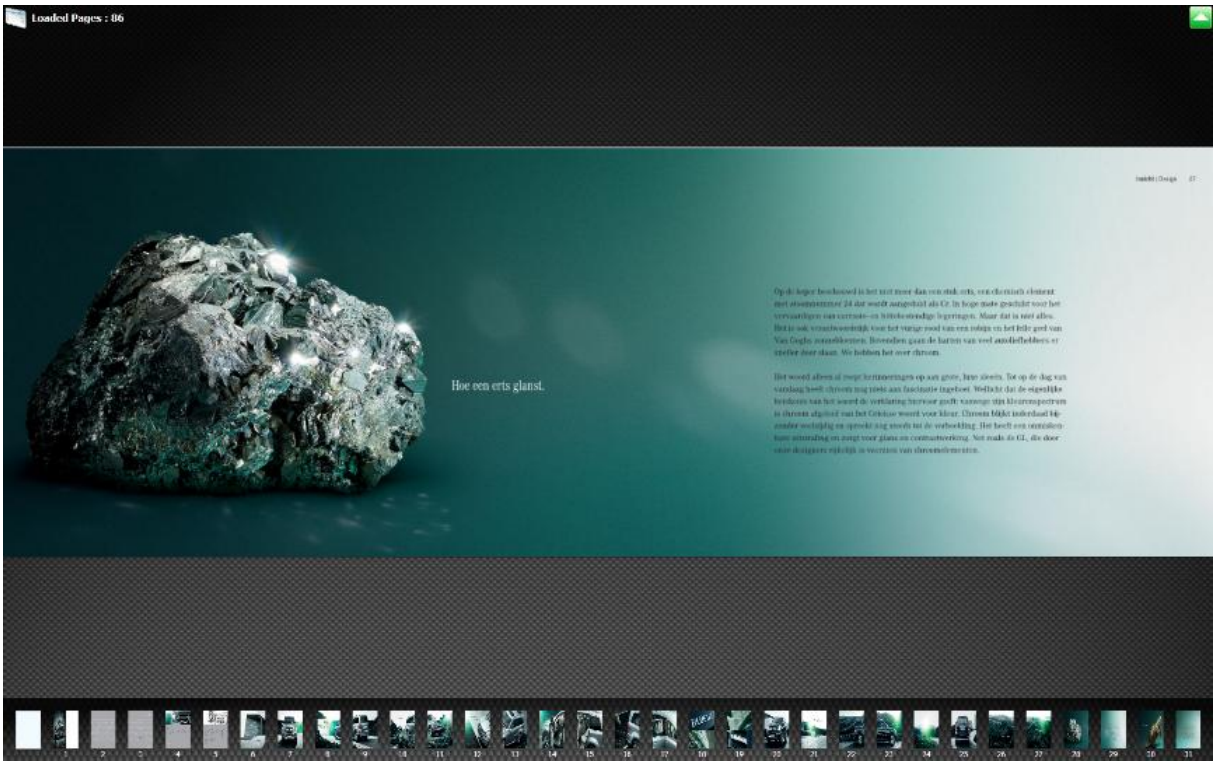
QuickPDF Demo Screenshots

The main screen displays the PDF file covers in random mode.

After the left control is clicked, the main screen displays the PDF file covers in position mode.



After clicking a pdf file, the pages screen shows two pages from that file. Remark the horizontal thumbnail view that was activated after the left control was clicked.



When the left page is clicked, the demo program shows the detail of that page in full view.

TMS Products Viewer Demo description

The TMS Products Viewer gives an overview of the different images from the Images directory.

When the control is clicked, the according webpage is loaded in a web browser.

TMS Products Viewer Demo features

The TMS Products Viewer shows how to use a TAdvSmoothPopup to load a webpage in a web browser.
The TMS Products Viewer gives an example of the use of the TMultiTouchRegionItemControlItem.
When the earth-icon is clicked, the web browser is loaded.

TMS Products Viewer Demo Screenshots

The main screen displays the PDF file covers in random mode

The detail screen displays the webpage loaded in the web browser.


tmssoftware.com
TMS Advanced Charts
TMS FlexCel Studio
TMS IntraWeb Planner
TMS Multitouch SDK
Online Info
HOME
PRODUCTS
BLOG
CUSTOMERS
CONSULTING
SUPPORT
TMS
MultiTouch SDK
Feature overview
Close