Bootstrap-inspired FireMonkey components for responsive, modern UI development

Buttons & Input Controls

Button

TFlexFMXAdvancedButton

Powerful button with Bootstrap styles, icons, HTML text, and badge notifications.

Input

TFlexFMXAdvancedSwitch

Modern toggle switch with ON/OFF states and smooth animations.

Input

TFlexFMXAdvancedRadioButton

Styled radio button with Bootstrap themes.

Input

TFlexFMXAdvancedDateTimeEditor

Date and time picker with modern UI.

Input

TFlexFMXAdvancedSignatureCapture

Capture signatures with touch or mouse input.

Input

TFlexFMXAdvancedRating

Rating / score control with numbered or block segments, cool-to-hot gradient colouring, and thermometer mode.

Input

TFlexFMXBreadCrumbList

Horizontally scrolling row of removable filter tags. Returns a CSV of active filter indices for easy query building.

Layout & Container

Layout

TFlexFMXLayout

Responsive container with Bootstrap-like grid system (12-column layout).

Container

TFlexFMXAdvancedPanel

Panel with gradient backgrounds, shadows, and rounded corners.

Container

TFlexFMXAdvancedRectangle

Styled rectangle with advanced fill and border options.

Layout

TFlexFMXScrollingFlowLayout

Responsive flow layout with Small / Medium / Large column-count breakpoints. Automatically resizes child cards to fill available width.

Layout

TFlexFMXSpacer

Invisible placeholder for padding out grid rows. Visible at design time, completely hidden at run time.

Layout

TFlexFMXRoundRect

Standard TRoundRect enhanced with FlexFMX CSS classes and layout-grid support.

Display & Feedback

Display

TFlexFMXBadge

Small count or label indicator (can float over other controls).

Display

TFlexFMXAdvancedHTMLLabel

Label with HTML formatting support (bold, italic, links, colors).

Display

TFlexFMXAdvancedImage

Image component with scaling, tinting, and effects.

Feedback

TFlexFMXToast

Non-blocking toast notifications for success, error, info messages.

Progress

TFlexFMXProgressBar

Linear progress indicator with styles and animations.

Progress

TFlexFMXProgressCircle

Circular progress indicator with percentage display.

Dialogs & Modals

Dialog

TFlexFMXMessageDialog

Custom message dialog with Bootstrap styling.

Dialog

TFlexFMXInputDialog

Input dialog for collecting user text input.

Dialog

TFlexFMXProgressDialog

Progress dialog for long-running operations.

Dialog

TFlexFMXActionSheet

Action sheet with multiple choice buttons.

Dialog

TFlexFMXPopup

Customizable popup container.

Calendar

Calendar

TFlexFMXAdvancedCalendar

Full-featured calendar with month/week/day views, event display, DataSource binding, recurring events, and Bootstrap colour palettes.

Data & Visualization

Hierarchy

TFlexFMXOrgChart

Organizational chart with zoom, search, expandable nodes, and multiple orientations.

Tree

TFlexFMXAdvancedTreeView

Hierarchical tree view with HTML text, badges, expandable nodes, and connector lines.

Chart

TFlexFMXCharts

Chart component for data visualization (line, bar, pie charts).

Timeline

TFlexFMXTimeline

Timeline component for displaying chronological events.

List

TFlexFMXTileList

Grid/tile layout for displaying collections of items.

Core & Utility Components

Core

TFlexFMXCSS

CSS-like property system for responsive grid layout, spacing, and alignment (Bootstrap-style).

Style

TFlexFMXStyleController

Centralized style controller for managing color schemes across components.

Input

TFlexFMXPassLock

PIN/passcode entry component.

Input

TFlexFMXLetterSelector

Alphabet selector for quick navigation.

Platform & Services

Platform

TFlexFMXPushNotification

Non-visual component for Firebase FCM (Android) and Apple APNs (iOS) push notification registration and receipt.

Getting Started

FlexFMX is a comprehensive component library for FireMonkey that brings Bootstrap-inspired styling and responsive layout capabilities to Delphi applications. The library is designed for:

- Cross-platform development (Windows, macOS, iOS, Android, Linux)
- Responsive, mobile-first interfaces
- Consistent, modern visual design
- Rapid application development

Key Concepts

- **FlexFMXCSS**: All components support Bootstrap-like CSS classes for responsive layouts
- **StyleController**: Centralized theme management for consistent appearance
- **Grid System**: 12-column responsive grid with breakpoints (xs, sm, md, lg, xl)
- **Component Integration**: Components work together seamlessly with shared styling

TFlexFMXLayout

How do I preview different screen sizes at design time?

Set the `DesignBreakpoint` property in the Object Inspector to `xs`, `sm`, `md`, `lg`, or `xl`. The layout will re-render in the Form Designer to show exactly how it will look at that viewport width.

How do I change and view LayoutIndex?

Each child control inside a `TFlexFMXLayout` exposes a `FlexFMXCSS.LayoutIndex` property. This controls the visual order of the child independent of its position in the component tree. To change it at design time, select the child control and set `FlexFMXCSS → LayoutIndex` in the Object Inspector. To change it at runtime:

```
MyButton.FlexFMXCSS.LayoutIndex := 2;
MyLayout.Refresh;
```

How do I create a two-column layout that stacks on mobile?

```
// Parent layout is the row
RowLayout.FlexFMXCSS.CSSClasses := 'row';

// Each child fills full width on mobile, half on medium+
LeftPanel.FlexFMXCSS.CSSClasses  := 'col-12 col-md-6';
RightPanel.FlexFMXCSS.CSSClasses := 'col-12 col-md-6';
```

How do I centre content both horizontally and vertically?

```
ContainerLayout.FlexFMXCSS.CSSClasses := 'row justify-content-center align-items-center vh-100';
```

Best practice: nesting layouts

Always set the outermost `TFlexFMXLayout` to `container` or `container-fluid`, then nest `row` layouts inside, with `col-*` classes on leaf controls. Avoid skipping the row level — going directly from container to col causes alignment issues.

TFlexFMXAdvancedCalendar

How do I connect the calendar to a database?

Drop a `TDataSource` linked to your dataset on the form, then set the calendar's `DataSource` property to it. Next, expand `FieldMapping` in the Object Inspector and map each sub-property to the corresponding field name in your dataset:

```
Calendar1.DataSource := DataSource1;
Calendar1.FieldMapping.EventIDField        := 'ID';
Calendar1.FieldMapping.StartDateTimeField  := 'StartDate';
Calendar1.FieldMapping.EndDateTimeField    := 'EndDate';
Calendar1.FieldMapping.TitleField          := 'Subject';
Calendar1.FieldMapping.ColorField          := 'EventColor'; // optional
```

The calendar refreshes automatically whenever the dataset is refreshed or a record is changed.

How do I add an event programmatically?

```
with Calendar1.Events.Add do
begin
  EventID   := 42;
  DateTime  := Now;
  EndDateTime := Now + (1/24); // 1 hour later
  Title     := 'Team Meeting';
  Color     := TAlphaColors.Steelblue;
end;
Calendar1.Refresh;
```

How do I respond when a user clicks an event?

Handle the `OnEventClick` event. The `AEvent` parameter gives you the full `TFlexFMXCalendarEvent` record:

```
procedure TForm1.Calendar1EventClick(Sender: TObject; AEvent: TFlexFMXCalendarEvent);
begin
  ShowMessage('Clicked: ' + AEvent.Title + ' (ID=' + AEvent.EventID.ToString + ')');
```

```
end;
```

How do I switch between Day, Week, and Month views at runtime?

```
Calendar1.ViewMode := cvmWeek;   // Day, Week, or Month
```

How do I navigate to a specific date?

```
Calendar1.CurrentDate := EncodeDate(2026, 6, 1);
```

Best practice: recurring events

Use a separate recurrence dataset and set `RecurrenceDataSource` with matching `RecurrenceFieldMapping`. Do not duplicate rows in your main events dataset for each occurrence — let the calendar engine expand the recurrence rule. This keeps your database clean and the calendar performant.

TFlexFMXAdvancedButton

How do I add an icon to a button?

```
// Assign an image list at design time via ImageSettings.ImageList, then:
Button1.ImageSettings.ImageIndex := 3;
Button1.ImageSettings.Position   := ipLeft;
Button1.ButtonLayout             := blImageAndText;
```

How do I show a notification badge on a button?

```
Button1.BadgeSettings.Text       := '5';
Button1.BadgeSettings.BadgeColor := TAlphaColors.Red;
```

Set `BadgeSettings.Text` to `''` or `'0'` to hide the badge.

How do I use HTML text in a button label?

```
Button1.UseHTMLText := True;
Button1.HTMLText    := '<b>Save</b> <font size="11">(Ctrl+S)</font>';
```

How do I create a group of toggle buttons where only one can be selected?

Set `GroupName` to the same string on all buttons in the group, and set `IsToggle := True`. Clicking one will automatically deselect the others.

Best practice: Bootstrap style consistency

Use `ButtonStyle` semantically — `bsDanger` for destructive actions, `bsSuccess` for confirmation, `bsSecondary` for cancel. Pair with a `TFlexFMXStyleController` so dark/light mode automatically adjusts all button colours.

TFlexFMXAdvancedTreeView

How do I populate the tree from a dataset?

The tree does not have a built-in DataSource — populate it in code after opening your dataset. A common pattern for self-referencing tables:

```
procedure TForm1.LoadTree;
var
  Node: TFlexFMXTreeNode;
begin
  TreeView1.Nodes.Clear;
  // Load top-level nodes first
  qryItems.Filter := 'ParentID IS NULL';
  qryItems.Filtered := True;
  qryItems.First;
  while not qryItems.Eof do
  begin
    Node := TreeView1.Nodes.Add;
    Node.Text := qryItems.FieldByName('Name').AsString;
    Node.Tag  := qryItems.FieldByName('ID').AsInteger;
    AddChildNodes(Node, Node.Tag);
    qryItems.Next;
```

```
  end;
  TreeView1.Refresh;
end;
```

How do I attach custom data to a node?

```
Node.Data := TMyRecord.Create; // or use Node.Tag for a simple integer ID
```

How do I respond to a node selection?

```
procedure TForm1.TreeView1NodeSelected(Sender: TObject; ANode: TFlexFMXTreeNode);
begin
  // ANode.Tag holds the ID you set when building the tree
  LoadDetailPanel(ANode.Tag);
end;
```

How do I use HTML text in a node?

```
Node.Text := '<b>Project Alpha</b><br><font color="#888">Due: 2026-06-01</font>';
```

How do I show a badge count on a node?

```
Node.BadgeText  := '3';
Node.BadgeColor := TAlphaColors.Red;
```

Best practice: large trees

Set `AutoHeight := False` and place the tree inside a `TScrollBox` for large datasets. Load child nodes lazily in the `OnExpand` event rather than loading the entire tree upfront — this keeps initial render time fast.

TFlexFMXCharts

How do I bind chart data from a database query?

```
Chart1.Labels.Clear;
Chart1.DataSets.Clear;

var DS := Chart1.DataSets.Add;
DS.Name  := 'Monthly Sales';
DS.Color := TAlphaColors.Royalblue;

qrySales.First;
while not qrySales.Eof do
begin
  Chart1.Labels.Add(qrySales.FieldByName('Month').AsString);
  DS.Data := DS.Data + [qrySales.FieldByName('Total').AsFloat];
  qrySales.Next;
end;
Chart1.Refresh;
```

How do I animate the chart on data load?

```
Chart1.Animated         := True;
Chart1.AnimationDuration := 0.4; // seconds
```

How do I switch chart types at runtime?

```
Chart1.ChartType := ctBar;  // ctLine, ctBar, ctPie, ctDoughnut, ctArea
Chart1.Refresh;
```

Best practice: multiple series

When adding multiple `DataSets`, make sure the `Labels` count matches the `Data` array length in every series. Mismatched lengths cause clipped rendering. Always call `Chart1.Refresh` once after all series are added rather than after each one.

TFlexFMXTileList

How do I populate tiles from a database?

```
TileList1.Items.Clear;
qryProducts.First;
while not qryProducts.Eof do
begin
  with TileList1.Items.Add do
  begin
    Title       := qryProducts.FieldByName('Name').AsString;
    Description := qryProducts.FieldByName('Description').AsString;
    FooterText  := CurrToStr(qryProducts.FieldByName('Price').AsCurrency);
    Tag         := qryProducts.FieldByName('ID').AsInteger;
    // Load image from blob or file path
    if not qryProducts.FieldByName('Image').IsNull then
      ImageBitmap.LoadFromStream(TBlobField(qryProducts.FieldByName('Image')).AsStream);
  end;
  qryProducts.Next;
end;
```

How do I respond when a tile is clicked?

```
procedure TForm1.TileList1TileClick(Sender: TObject; AIndex: Integer);
begin
  // Use Tag to identify the record
  LoadProduct(TileList1.Items[AIndex].Tag);
end;
```

How do I make tiles responsive (auto columns)?

```
TileList1.ColumnsCount := 0;   // 0 = auto-calculate from available width
TileList1.TileWidth    := 220;
TileList1.Spacing      := 12;
```

Best practice: images

Use `TileHeight` with a fixed value and keep images the same aspect ratio for a uniform grid. If images vary in size, enable `AutoHeight := True` so taller tiles do not clip their content.

TFlexFMXAdvancedPanel

How do I add a collapsible panel (accordion)?

```
Panel1.Collapsible := True;
Panel1.Collapsed   := True;   // starts collapsed
Panel1.ShowHeader  := True;
Panel1.HeaderText  := 'Advanced Options';
```

How do I change the panel header colour?

```
Panel1.ColorScheme := ccsPrimary;  // Bootstrap colour palette
```

How do I use a panel as a card with a footer?

```
Panel1.ShowHeader  := True;
Panel1.HeaderText  := 'Order Summary';
Panel1.ShowFooter  := True;
Panel1.FooterText  := 'Total: £125.00';
```

TFlexFMXAdvancedDateTimeEditor

How do I connect it to a database field?

```
DateEditor1.DataSource := DataSource1;
DateEditor1.DataField  := 'AppointmentDate';
```

How do I restrict selectable dates?

```
DateEditor1.MinDate := Today;
DateEditor1.MaxDate := IncMonth(Today, 3);
```

How do I get/set the value in code?

```
// Get
var D := DateEditor1.Date;

// Set
DateEditor1.Date := EncodeDate(2026, 12, 25);
```

TFlexFMXAdvancedSignatureCapture

How do I save the signature to a database blob?

```
var Stream := TMemoryStream.Create;
try
  SignatureCapture1.SaveToStream(Stream);
  Stream.Position := 0;
  TBlobField(qryOrders.FieldByName('Signature')).LoadFromStream(Stream);
finally
  Stream.Free;
end;
```

How do I load a saved signature back?

```
var Stream := TMemoryStream.Create;
try
  TBlobField(qryOrders.FieldByName('Signature')).SaveToStream(Stream);
  Stream.Position := 0;
  SignatureCapture1.LoadFromStream(Stream);
finally
  Stream.Free;
end;
```

How do I check if the user has signed?

```
if SignatureCapture1.IsEmpty then
  ShowMessage('Please sign before continuing.');
```

How do I clear the signature?

```
SignatureCapture1.Clear;
```

TFlexFMXToast

How do I show a toast notification?

```
Toast1.Message    := 'Record saved successfully.';
Toast1.ToastStyle := tsSuccess;
Toast1.Show;
```

How do I change the style and auto-hide delay?

```
Toast1.ToastStyle  := tsWarning;   // tsDefault, tsSuccess, tsDanger, tsWarning, tsInfo
Toast1.AutoHide    := True;
Toast1.Delay       := 3000;        // milliseconds
Toast1.CompactMode := True;        // smaller footprint
Toast1.Show;
```

How do I show a toast without placing a component on the form?

Use the class method — no instance needed:

```
TFlexFMXToast.ShowToast(Self, 'Done', 'Record saved.',
  tpTopRight, tsSuccess, True, 3000);
```

How do I control where the toast appears?

Pass a `TToastPosition` to `ShowToast`: `tpTopLeft`, `tpTopCenter`, `tpTopRight`, `tpBottomLeft`, `tpBottomCenter`, `tpBottomRight`.

Best practice

Prefer the `ShowToast` class method for fire-and-forget notifications. Drop a persistent `TFlexFMXToast` instance only when you need to pre-

configure properties at design time.

TFlexFMXAdvancedHTMLLabel

How do I enable HTML rendering?

`HTMLText` is a **Boolean** that enables HTML parsing. The actual HTML content goes in `HTMLSource`. The plain text value is in `Text` (inherited).

```
Label1.HTMLText   := True;   // enable HTML parsing
Label1.HTMLSource := '<b>Status:</b> <font color="#28a745">Active</font> '
                   + '<font size="11">(since 2026-01-01)</font>';
```

How do I make a clickable hyperlink inside the label?

```
Label1.HTMLText   := True;
Label1.HTMLSource := 'Visit our <a href="https://example.com">website</a> for details.';

procedure TForm1.Label1LabelLinkClicked(Sender: TObject; const AURL: string);
begin
  ShellExecute(0, 'open', PChar(AURL), nil, nil, SW_SHOWNORMAL);
end;
```

Wire up the `OnLinkClicked` event in the Object Inspector.

How do I enable word wrap and auto-sizing?

```
Label1.AutoWrap := True;
Label1.AutoSize := True;
```

How do I truncate long text with an ellipsis?

```
Label1.TruncateMode := tmEllipsis;
Label1.MaxLines     := 2;
```

TFlexFMXStyleController

How do I switch between light and dark colour schemes at runtime?

Use the `ColorScheme` property which accepts a `TFlexFMXGridColorScheme` value (e.g. `gcsLight`, `gcsDark`, `gcsPrimary`, `gcsCustom` etc.):

```
StyleController1.ColorScheme := gcsDark;
```

How do I apply custom colours?

Set `ColorScheme := gcsCustom` then modify the sub-property groups:

```
StyleController1.ColorScheme := gcsCustom;
StyleController1.AdvancedPanelStyles.FillColor  := TAlphaColors.Navy;
StyleController1.AdvancedButtonColors.StandardColor := TAlphaColors.Darkslateblue;
```

How do I know when the style changes?

Handle the `OnStyleChanged` event or call `Subscribe` from a component that needs to refresh when the scheme changes:

```
StyleController1.Subscribe(MyPanel.StyleChanged);
```

Best practice: global theming

Place one `TFlexFMXStyleController` on a shared data module and point every FlexFMX control's `StyleController` property to it. Store the user's preference in your settings and restore it in the data module's `OnCreate`.

TFlexFMXAdvancedRating

How do I set and read the rating value?

```
Rating1.Value    := 4;      // set
Rating1.MaxValue := 5;      // default is 5
ShowMessage(Rating1.Value.ToString + ' / ' + Rating1.MaxValue.ToString);
```

How do I make the rating read-only?

```
Rating1.ReadOnly := True;
```

How do I save the rating to a database when it changes?

```
procedure TForm1.Rating1RatingChange(Sender: TObject; AValue: Integer);
begin
  qryReviews.Edit;
  qryReviews.FieldByName('Stars').AsInteger := AValue;
  qryReviews.Post;
end;
```

Wire up the `OnRatingChange` event in the Object Inspector.

How do I change the rating style (stars vs numbers)?

```
Rating1.RatingStyle := rsStars;    // rsNumbers, rsStars, rsCircles, rsSquares
```

TFlexFMXProgressDialog

How do I show progress for a long-running task?

```
ProgressDialog1.Title   := 'Importing data…';
ProgressDialog1.Message := 'Please wait.';
ProgressDialog1.Show;
try
  for I := 1 to RecordCount do
  begin
    ProcessRecord(I);
    ProgressDialog1.Progress := (I / RecordCount) * 100;
    Application.ProcessMessages;
  end;
finally
  ProgressDialog1.Hide;
end;
```

How do I show an indeterminate (spinning) progress indicator?

```
ProgressDialog1.Indeterminate := True;
ProgressDialog1.Show;
```

TFlexFMXPassLock

How do I set up a PIN and prompt the user on app start?

```
// On first run, let the user set a PIN:
PassLock1.SetPIN('1234');

// On subsequent starts, validate:
PassLock1.OnSuccess := HandleUnlock;
PassLock1.Show;
```

How do I respond to a successful unlock?

```
procedure TForm1.HandleUnlock(Sender: TObject);
begin
  MainForm.Show;
end;
```

TFlexFMXOrgChart

How do I populate an org chart from a self-referencing table?

```
OrgChart1.Nodes.Clear;
```

```
// Add root node
var Root := OrgChart1.Nodes.Add;
Root.Title    := 'CEO';
Root.SubTitle := 'Jane Smith';

// Add child
var Mgr := Root.Children.Add;
Mgr.Title    := 'CTO';
Mgr.SubTitle := 'John Doe';

OrgChart1.Refresh;
```

How do I add a photo to an org chart node?

```
Node.ImageBitmap.LoadFromFile('avatar.png');
```

TFlexFMXAdvancedSwitch

How do I toggle the switch on or off in code?

```
Switch1.IsOn := True;
```

How do I change the colour scheme?

Use the `ColorScheme` property (type `TFlexFMXGridColorScheme`):

```
Switch1.ColorScheme := gcsPrimary;   // gcsSuccess, gcsDanger, gcsWarning, etc.
```

For fully custom colours, set `ColorScheme := gcsCustom` and configure them through a linked `TFlexFMXStyleController`.

How do I set custom on/off labels?

```
Switch1.OnText  := 'YES';
Switch1.OffText := 'NO';
```

How do I respond when the user toggles the switch?

```
procedure TForm1.Switch1Switch(Sender: TObject; IsOn: Boolean);
begin
  qrySettings.Edit;
  qrySettings.FieldByName('IsActive').AsBoolean := IsOn;
  qrySettings.Post;
end;
```

Wire up the `OnSwitch` event in the Object Inspector.

TFlexFMXBreadCrumbList

How do I build a navigation breadcrumb trail?

```
BreadCrumb1.Items.Clear;
BreadCrumb1.Items.Add.Text := 'Home';
BreadCrumb1.Items.Add.Text := 'Products';
with BreadCrumb1.Items.Add do
begin
  Text    := 'Widget A';
  IsActive := True;  // marks as current page
end;
```

How do I respond when the user clicks a breadcrumb item?

```
procedure TForm1.BreadCrumb1ItemClick(Sender: TObject; AIndex: Integer);
begin
  NavigateTo(AIndex);
end;
```

TFlexFMXCSS / CSS Classes (General Tips)

How do I apply CSS classes to any FlexFMX control?

Every FlexFMX control exposes a `FlexFMXCSS` sub-property. Set `FlexFMXCSS.CSSClasses` in the Object Inspector or in code:

```
MyButton.FlexFMXCSS.CSSClasses := 'col-6 mt-3 mb-2';
```

What CSS spacing values are available?

Margins and paddings follow Bootstrap naming: `m-0` through `m-5` (all sides), and directional variants `mt-`, `mb-`, `ml-`, `mr-`, `mx-`, `my-`. Same prefix pattern for padding with `p-`.

Best practice: avoid mixing Align and CSS classes

Do not set the FMX `Align` property on controls that are managed by a `TFlexFMXLayout`. The layout engine sets size and position itself; a conflicting `Align` value will cause unpredictable results. Leave `Align` as `alNone`.

← Back to Index

TFlexFMXActionSheet

iOS-style action sheet that slides up from bottom with multiple action buttons.

Key Properties

Title: string
Action sheet title/heading.

Message: string
Optional descriptive message text.

Actions: TActionSheetActions
Collection of action button items.

ShowCancelButton: Boolean
Display a separate Cancel button at bottom.

CancelButtonText: string
Custom text for cancel button (default: "Cancel").

SelectedIndex: Integer
Index of action that was selected (-1 if cancelled).

AnimationDuration: Single
Slide animation duration in seconds.

Action Properties

Caption: string
Action button text.

Style: TActionStyle
asDefault, asDestructive (red), asCancel.

ImageIndex: Integer
Optional icon from image list.

Enabled: Boolean
Whether action can be selected.

Usage Examples

Simple Action Sheet

```
ActionSheet1.Title := 'Choose Action';
ActionSheet1.Actions.Clear;
with ActionSheet1.Actions.Add do
begin
  Caption := 'Edit';
  Style := asDefault;
end;
with ActionSheet1.Actions.Add do
begin
  Caption := 'Delete';
  Style := asDestructive;
end;
ActionSheet1.ShowCancelButton := True;

if ActionSheet1.Execute then
begin
  case ActionSheet1.SelectedIndex of
    0: EditItem;
    1: DeleteItem;
  end;
end;
```

Share Actions

```
ActionSheet1.Title := 'Share';
ActionSheet1.Actions.Clear;
ActionSheet1.Actions.Add.Caption := 'Email';
ActionSheet1.Actions.Add.Caption := 'Message';
ActionSheet1.Actions.Add.Caption := 'Copy Link';
ActionSheet1.ShowCancelButton := True;

if ActionSheet1.Execute then
  ShareVia(ActionSheet1.SelectedIndex);
```

Conditional Actions

```
ActionSheet1.Actions.Clear;
ActionSheet1.Actions.Add.Caption := 'Download';
with ActionSheet1.Actions.Add do
begin
  Caption := 'Delete from Cloud';
  Style := asDestructive;
  Enabled := UserHasPermission;
end;
ActionSheet1.Execute;
```

Methods

- `Execute` - Show action sheet and return True if action selected
- `Show` - Show non-modally
- `Hide` - Dismiss action sheet

Events

- `OnActionClick` - Action button clicked (provides Index)
- `OnCancel` - Cancel button or backdrop clicked
- `OnShow` - Action sheet appears
- `OnHide` - Action sheet dismissed

Notes

- Slides up from bottom on mobile, centered popup on desktop
- Clicking backdrop dismisses (counts as cancel)
- Destructive actions displayed in red
- Cancel button is visually separated at bottom
- Ideal for mobile apps and touch interfaces
- ColorScheme integration for automatic theming

← Back to Index

← Back to Index

TFlexFMXAdvancedButton

A powerful button component with Bootstrap-style variants, icon support, HTML text rendering, badge notifications, and comprehensive styling options.

Overview

TFlexFMXAdvancedButton extends the standard FMX button with:

- Bootstrap-style button variants (Primary, Secondary, Success, Danger, etc.)
- Outline and solid styles
- Icon support with tinting and positioning
- HTML text rendering (bold, italic, strikethrough, font sizes)
- Badge notifications (positioned at top-right)
- Button grouping with selection states
- Hover and click effects
- FlexFMX CSS integration for responsive layouts

Key Properties

ButtonStyle: *TFlexFMXButtonStyle*
Defines the visual style of the button. Options include:

- `bsPrimary` - Blue primary button
- `bsSecondary` - Gray secondary button
- `bsSuccess` - Green success button
- `bsDanger` - Red danger button
- `bsWarning` - Yellow warning button
- `bsInfo` - Cyan info button
- `bsLight` - Light gray button
- `bsDark` - Dark button
- `bsLink` - Link-style button (transparent)
- `bsOutline*` - Outline variants of above styles

ButtonSize: *TFlexFMXButtonSize*
Controls the button size: `bsSmall` (32px), `bsDefault` (40px), `bsLarge` (48px)

Text: *string*
The button's display text.

UseHTMLText: *Boolean*
When True, enables HTML rendering in the button text.

HTMLText: *string*

HTML formatted text when UseHTMLText is True. Supports:
- `<b>Bold</b>`
- `<i>Italic</i>`
- `<s>Strikethrough</s>`
- `<font size="20">Sized text</font>`
- `<br>` for line breaks

ImageSettings: *TFlexFMXImageSettings*
Configuration for button icons:

- `ImageList` - Source image list
- `ImageIndex` - Index of the icon
- `Position` - Icon position (Left, Right, Top, Bottom, Center)
- `UseTint` - Apply color tinting
- `TintColor` - Tint color
- `ScaleMode` - isStretch, isFit, isCrop
- `Size` - Icon size (0 = auto)
- `Spacing` - Space between icon and text
- `Padding*` - Padding around icon

BadgeSettings: *TFlexFMXButtonBadgeSettings*
Badge notification configuration:

- `Text` - Badge text (empty or "0" hides badge)
- `BadgeColor` - Background color (default: Red)
- `Position` - Badge position (default: bpTopRight)
- `OffsetX` - Horizontal offset (default: 8)
- `OffsetY` - Vertical offset (default: -8)

ButtonLayout: *TFlexFMXButtonLayout*
Layout mode: `blImageAndText` , `blImageOnly` , `blTextOnly`

GroupName: *string*
Groups buttons together for radio-button behavior.

StaysDown: *Boolean*
When True and part of a group, button stays selected when clicked.

FlexFMXCSS: *TFlexFMXCSS*
CSS classes for responsive layout integration (Bootstrap-like classes).

Usage Examples

Basic Button

```
Button1.ButtonStyle := bsPrimary;
Button1.Text := 'Click Me';
Button1.ButtonSize := bsDefault;
```

Button with Icon

```
Button1.ImageSettings.ImageList := ImageList1;
Button1.ImageSettings.ImageIndex := 0;
Button1.ImageSettings.Position := ipLeft;
Button1.ImageSettings.UseTint := True;
```

Button with HTML Text

```
Button1.UseHTMLText := True;
Button1.HTMLText := '<b>Bold</b> text with <i>italic</i>';
```

Button with Badge

```
Button1.BadgeSettings.Text := '5';
Button1.BadgeSettings.BadgeColor := TAlphaColors.Red;
```

Button Group

```
Button1.GroupName := 'MyGroup';
Button1.StaysDown := True;
Button2.GroupName := 'MyGroup';
Button2.StaysDown := True;
```

Image-Only Button

```
Button1.ButtonLayout := blImageOnly;
Button1.ImageSettings.ImageList := ImageList1;
Button1.ImageSettings.ImageIndex := 0;
Button1.ImageSettings.Position := ipCenter;
Button1.ImageSettings.ScaleMode := isFit;
```

Events

Event	Description
OnClick	Fired when button is clicked

Notes

- Icons are automatically tinted to match button text color when `UseTint` is enabled
- Badge automatically hides when text is empty or "0"
- Button groups support single-selection behavior similar to radio buttons

- HTML text supports word wrapping with state preservation across lines
- Images maintain aspect ratio and center when using `isFit` scale mode
- Supports both design-time and runtime CSS configuration via FlexFMXCSS

← Back to Index

← Back to Index

TFlexFMXAdvancedCalendar

An Outlook-style calendar component supporting Day, Week and Month views with live data-source binding, recurrence rules, drag-to-create, and Bootstrap colour theming.

Overview

- Three view modes: Day, Week, Month (`TFlexFMXCalendarViewMode`)
- Time-slot granularity: 15, 30 or 60-minute slots
- Events collection with colour coding and HTML text
- Live `TDataSource` binding with flexible field mapping
- Recurrence and exclusion data-source support
- Bootstrap colour palette integration
- FlexFMX CSS / layout grid support
- StyleController integration for light/dark theming

Key Properties

ViewMode: *TFlexFMXCalendarViewMode*
Current calendar view. Values: `cvmDay` , `cvmWeek` , `cvmMonth` .

Granularity: *TFlexFMXCalendarGranularity*
Time-slot height for Day/Week views. Values: `cg15Minutes` , `cg30Minutes` , `cg60Minutes` .

CurrentDate: *TDate*
The date currently shown. Changing this navigates the calendar.

Events: *TFlexFMXCalendarEvents*
Collection of `TFlexFMXCalendarEvent` items. Each item exposes:

- `EventID` — unique integer identifier
- `DateTime` / `EndDateTime` — start and end
- `Title` — displayed on the event card
- `Text` — optional body text
- `Color` — per-event colour override
- `GroupID` / `RecurrenceType` — recurrence grouping

DataSource: *TDataSource*
Bind a live dataset. Use `FieldMapping` to map fields.

FieldMapping: *TFlexFMXCalendarFieldMapping*
Maps dataset field names to calendar properties: `EventIDField` , `StartDateTimeField` , `EndDateTimeField` , `TitleField` , `TextField` , `ColorField` , `GroupIDField` .

RecurrenceDataSource / **ExclusionDataSource**: *TDataSource*
Optional datasets for recurring event rules and exclusion dates. Map fields via `RecurrenceFieldMapping` and `ExclusionFieldMapping`.

ColorPalette: *TFlexFMXCalendarColorPalette*
Bootstrap colour theme for header and navigation buttons: `ccpDefault`, `ccpPrimary`, `ccpSuccess`, `ccpDanger`, etc.

TitleSettings: *TFlexFMXCalendarTitleSettings*
Font properties for event card titles: `FontFamily`, `FontSize`, `FontColor`, `FontStyle`.

Visual Properties

Property	Type	Description
`HeaderHeight`	Single	Height of the top navigation bar
`TimeColumnWidth`	Single	Width of the hour labels column (Day/Week)
`HourHeight`	Single	Pixel height of one hour slot
`BackgroundColor`	TAlphaColor	Calendar background
`GridColor`	TAlphaColor	Grid line colour
`HeaderColor`	TAlphaColor	Header background colour
`TodayHighlightColor`	TAlphaColor	Today cell highlight
`EventDefaultColor`	TAlphaColor	Fallback event colour
`CurrentTimeColor`	TAlphaColor	Colour of the current-time line
`ShowCurrentTime`	Boolean	Show red current-time indicator
`ShowWeekNumbers`	Boolean	Display ISO week numbers

Events

Event	Description
`OnEventClick`	User taps an event card. Receives the `TFlexFMXCalendarEvent`.
`OnEventLongPress`	Long-press on an event, with X/Y coordinates.
`OnDateClick`	User taps an empty day cell. Receives the clicked `TDateTime`.
`OnViewChanged`	Fires when the view mode or navigation date changes.

Quick-start Example

```
// Drop TFlexFMXAdvancedCalendar on a form and set ViewMode in the Object Inspector.
// Add events at run time:
var
  Ev: TFlexFMXCalendarEvent;
begin
  Ev := Calendar1.Events.Add;
  Ev.EventID    := 1;
  Ev.DateTime   := EncodeDateTime(2026, 5, 1, 9, 0, 0, 0);
  Ev.EndDateTime:= EncodeDateTime(2026, 5, 1, 10, 30, 0, 0);
  Ev.Title      := 'Team Stand-up';
  Ev.Color      := TAlphaColorRec.Cornflowerblue;
  Calendar1.Repaint;
end;
```

DataSource Binding

```
// 1. Drop TDataSource pointing to your query/table.
// 2. Assign it to Calendar1.DataSource.
// 3. Map fields:
Calendar1.FieldMapping.StartDateTimeField := 'STARTDATE';
Calendar1.FieldMapping.EndDateTimeField   := 'ENDDATE';
Calendar1.FieldMapping.TitleField         := 'TITLE';
Calendar1.FieldMapping.ColorField         := 'COLOR';    // optional integer ARGB
// The calendar will reload whenever the dataset navigates or refreshes.
```

Tip: Use `FlashEvent(EventID, DateTime)` at run time to make an event card blink briefly — useful for drawing attention to a newly created booking.

Tip: For mobile apps, set `Granularity` to `cg30Minutes` to make touch targets bigger in the Day/Week view.

← Back to Index

TFlexFMXAdvancedDateTimeEditor

Enhanced date/time picker with calendar, time selection, and range validation.

Key Properties

DateTime: TDateTime
Selected date and time value.

EditorType: TDateTimeEditorType
Type: dtDate, dtTime, dtDateTime.

MinDate / MaxDate: TDateTime
Valid date range limits.

Format: string
Display format string (e.g., 'MM/DD/YYYY', 'HH:NN').

ShowCalendar: Boolean
Display inline calendar view.

CalendarStyle: TCalendarStyle
csDropdown (popup) or csInline (always visible).

TimePickerStyle: TTimePickerStyle
tpSpinner (up/down), tpWheel (iOS style), tpClock (analog).

FirstDayOfWeek: TDayOfWeek
Calendar starting day (Sunday or Monday).

ShowWeekNumbers: Boolean
Display week numbers in calendar.

TodayButton: Boolean
Show "Today" quick select button.

Usage Examples

Date Picker

```
DateEditor1.EditorType := dtDate;
DateEditor1.DateTime := Date;
DateEditor1.Format := 'MM/DD/YYYY';
DateEditor1.TodayButton := True;
```

Time Picker

```
DateEditor1.EditorType := dtTime;
DateEditor1.TimePickerStyle := tpWheel;
DateEditor1.Format := 'HH:NN AM/PM';
```

Date Range Validation

```
DateEditor1.EditorType := dtDate;
DateEditor1.MinDate := Date; // No past dates
DateEditor1.MaxDate := Date + 90; // Max 90 days ahead
DateEditor1.DateTime := Date + 7;
```

Inline Calendar

```
DateEditor1.CalendarStyle := csInline;
DateEditor1.ShowWeekNumbers := True;
DateEditor1.FirstDayOfWeek := Monday;
```

Event Scheduling

```
DateEditor1.EditorType := dtDateTime;
DateEditor1.Format := 'MM/DD/YYYY HH:NN';
DateEditor1.DateTime := Now + 1; // Tomorrow, current time

procedure TForm1.DateEditor1Change(Sender: TObject);
begin
  EventDateTime := DateEditor1.DateTime;
  UpdateScheduleDisplay;
end;
```

Methods

- `Clear` - Reset to default/empty value
- `SetToday` - Set to current date
- `SetToNow` - Set to current date and time
- `ShowPicker` - Display calendar/time picker popup
- `Validate` - Check if value within min/max range

Events

- `OnChange` - Date/time value changed
- `OnValidate` - Before accepting new value (return False to reject)
- `OnCalendarShow` - Calendar popup displayed

- `OnCalendarHide` - Calendar popup closed

Notes

- Automatically formats display based on locale
- Mobile-optimized picker wheels for touch devices
- Keyboard navigation supported (arrows, Page Up/Down)
- Invalid dates are rejected with visual feedback
- Supports 12-hour and 24-hour time formats
- Calendar highlights today's date
- ColorScheme integration for automatic theming
- Use for appointments, bookings, deadlines, etc.

← Back to Index

← Back to Index

TFlexFMXAdvancedHTMLLabel

Label control with support for basic HTML formatting tags.

Key Properties

HTMLText: string
Text with HTML formatting tags.

WordWrap: Boolean
Enable automatic word wrapping (preserves HTML formatting).

TextSettings: TTextSettings
Font family, size, color, and style settings.

AutoSize: Boolean
Automatically adjust height to fit content.

VertTextAlign / HorzTextAlign: TTextAlign
Vertical and horizontal text alignment.

Supported HTML Tags

- `<b>...</b>` - **Bold text**
- `<i>...</i>` - *Italic text*
- `<s>...</s>` - ~~Strikethrough text~~
- `<u>...</u>` - Underlined text
- `<a href="url">...</a>` - Clickable hyperlink (triggers OnLinkClicked event)
- `<font size="XX">...</font>` - Custom font size
- `<font color="#RRGGBB">...</font>` - Custom text color
- `<br>` - Line break

Usage Examples

Basic HTML

```
Label1.HTMLText := 'This is <b>bold</b> and this is <i>italic</i>.';
```

Font Sizes

```
Label1.HTMLText := '<font size="24">Large</font> and <font size="12">small</font>.';
```

Multi-line with Formatting

```
Label1.HTMLText := 'Line 1: <b>Important</b><br>' +
                   'Line 2: <i>Additional info</i><br>' +
                   'Line 3: <s>Deprecated</s>';
Label1.WordWrap := True;
```

Mixed Formatting

```
Label1.HTMLText := 'Status: <b><font size="18">ACTIVE</font></b>';
```

Clickable Links

```
Label1.HTMLText := 'Visit <a href="https://www.example.com">our website</a> for more info.';

procedure TForm1.Label1LinkClicked(Sender: TObject; const LinkHref, LinkText: string);
begin
  // LinkHref = 'https://www.example.com'
  // LinkText = 'our website'
  ShellExecute(0, 'open', PChar(LinkHref), nil, nil, SW_SHOWNORMAL);
end;
```

Events

- `OnLinkClicked(Sender: TObject; const LinkHref, LinkText: string)` - Triggered when user clicks a hyperlink. Provides both the href URL and the link display text.

Notes

- HTML parsing preserves formatting across word-wrapped lines
- Nested tags are supported (e.g., `<b><i>text</i></b>`)
- Font size is in pixels
- Font color uses hex format (#RRGGBB or #AARRGGBB)
- Links are automatically underlined and change cursor to hand pointer on hover
- Unknown tags are ignored and rendered as plain text
- Use for dynamic, formatted text displays with interactive links

← Back to Index

← Back to Index

TFlexFMXAdvancedImage

Enhanced image control with rounded corners, shadows, borders, and click effects.

Key Properties

Bitmap / MultiResBitmap: TBitmap / TFixedMultiResBitmap
Image source (use MultiResBitmap for high-DPI support).

ScaleMode: TScaleMode
How image scales: isFit (preserve aspect ratio), isStretch (fill container), isCrop, isCenter.

CornerRadius: Single
Rounded corner radius (0 = square, large value = circle).

Shadow: Boolean
Enable drop shadow effect.

BorderWidth: Single
Border thickness in pixels.

BorderColor: TAlphaColor
Border color.

Clickable: Boolean
Enable click interaction with visual feedback.

HoverEffect: Boolean
Show hover effect on mouse over (desktop).

CSS Classes

- `img-fluid` - Responsive image (fills container)
- `rounded` - Rounded corners
- `rounded-circle` - Circular image (equal width/height)
- `img-thumbnail` - Thumbnail style with border
- `shadow` - Drop shadow

Usage Examples

Profile Picture (Circular)

```
Image1.FlexFMXCSS.Text := 'rounded-circle shadow';
Image1.Width := 100;
Image1.Height := 100;
Image1.ScaleMode := isFit;
```

Photo Thumbnail

```
Image1.FlexFMXCSS.Text := 'img-thumbnail';
Image1.BorderWidth := 2;
Image1.BorderColor := TAlphaColors.Lightgray;
Image1.CornerRadius := 5;
```

Clickable Image

```
Image1.Clickable := True;
Image1.OnClick := ImageClickHandler;
```

High-DPI Image

```
// Add multiple resolutions for crisp display on all devices
Image1.MultiResBitmap.LoadItemFromFile('icon@1x.png', 1.0);
Image1.MultiResBitmap.LoadItemFromFile('icon@2x.png', 2.0);
Image1.MultiResBitmap.LoadItemFromFile('icon@3x.png', 3.0);
```

Events

- `OnClick` - Image clicked (when Clickable = True)
- `OnMouseEnter / OnMouseLeave` - Hover events

Notes

- Use `isFit` scale mode to preserve aspect ratio and prevent distortion
- Images are automatically centered when using isFit mode
- MultiResBitmap automatically selects appropriate resolution for screen density
- Supports transparent PNGs and other common formats
- For best quality on mobile, provide @2x and @3x versions

← Back to Index

← Back to Index

TFlexFMXAdvancedPanel

Enhanced panel with Bootstrap-style borders, shadows, rounded corners, and collapsible content.

Key Properties

BorderStyle: TBorderStyle
Border style: bsNone, bsSolid, bsDashed, bsDotted, etc.

BorderColor: TAlphaColor
Border color (supports transparency).

BorderWidth: Single
Border thickness in pixels.

CornerRadius: Single
Rounded corner radius (0 = square corners).

Shadow: Boolean
Enable drop shadow effect.

ShadowColor: TAlphaColor
Shadow color and opacity.

ShadowBlur: Single
Shadow blur radius (higher = softer shadow).

ShadowOffsetX / ShadowOffsetY: Single
Shadow offset position.

Collapsible: Boolean
Enable collapse/expand functionality.

Collapsed: Boolean
Current collapsed state (runtime).

HeaderHeight: Single
Height of the header area (when collapsible).

HeaderText: string
Text displayed in the header.

AnimationDuration: Single
Collapse/expand animation duration in seconds.

CSS Classes

- `card` - Bootstrap card style
- `card-primary`, `card-secondary`, `card-success` - Colored cards
- `shadow`, `shadow-sm`, `shadow-lg` - Shadow variations
- `rounded`, `rounded-lg` - Rounded corners
- `border`, `border-primary` - Border styles

Usage Examples

Card Panel

```
Panel1.FlexFMXCSS.Text := 'card shadow';
Panel1.CornerRadius := 8;
Panel1.Padding.Rect := RectF(15, 15, 15, 15);
```

Collapsible Panel

```
Panel1.Collapsible := True;
Panel1.HeaderText := 'Click to expand';
Panel1.HeaderHeight := 40;
Panel1.AnimationDuration := 0.3;
```

Shadow Panel

```
Panel1.Shadow := True;
Panel1.ShadowBlur := 10;
Panel1.ShadowOffsetY := 5;
Panel1.ShadowColor := $30000000; // Semi-transparent black
```

Events

- `OnCollapse` - Triggered when panel collapses
- `OnExpand` - Triggered when panel expands
- `OnHeaderClick` - Header area clicked

Notes

- Use with TFlexFMXLayout for responsive card grids
- Supports ColorScheme integration for automatic theming
- Animations use FMX built-in animation system

← Back to Index

← Back to Index

TFlexFMXAdvancedRadioButton

Enhanced radio button with custom styling, colors, and Bootstrap integration.

Key Properties

Text: string
Radio button label text.

IsChecked: Boolean
Whether radio button is selected.

GroupName: string
Radio button group (only one per group can be checked).

CheckColor: TAlphaColor
Color of the checked indicator.

CircleColor: TAlphaColor
Radio button circle outline color.

TextColor: TAlphaColor
Label text color.

ButtonStyle: TRadioStyle
rsDefault (circle), rsButton (button appearance).

Size: Single
Radio button circle size.

CSS Classes

- `form-check` - Standard form radio styling
- `form-check-primary`, `form-check-success` - Colored radios
- `btn-check` - Button-style radio (toggle buttons)

Usage Examples

Radio Button Group

```
// Create radio group
RadioButton1.Text := 'Option 1';
RadioButton1.GroupName := 'MyGroup';
RadioButton1.IsChecked := True;

RadioButton2.Text := 'Option 2';
RadioButton2.GroupName := 'MyGroup';

RadioButton3.Text := 'Option 3';
RadioButton3.GroupName := 'MyGroup';
```

Custom Colors

```
RadioButton1.CheckColor := TAlphaColors.Green;
RadioButton1.CircleColor := TAlphaColors.Darkgray;
RadioButton1.TextColor := TAlphaColors.Black;
```

Button Style

```
RadioButton1.ButtonStyle := rsButton;
RadioButton1.FlexFMXCSS.Text := 'btn-primary';
RadioButton1.Width := 120;
```

Get Selected Value

```
function GetSelectedOption: string;
begin
  if RadioButton1.IsChecked then
    Result := 'Option1'
  else if RadioButton2.IsChecked then
    Result := 'Option2'
  else if RadioButton3.IsChecked then
    Result := 'Option3';
end;
```

Events

- `OnChange` - Radio button checked state changed
- `OnClick` - Radio button clicked

Notes

- Only one radio button per GroupName can be checked
- Setting IsChecked = True automatically unchecks others in group
- Use button style for toolbar or segmented control appearance
- ColorScheme integration for automatic theming
- Supports right-to-left languages

← Back to Index

← Back to Index

TFlexFMXAdvancedRating

A customisable rating / score control that renders numbered or block segments with a gradient colour range from cool (low) to hot (high). Supports horizontal and vertical orientations, read-only mode, and both standard and thermometer display styles.

Overview

- Number blocks or filled bar (thermometer) display styles
- Cool → Mid → Hot gradient colour interpolation
- Horizontal or vertical orientation
- Configurable minimum, maximum and current value
- Read-only mode for display-only scenarios
- Optional sort order (lowest-first or highest-first)
- FlexFMX CSS / layout grid support
- StyleController integration

Key Properties

Value: *Integer*
The current selected rating. Must be between `MinValue` and `MaxValue`.

MinValue / **MaxValue**: *Integer*
Range of the scale. Defaults: `MinValue = 1`, `MaxValue = 5`.

RatingStyle: *TFlexFMXRatingStyle*
`rsNumbers` — individual numbered segments (default).
`rsBlocks` — solid filled blocks without numbers.

Mode: *TFlexFMXRatingMode*
`rmStandard` — highlights segments up to the selected value.
`rmThermometer` — fills a continuous bar proportionally.

Orientation: *TFlexFMXRatingOrientation*
`roHorizontal` (default) or `roVertical`.

SortOrder: *TFlexFMXRatingSortOrder*
`rsoLowestFirst` — low values on the left/top.
`rsoHighestFirst` — high values on the left/top.

Colour Properties

Property	Default	Description

`CoolColor`	Green (#28A745)	Colour for low values
`MidColor`	Amber (#FFC107)	Colour for mid-range values
`HotColor`	Red (#DC3545)	Colour for high values
`TextColor`	Gray	Unselected segment text colour
`SelectedTextColor`	White	Selected segment text colour
`ShowGradient`	True	Interpolate gradient between cool/hot
`UnselectedOpacity`	0.35	Opacity of unselected segments

Behaviour Properties

Property	Default	Description
`ReadOnly`	False	Prevent user interaction
`AllowZero`	False	Allow clicking the current value to deselect it (set to 0)
`ItemSpacing`	4	Gap between segments in pixels

Events

Event	Description
`OnChange`	Fired when `Value` changes. Receives the new integer value.
`OnRatingChange`	Alias for `OnChange` — use whichever you prefer.

Usage Examples

```
// Severity scale 1-10, inverted gradient (hot = low severity = green)
Rating1.MinValue  := 1;
Rating1.MaxValue  := 10;
Rating1.SortOrder := rsoHighestFirst;  // 10 on left, 1 on right
Rating1.CoolColor := TAlphaColorRec.Red;
Rating1.HotColor  := TAlphaColorRec.Lime;
Rating1.Value     := 7;

// Read-only display in a list item
Rating1.ReadOnly := True;
Rating1.Value    := RecordScore;
```

Tip: Set `Mode = rmThermometer` for NPS-style 0-10 scales — the continuous fill makes fractional progress clearer than discrete blocks.

Tip: For vertical orientation inside a fixed-height panel, set `Align = alClient` and let `Orientation = roVertical` fill the available space automatically.

← Back to Index

TFlexFMXAdvancedRectangle

Enhanced rectangle shape with gradients, shadows, borders, and effects.

Key Properties

Fill: TBrush
Fill color, gradient, or pattern.

Stroke: TStrokeBrush
Border stroke properties (color, thickness, style).

CornerRadius: Single
Rounded corner radius (0 = square).

Corners: TCorners
Which corners to round: [crTopLeft, crTopRight, crBottomLeft, crBottomRight].

Shadow: Boolean
Enable drop shadow effect.

ShadowColor: TAlphaColor
Shadow color and opacity.

ShadowBlur: Single
Shadow blur radius.

ShadowOffsetX / ShadowOffsetY: Single
Shadow position offset.

GradientType: TGradientType
Gradient style: gtLinear, gtRadial, gtSweep.

GradientStops: TGradientPoints
Gradient color stops collection.

Usage Examples

Solid Color Rectangle

```
Rect1.Fill.Color := TAlphaColors.Blue;
Rect1.CornerRadius := 10;
Rect1.Stroke.Color := TAlphaColors.Darkblue;
Rect1.Stroke.Thickness := 2;
```

Gradient Background

```
Rect1.Fill.Kind := TBrushKind.Gradient;
Rect1.Fill.Gradient.Color := TAlphaColors.Blue;
Rect1.Fill.Gradient.Color1 := TAlphaColors.Purple;
Rect1.GradientType := gtLinear;
```

Card with Shadow

```
Rect1.Fill.Color := TAlphaColors.White;
Rect1.CornerRadius := 8;
Rect1.Shadow := True;
Rect1.ShadowBlur := 15;
Rect1.ShadowOffsetY := 5;
Rect1.ShadowColor := $30000000; // 19% black
```

Rounded Top Only

```
Rect1.CornerRadius := 15;
Rect1.Corners := [crTopLeft, crTopRight];
Rect1.Fill.Color := TAlphaColors.Lightgray;
```

Badge/Chip Shape

```
Rect1.Height := 30;
Rect1.Width := 80;
Rect1.CornerRadius := 15; // Pill shape
Rect1.Fill.Color := TAlphaColors.Green;
Rect1.Stroke.Kind := TBrushKind.None;
```

Dashed Border

```
Rect1.Fill.Color := TAlphaColors.White;
Rect1.Stroke.Color := TAlphaColors.Gray;
Rect1.Stroke.Thickness := 2;
Rect1.Stroke.Dash := TStrokeDash.Dash;
```

CSS Classes

- `rounded` - Rounded corners
- `rounded-circle` - Circular shape (equal width/height)
- `shadow`, `shadow-sm`, `shadow-lg` - Shadow variations
- `border`, `border-primary` - Border styles
- `bg-primary`, `bg-secondary` - Background colors

Notes

- Use as background shapes, dividers, or decorative elements
- Gradients support multiple color stops for complex effects
- Transparent fills allow click-through to controls below
- Can be used as container (set HitTest = False for children to receive clicks)
- Supports all FMX brush types (solid, gradient, bitmap)
- ColorScheme integration for automatic theming
- Efficient rendering even with many instances

← Back to Index

← Back to Index

TFlexFMXAdvancedSignatureCapture

Touch-enabled signature capture control for collecting digital signatures with smooth drawing.

Key Properties

PenColor: TAlphaColor
Signature ink color.

PenWidth: Single
Signature line thickness.

BackgroundColor: TAlphaColor
Canvas background color.

ShowBaseline: Boolean
Display a horizontal baseline for signature.

BaselineText: string
Text to display below baseline (e.g., "Sign here").

Smoothing: Boolean
Enable stroke smoothing for natural appearance.

SignatureBitmap: TBitmap
Captured signature as bitmap image.

IsEmpty: Boolean (read-only)
True if no signature has been drawn.

MinStrokeLength: Single
Minimum stroke length to count as valid signature.

Usage Examples

Basic Signature Capture

```
SignatureCapture1.PenColor := TAlphaColors.Blue;
SignatureCapture1.PenWidth := 2;
SignatureCapture1.ShowBaseline := True;
SignatureCapture1.BaselineText := 'Sign above this line';
SignatureCapture1.Clear;
```

Save Signature

```
if not SignatureCapture1.IsEmpty then
begin
  SignatureCapture1.SignatureBitmap.SaveToFile('signature.png');
  ShowMessage('Signature saved');
end
else
  ShowMessage('Please sign first');
```

Load Existing Signature

```
var
  Bmp: TBitmap;
begin
  Bmp := TBitmap.Create;
  try
    Bmp.LoadFromFile('signature.png');
    SignatureCapture1.LoadFromBitmap(Bmp);
  finally
    Bmp.Free;
  end;
end;
```

Validation

```
procedure ValidateSignature;
begin
  if SignatureCapture1.IsEmpty then
  begin
    ShowMessage('Signature required');
    Exit;
  end;

  // Convert to base64 for storage
  var Base64 := BitmapToBase64(SignatureCapture1.SignatureBitmap);
  SaveSignatureToDatabase(Base64);
end;
```

Methods

- `Clear` - Erase all signature strokes
- `LoadFromBitmap(Bitmap)` - Load signature from bitmap
- `SaveToFile(FileName)` - Save signature as image
- `GetAsBase64` - Get signature as Base64 string

Events

- `OnDrawStart` - User begins drawing
- `OnDrawing` - User is drawing (mouse/touch move)
- `OnDrawEnd` - User finishes stroke
- `OnClear` - Signature cleared

Notes

- Optimized for touch input on mobile devices
- Smoothing algorithm produces natural-looking signatures
- Works with mouse, touch, and stylus input
- Signature bitmap is transparent by default
- Consider adding Clear and Save buttons for user convenience
- Use validation to ensure signature is not empty before submission

← Back to Index

← Back to Index

TFlexFMXAdvancedSwitch

Modern toggle switch with ON/OFF states, smooth animations, and customizable appearance.

Key Properties

IsOn: Boolean
Gets or sets the switch state (True = ON, False = OFF).

ThumbColor: TAlphaColor
Color of the sliding thumb/handle.

TrackOnColor: TAlphaColor
Background color when switch is ON.

TrackOffColor: TAlphaColor
Background color when switch is OFF.

OnLabel: string
Text displayed on the left when switch is ON.

OffLabel: string
Text displayed on the right when switch is OFF.

Usage Example

```
Switch1.IsOn := True;
Switch1.TrackOnColor := TAlphaColors.Green;
Switch1.TrackOffColor := TAlphaColors.Gray;
Switch1.OnLabel := 'ON';
Switch1.OffLabel := 'OFF';
```

Events

- `OnSwitch` - Fired when switch state changes

← Back to Index

← Back to Index

TFlexFMXAdvancedTreeView

A powerful hierarchical tree view component with HTML text rendering, badge notifications, expandable/collapsible nodes, connector lines, and Bootstrap-style color schemes.

Overview

TFlexFMXAdvancedTreeView provides a feature-rich tree structure display with:

- **HTML Text Rendering** - Each node supports full HTML formatting (bold, italic, font sizes, colors)
- **Badge Notifications** - Display counts or indicators on any node
- **Expandable Hierarchy** - Unlimited nesting depth with expand/collapse functionality
- **Connector Lines** - Visual lines showing parent-child relationships
- **Color Schemes** - Bootstrap-style themes for consistent appearance
- **Row Highlights** - Hover and selection states with customizable colors
- **Auto Height** - Dynamically sizes to fit all expanded nodes
- **Multi-line Nodes** - Nodes can contain multi-line HTML text
- **Data Binding** - Attach custom data objects to nodes

Architecture

The component uses an efficient architecture:

- Inherits from `TRectangle`
- Row backgrounds, expander triangles, and connector lines are drawn on the Canvas during `Paint`
- Node text is rendered using `TFlexFMXAdvancedHTMLLabel` children positioned during `RebuildTree`
- Nodes are arranged vertically with child nodes indented to the right of their parent
- This design avoids component tree issues while providing excellent performance

Key Properties

Node Management

Nodes: *TFlexFMXTreeNodeCollection*
The root collection of tree nodes. Access and manipulate the tree structure through this collection.

SelectedNode: *TFlexFMXTreeNode*
The currently selected node (read-only). Set programmatically or by user click.

Layout Properties

IndentWidth: *Single*
Horizontal spacing for each level of hierarchy (default: 24px)

NodeHeight: *Single*
Fixed height for each node row. Set to 0 for auto-sizing based on HTML content (default: 0)

ExpanderSize: *Single*
Size of the expander triangle indicator (default: 12px)

AutoHeight: *Boolean*
When True, the tree view automatically adjusts its height to fit all visible (expanded) nodes (default: False)

Color Properties

NodeColorScheme: *TFlexFMXTreeNodeColorScheme*
Bootstrap-style color scheme applied to expander triangles and connector lines:

- `tcsDefault` - Default gray theme
- `tcsPrimary` - Blue theme
- `tcsSecondary` - Gray theme
- `tcsSuccess` - Green theme
- `tcsDanger` - Red theme
- `tcsWarning` - Yellow theme
- `tcsInfo` - Cyan theme
- `tcsLight` - Light theme
- `tcsDark` - Dark theme

NodeFillColor: *TAlphaColor*
Default background color for node rows (default: White)

NodeHoverColor: *TAlphaColor*
Background color when hovering over a node (default: Light blue)

SelectedNodeColor: *TAlphaColor*
Background color for the selected node (default: Blue tint)

AlternatingRowColor: *TAlphaColor*
Alternate row background color (set to Null to disable alternating rows)

NodeBorderColor: *TAlphaColor*
Border color between node rows (default: Light gray)

Connector Properties

ShowConnectors: *Boolean*
When True, displays connector lines between parent and child nodes (default: True)

ConnectorColor: *TAlphaColor*
Color of the connector lines (default: Gray)

ConnectorThickness: *Single*
Line thickness for connectors (default: 1.0)

Image Support

ImageList: *TImageList*
Optional image list for node icons (future enhancement)

TFlexFMXTreeNode Properties

Each node in the tree has the following properties:

Text: *string*
HTML-formatted text to display in the node. Supports:

- `<b>Bold</b>`
- `<i>Italic</i>`
- `<font size="20" color="#FF0000">Styled text</font>`
- `<br>` for line breaks

Expanded: *Boolean*
Controls whether the node's children are visible (default: True)

Children: *TFlexFMXTreeNodeCollection*
Collection of child nodes

BadgeSettings: *TFlexFMXTreeNodeBadgeSettings*
Badge configuration for this node:

- `Text` - Badge text (number or label)
- `BadgeColor` - Background color of the badge
- `Position` - Badge placement (bpTopRight, bpTopLeft, bpRightCenter, etc.)
- `OffsetX, OffsetY` - Fine-tune badge position
- `PulsateEnabled` - Enable pulsating animation
- `PulsateCount` - Number of pulses (0 = infinite)

BackgroundColor: *TAlphaColor*
Override background color for this specific node (Null = use default)

Tag: *Integer*
Integer value for storing custom data

Data: *TObject*
Attach any TObject instance to the node

DataObject: *TValue*
Store any Delphi type (object, record, etc.) using TValue

DataString: *string*
String data associated with the node (e.g., ID or key)

Events

OnNodeClick: *TFlexFMXTreeNodeEvent*
Fired when a node is clicked. Use to respond to node selection.

OnNodeExpand: *TFlexFMXTreeNodeExpandEvent*
Fired when a node is expanded or collapsed. The `Expanded` parameter indicates the new state.

Methods

Tree-Level Methods

```
procedure RebuildTree;
```

Immediately rebuilds the tree layout. Called automatically when needed, but can be called manually after programmatic changes.

```
procedure ExpandAll;
```

Expands all nodes in the entire tree.

```
procedure CollapseAll;
```

Collapses all nodes in the tree.

```
procedure ExpandLevel(ALevel: Integer);
```

Expands nodes up to a specific level (0-based). Example: `ExpandLevel(1)` expands root and first children.

```
function GetNodeByTag(ATag: Integer): TFlexFMXTreeNode;
```

Finds a node by its Tag value.

Node-Level Methods

```
function AddChild: TFlexFMXTreeNode;
```

Adds a child node to this node and returns it.

```
class function IsAlive(ANode: TFlexFMXTreeNode): Boolean;
```

Safely checks if a node pointer is still valid (not freed). Safe even with nil or invalid pointers.

Code Examples

Basic Tree Creation

```
 var
  ParentNode, ChildNode: TFlexFMXTreeNode;
begin
  // Add root node
  ParentNode := TreeView.Nodes.Add;
  ParentNode.Text := '<b>Animals</b>';
  ParentNode.Tag := 1;

  // Add child node
  ChildNode := ParentNode.AddChild;
  ChildNode.Text := 'Lion - <i>Panthera leo</i>';
  ChildNode.Tag := 10;

  // Add another child
  ChildNode := ParentNode.AddChild;
  ChildNode.Text := 'Elephant';
  ChildNode.Tag := 11;
end;
```

Adding Badges to Nodes

```
 var
  Node: TFlexFMXTreeNode;
begin
  Node := TreeView.Nodes.Add;
  Node.Text := '<b>Messages</b>';

  // Add a badge showing unread count
  Node.BadgeSettings.Text := '5';
  Node.BadgeSettings.BadgeColor := $FF28A745;  // Success green
  Node.BadgeSettings.Position := bpRightCenter;
end;
```

Multi-line Node Text

```
var
  Node: TFlexFMXTreeNode;
begin
  Node := TreeView.Nodes.Add;
  Node.Text := '<font size="18"><b>Delphi</b></font><br>' +
               '<font size="14" color="#CC0000">Our favourite language!</font><br>' +
               '<font size="12"><i>RAD Studio 12</i></font>';
end;
```

Handling Node Click Events

```
procedure TForm1.TreeViewNodeClick(Sender: TObject; Node: TFlexFMXTreeNode);
begin
  if Assigned(Node) then
  begin
    ShowMessage('Clicked: ' + Node.Text + ', Tag: ' + Node.Tag.ToString);

    // Toggle expansion on click
    Node.Expanded := not Node.Expanded;
  end;
end;
```

Programmatic Node Management

```
// Expand all nodes
TreeView.ExpandAll;

// Collapse all nodes
TreeView.CollapseAll;

// Expand only first two levels
TreeView.ExpandLevel(1);

// Find node by tag
var
  Node: TFlexFMXTreeNode;
begin
  Node := TreeView.GetNodeByTag(42);
  if Assigned(Node) then
    Node.Expanded := True;
end;
```

Custom Node Colors

```
 var
   Node: TFlexFMXTreeNode;
begin
   Node := TreeView.Nodes.Add;
   Node.Text := 'Important Item';
   Node.BackgroundColor := $FFFFE0E0;  // Light red background
end;
```

Best Practices

Performance: The tree view is optimized for large hierarchies. Only visible (expanded) nodes are rendered, making it efficient even with thousands of nodes.

HTML Formatting: Use HTML formatting for rich node text, but avoid overly complex HTML as it may impact rendering performance on very large trees.

Thread Safety: Modify the tree structure only from the main UI thread. The component handles mouse interactions and rebuilds automatically.

AutoHeight: Enable AutoHeight when the tree view is in a scrollable container to automatically size the control to fit all expanded nodes.

CSS Integration

TFlexFMXAdvancedTreeView supports FlexFMX CSS classes for responsive layout within a TFlexFMXLayout container:

```
TreeView.FlexFMXCSS.CSSClasses := 'col-12 col-md-6 m-2 h-400px';
```

This allows the tree view to participate in responsive grid layouts with proper spacing and sizing.

← Back to Index

← Back to Index

TFlexFMXBadge

Small count or label indicator that can float over other controls (like notification badges).

Key Properties

Text: string
Badge text content (numbers, letters, or short text).

BadgeColor: TAlphaColor
Background color of the badge.

TextColor: TAlphaColor
Text color (default: White).

FloatingMode: Boolean
When True, badge floats over its target control.

TargetControl: TControl
Control that the badge should float over (when FloatingMode = True).

BadgePosition: TBadgePosition
Position relative to target: bpTopLeft, bpTopRight, bpBottomLeft, bpBottomRight, bpCenter, etc.

OffsetX / OffsetY: Single
Fine-tune badge position with pixel offsets.

AutoSize: Boolean
Automatically size badge to fit text content.

PulsateEnabled: Boolean
Enable pulsating animation effect for the badge.

PulsateCount: Integer
Number of pulse cycles (0 = infinite loop).

Methods

- `StartPulsate` - Manually start the pulsating animation
- `StopPulsate` - Stop the pulsating animation

Usage Examples

Standalone Badge

```
Badge1.Text := '5';
Badge1.BadgeColor := TAlphaColors.Red;
```

Floating Badge

```
Badge1.FloatingMode := True;
Badge1.TargetControl := Button1;
Badge1.BadgePosition := bpTopRight;
Badge1.Text := '3';
```

Pulsating Badge (Attention Grabber)

```
Badge1.Text := '!';
Badge1.BadgeColor := TAlphaColors.Red;
Badge1.PulsateEnabled := True;
Badge1.PulsateCount := 0; // Infinite loop

// Or start manually:
Badge1.StartPulsate;

// Stop when acknowledged:
Badge1.StopPulsate;
```

Notes

- Use FlexFMXCSS classes like 'badge-primary', 'badge-danger' for Bootstrap styling
- Floating badges automatically stay on top of their target control
- Badge auto-hides when text is empty

← Back to Index

← Back to Index

TFlexFMXBreadCrumbList

A horizontally scrolling row of removable filter tags (breadcrumbs). Each item is rendered as a styled button with a close (×) button. Commonly used to show the active filter set above a list or grid.

Overview

- Items appear as `TFlexFMXAdvancedButton` tags with close buttons
- Horizontally scrollable — works in narrow screens
- Optional alphabetical sorting
- Bootstrap button style applied to all tags
- Each item carries a `FilterIndex` for easy look-up
- Read `FilterString` to get a CSV of all active filter indices
- FlexFMX CSS / layout grid support

Key Properties

Items: *TFlexFMXBreadCrumbItems*
Collection of `TFlexFMXBreadCrumbItem` entries. Each item has:

- `Text` — label shown on the tag button
- `FilterIndex` — integer identifier (e.g., dataset field value or filter ID)

ButtonStyle: *TFlexFMXButtonStyle*
Bootstrap style applied to all tag buttons. Default: `bsPrimary`. Change to `bsSecondary`, `bsInfo`, etc. as needed.

Sorted: *Boolean*
When `True`, tags are displayed alphabetically. Default: `False`.

FilterString: *string* (read-only)
Comma-separated list of all current `FilterIndex` values, e.g. `'1, 3, 7'`. Use this to build a WHERE clause.

Methods

Method	Description
`AddItem(Text, FilterIndex)`	Add a new tag at run time
`RemoveItem(Index)`	Remove a tag by its collection index
`Clear`	Remove all tags

Events

Event	Description
`OnItemAdded`	Fired after a new tag is added
`OnItemRemoved`	Fired after a tag is removed (e.g., user pressed ×)

Usage Example

```
// Build filter tags from a checked list box selection
BreadCrumbList1.Clear;
for I := 0 to CheckListBox.Items.Count - 1 do
  if CheckListBox.Checked[I] then
    BreadCrumbList1.AddItem(CheckListBox.Items[I], I);

// In OnItemRemoved, refresh the data
procedure TForm1.BreadCrumbList1ItemRemoved(Sender: TObject);
begin
  ApplyFilter(BreadCrumbList1.FilterString);
end;
```

Tip: Handle both `OnItemAdded` and `OnItemRemoved` with the same handler that calls your filter routine — then adding or removing a tag instantly refreshes the list.

Tip: Set `Sorted = True` when the tags come from a user-driven multi-select so the row stays visually consistent regardless of selection order.

← Back to Index

TFlexFMXCharts

Responsive chart component supporting line, bar, pie, and doughnut charts with animations.

Key Properties

ChartType: TChartType
Type: ctLine, ctBar, ctPie, ctDoughnut, ctArea.

DataSets: TChartDataSets
Collection of data series to display.

Labels: TStringList
X-axis labels or category names.

ShowLegend: Boolean
Display chart legend.

LegendPosition: TLegendPosition
Legend placement: lpTop, lpBottom, lpLeft, lpRight.

ShowGrid: Boolean
Display background grid lines.

Animated: Boolean
Enable entry animations.

AnimationDuration: Single
Animation duration in seconds.

ShowValues: Boolean
Display data values on chart.

DataSet Properties

Label: string
Data series name (shown in legend).

Data: TArray<Single>
Array of numeric values.

Color: TAlphaColor
Series color.

FillColor: TAlphaColor
Fill color (for area charts).

Usage Examples

Line Chart

```
Chart1.ChartType := ctLine;
Chart1.Labels.Text := 'Jan'#13'Feb'#13'Mar'#13'Apr'#13'May';

with Chart1.DataSets.Add do
begin
  Name := 'Sales';
  Data := [45, 52, 61, 58, 72];
  Color := TAlphaColors.Blue;
end;

Chart1.ShowLegend := True;
Chart1.Animated := True;
```

Bar Chart (Multiple Series)

```
Chart1.ChartType := ctBar;
Chart1.Labels.Text := 'Q1'#13'Q2'#13'Q3'#13'Q4';

with Chart1.DataSets.Add do
begin
  Name := '2023';
  Data := [100, 120, 140, 160];
  Color := TAlphaColors.Blue;
end;

with Chart1.DataSets.Add do
begin
  Name := '2024';
  Data := [110, 135, 150, 180];
  Color := TAlphaColors.Green;
end;
```

Pie Chart

```
Chart1.ChartType := ctPie;
Chart1.Labels.Text := 'Product A'#13'Product B'#13'Product C';

with Chart1.DataSets.Add do
begin
  Name := 'Market Share';
  Data := [45, 30, 25];
  // Colors assigned automatically or set individually
end;

Chart1.ShowValues := True;
```

Real-time Update

```
procedure UpdateChart;
begin
  var NewValue := GetLatestValue;
  Chart1.DataSets[0].Data := Chart1.DataSets[0].Data + [NewValue];
  Chart1.Labels.Add(TimeToStr(Now));
  Chart1.Refresh;
end;
```

Methods

- `Refresh` - Redraw chart with current data
- `Clear` - Remove all data and labels
- `ExportToBitmap` - Get chart as bitmap image
- `AnimateIn` - Replay entry animation

Events

- `OnDataPointClick` - User clicks data point (provides Series, Index, Value)
- `OnLegendClick` - Legend item clicked
- `OnAnimationComplete` - Animation finished

Notes

- Automatically scales to fit available space
- Touch-friendly with zoom and pan support
- Colors auto-generated if not specified
- Supports empty data points (gaps in line charts)
- Use for dashboards, analytics, and data visualization
- ColorScheme integration for automatic theming

← Back to Index

← Back to Index

TFlexFMXCSS

A powerful CSS-like property system for FlexFMX components that provides Bootstrap-style responsive grid layout, spacing, alignment, and sizing controls. This is the core configuration class that enables declarative layout styling similar to web development.

Overview

TFlexFMXCSS is a property object attached to FlexFMX components (like TFlexFMXLayout) that allows you to define responsive layouts using CSS class names or structured property values. It implements a 12-column responsive grid system with breakpoints for different screen sizes.

Two Ways to Configure

1. CSS Classes (String-based)

Define styles using Bootstrap-like CSS class names in the `CSSClasses` property:

```
MyLayout.FlexFMXCSS.CSSClasses := 'col-12 col-md-6 col-lg-4 p-3 mb-2';
```

2. Structured Properties (Object-based)

Configure styles using the `Values` sub-properties through the Object Inspector:

```
MyLayout.FlexFMXCSS.Values.Columns.ColMD := 6;
MyLayout.FlexFMXCSS.Values.Padding.All := 3;
MyLayout.FlexFMXCSS.Values.Margins.Bottom := 2;
```

Key Properties

CSSClasses: *string*
Space-separated CSS class names (e.g., 'col-6 p-3 mb-2'). Supports all Bootstrap-style classes for columns, spacing, alignment, and sizing.

Values: *TFlexFMXCSSValues*
Structured property access for programmatic configuration. Expands into sub-properties:

- **Columns** - Responsive column widths (ColAll, ColXS, ColSM, ColMD, ColLG, ColXL)
- **Margins** - Margin spacing (All, Left, Right, Top, Bottom)
- **Padding** - Internal padding (All, Left, Right, Top, Bottom)
- **HAlign** - Horizontal alignment (haNone, haLeft, haCenter, haRight)
- **VAlign** - Vertical alignment (vaAlignNone, vaAlignTop, vaAlignMiddle, vaAlignBottom)
- **HeightMode** - Height calculation mode (hmDefault, hmPercentage, hmViewportHeight, hmPixels)
- **HeightValue** - Numeric value for height

- **MinWidth/MaxWidth** - Width constraints in pixels

BeginNewRow: *TFlexFMXBreakpoint*
Force this control to start a new row at specified breakpoint (bpNever, bpXS, bpSM, bpMD, bpLG, bpXL, bpXXL, bpAll).

VerticalAlign: *TFlexFMXVerticalAlign*
Vertical alignment within flex container (vaStretch, vaTop, vaCenter, vaBottom).

DesignModeBreakpoint: *string*
Preview breakpoint at design-time ('xs', 'sm', 'md', 'lg', 'xl', 'xxl', or empty for actual size). Allows testing responsive layouts in the IDE without running the app.

Supported CSS Class Names

Column Classes (12-column grid)

- `col` - Auto-width column
- `col-{1-12}` - Fixed column width (e.g., col-6 = 50%, col-4 = 33.33%)
- `col-{breakpoint}-{1-12}` - Responsive column width (e.g., col-md-4, col-lg-3)
- `col-auto` - Size to content

Spacing Classes (Bootstrap scale: 0-5)

Scale values: 0=0px, 1=4px, 2=8px, 3=16px, 4=24px, 5=48px

- `m-{0-5}` - Margin all sides (e.g., m-3 = 16px all sides)
- `mt-{0-5}`, `mb-{0-5}` - Margin top/bottom
- `ml-{0-5}`, `mr-{0-5}` - Margin left/right
- `mx-{0-5}` - Margin horizontal (left + right)
- `my-{0-5}` - Margin vertical (top + bottom)
- `p-{0-5}` - Padding all sides
- `pt-{0-5}`, `pb-{0-5}` - Padding top/bottom
- `pl-{0-5}`, `pr-{0-5}` - Padding left/right
- `px-{0-5}` - Padding horizontal
- `py-{0-5}` - Padding vertical

Container Classes

- `container` - Fixed-width responsive container with max-width per breakpoint
- `container-fluid` - Full-width container (100% width)
- `row` - Flex row container for columns
- `no-gutters` - Remove spacing between columns in a row

Sizing Classes

- `w-{25,50,75,100}` - Width percentage (e.g., w-50 = 50% width)
- `h-{25,50,75,100}` - Height percentage
- `min-w-{px}` - Minimum width in pixels
- `max-w-{px}` - Maximum width in pixels

Alignment Classes

- `justify-content-start`, `center`, `end`, `between`, `around` - Horizontal alignment in flex container
- `align-items-start`, `center`, `end`, `stretch` - Vertical alignment in flex container
- `text-left`, `text-center`, `text-right` - Text alignment

Responsive Breakpoints

Breakpoint	Class Prefix	Width Range	Device Type
Extra Small	`xs`	<576px	Mobile phones (portrait)
Small	`sm`	≥576px	Mobile phones (landscape)
Medium	`md`	≥768px	Tablets
Large	`lg`	≥992px	Desktops
Extra Large	`xl`	≥1200px	Large desktops
Extra Extra Large	`xxl`	≥1400px	Wide monitors

Usage Examples

Basic Responsive Layout

```
// Full width on mobile, half width on tablet+, third width on desktop+
Layout1.FlexFMXCSS.CSSClasses := 'col-12 col-md-6 col-lg-4';
```

Adding Spacing

```
// 3 columns with padding and margin
Layout1.FlexFMXCSS.CSSClasses := 'col-4 p-3 mb-2';
```

Centered Content

```
// Centered content box with max width
Layout1.FlexFMXCSS.CSSClasses := 'container mx-auto text-center';
```

Flex Row with Alignment

```
// Row with centered columns
Layout1.FlexFMXCSS.CSSClasses := 'row justify-content-center align-items-center';
```

Programmatic Configuration

```
with MyLayout.FlexFMXCSS do
begin
  // Set up responsive columns
  Values.Columns.ColAll := 12;  // Default: full width
  Values.Columns.ColMD := 6;    // Tablet: half width
  Values.Columns.ColLG := 4;    // Desktop: third width

  // Add spacing
  Values.Padding.All := 3;      // 16px padding
  Values.Margins.Bottom := 2;   // 8px bottom margin

  // Set alignment
  Values.HAlign := haCenter;
  Values.VAlign := vaAlignMiddle;
end;
```

Complex Grid Layout

```
// Parent container
ParentLayout.FlexFMXCSS.CSSClasses := 'container-fluid';

// Row
RowLayout.FlexFMXCSS.CSSClasses := 'row no-gutters';

// Column 1: Sidebar (fixed width on desktop)
Sidebar.FlexFMXCSS.CSSClasses := 'col-12 col-lg-3 p-3';

// Column 2: Main content (flexible width)
MainContent.FlexFMXCSS.CSSClasses := 'col-12 col-lg-9 p-4';

// Column 3: Right panel (appears only on large screens)
RightPanel.FlexFMXCSS.CSSClasses := 'col-lg-2 p-3';
```

Design-Time Preview

```
// Preview tablet layout at design-time
Layout1.FlexFMXCSS.DesignModeBreakpoint := 'md';

// Preview mobile layout
Layout1.FlexFMXCSS.DesignModeBreakpoint := 'xs';

// Reset to actual form size
Layout1.FlexFMXCSS.DesignModeBreakpoint := '';
```

Methods

- `GetEffectiveCSS` - Returns the combined CSS string from all configurations
- `ParseCSSString(const CSSString: string)` - Parse CSS class string and update structured properties
- `Assign(Source: TPersistent)` - Copy CSS settings from another TFlexFMXCSS instance

Best Practices

CSS Classes vs. Structured Properties:

- **Use CSS Classes** when you prefer compact, declarative styling similar to web development
- **Use Structured Properties** when you need programmatic access or prefer Object Inspector configuration
- Both methods can be mixed - they automatically synchronize

Mobile-First Approach

Define base styles for mobile (no prefix), then override for larger screens:

```
// Mobile: full width, Tablet: half, Desktop: quarter
Layout1.FlexFMXCSS.CSSClasses := 'col-12 col-md-6 col-xl-3';
```

Consistent Spacing

Use the Bootstrap spacing scale consistently throughout your app:

```
// Small spacing: p-1, m-1 (4px)
// Medium spacing: p-3, m-3 (16px)  ← Most common
// Large spacing: p-5, m-5 (48px)
```

Container Rules

- Use `container` for page-like sections with constrained width
- Use `container-fluid` for full-bleed sections
- Always use `row` as parent for `col-*` children
- Column widths should sum to 12 within a row

Common Patterns

Hero Section

```
HeroLayout.FlexFMXCSS.CSSClasses := 'container-fluid p-5 text-center';
```

Card Grid

```
// Container
GridLayout.FlexFMXCSS.CSSClasses := 'container';

// Row
RowLayout.FlexFMXCSS.CSSClasses := 'row';

// Each card
Card1.FlexFMXCSS.CSSClasses := 'col-12 col-sm-6 col-lg-4 p-3';
Card2.FlexFMXCSS.CSSClasses := 'col-12 col-sm-6 col-lg-4 p-3';
Card3.FlexFMXCSS.CSSClasses := 'col-12 col-sm-6 col-lg-4 p-3';
```

Sidebar Layout

```
// Sidebar: hidden on mobile, 25% on desktop
Sidebar.FlexFMXCSS.CSSClasses := 'col-lg-3 p-4';

// Content: full width on mobile, 75% on desktop
Content.FlexFMXCSS.CSSClasses := 'col-12 col-lg-9 p-4';
```

Form Layout

```
// Label column
LabelColumn.FlexFMXCSS.CSSClasses := 'col-12 col-md-4 text-right pr-3';

// Input column
InputColumn.FlexFMXCSS.CSSClasses := 'col-12 col-md-8';
```

Troubleshooting

Layout Not Updating

- Ensure parent is a TFlexFMXLayout component
- Check that DesignModeBreakpoint matches your test size
- Verify column numbers sum to 12 or less in each row

Spacing Not Applied

- Margins/padding only work within TFlexFMXLayout containers
- Check class names for typos (e.g., 'p3' vs 'p-3')
- Ensure parent layout has sufficient size

Design-Time vs Runtime Differences

- Clear DesignModeBreakpoint property for runtime behavior
- Test on actual device sizes, not just designer preview
- Remember that form size at design-time may not match runtime

Related Components

- **TFlexFMXLayout** - Main container that processes FlexFMXCSS properties
- **TFlexFMXAdvancedPanel** - Panel with FlexFMXCSS support
- **TFlexFMXAdvancedButton** - Button with FlexFMXCSS support
- **TFlexFMXTileList** - Responsive tile grid with FlexFMXCSS integration

Notes

- FlexFMXCSS brings web-style responsive design to FireMonkey desktop/mobile apps
- Compatible with standard FMX controls when placed inside TFlexFMXLayout
- CSS classes are parsed at runtime for maximum flexibility
- Design-time preview helps visualize responsive behavior without running app
- Follows Bootstrap 4/5 naming conventions for familiar web developer experience
- All spacing values use a consistent 8px grid system for visual harmony

← Back to Index

← Back to Index

TFlexFMXInputDialog

Modal dialog for collecting text input from users with validation support.

Key Properties

Title: string
Dialog title text.

Prompt: string
Prompt message explaining what input is needed.

DefaultValue: string
Pre-filled default input value.

InputValue: string
User's entered value (after Execute).

InputType: TInputType
Type: itText, itPassword, itNumber, itEmail, itPhone.

MaxLength: Integer
Maximum character limit (0 = unlimited).

Required: Boolean
Prevent OK button if input is empty.

ValidationPattern: string
Regular expression for input validation.

ValidationMessage: string
Error message shown when validation fails.

Multiline: Boolean
Allow multiple lines of text input.

Usage Examples

Simple Text Input

```
Dialog1.Title := 'Enter Name';
Dialog1.Prompt := 'Please enter your name:';
Dialog1.DefaultValue := '';
Dialog1.Required := True;
if Dialog1.Execute = mrOK then
  ShowMessage('Hello, ' + Dialog1.InputValue);
```

Password Input

```
Dialog1.Title := 'Authentication';
Dialog1.Prompt := 'Enter your password:';
Dialog1.InputType := itPassword;
Dialog1.Required := True;
if Dialog1.Execute = mrOK then
  ValidatePassword(Dialog1.InputValue);
```

Email Validation

```
Dialog1.Title := 'Email Address';
Dialog1.Prompt := 'Enter your email:';
Dialog1.InputType := itEmail;
Dialog1.ValidationPattern := '^[\w-\.]+@([\w-]+\.)+[\w-]{2,4}$';
Dialog1.ValidationMessage := 'Invalid email format';
if Dialog1.Execute = mrOK then
  SendEmail(Dialog1.InputValue);
```

Multiline Input

```
Dialog1.Title := 'Comments';
Dialog1.Prompt := 'Enter your feedback:';
Dialog1.Multiline := True;
Dialog1.MaxLength := 500;
if Dialog1.Execute = mrOK then
  SaveFeedback(Dialog1.InputValue);
```

Numeric Input

```
Dialog1.Title := 'Enter Age';
Dialog1.Prompt := 'How old are you?';
Dialog1.InputType := itNumber;
Dialog1.ValidationPattern := '^\d{1,3}$';
Dialog1.ValidationMessage := 'Please enter a valid age';
if Dialog1.Execute = mrOK then
  Age := StrToInt(Dialog1.InputValue);
```

Methods

- `Execute` - Show dialog and return result (mrOK or mrCancel)
- `Validate` - Manually validate current input

Events

- `OnValidate` - Custom validation handler (return True if valid)
- `OnInputChange` - Input text changed
- `OnShow` - Dialog appears
- `OnClose` - Dialog dismissed

Notes

- Input is automatically focused when dialog opens
- Enter key submits, Escape cancels
- Validation occurs before dialog closes
- Password input displays masked characters
- Number input shows numeric keyboard on mobile
- ColorScheme integration for automatic theming

← Back to Index

← Back to Index

TFlexFMXLayout

A responsive container component that implements Bootstrap-like grid system with CSS classes for flexible, mobile-first layouts.

Overview

TFlexFMXLayout provides a powerful layout system inspired by Bootstrap's grid, allowing you to create responsive interfaces that adapt to different screen sizes using CSS-like classes.

Key Features

- Bootstrap-style grid system (12-column layout)
- Responsive breakpoints (xs, sm, md, lg, xl)
- FlexBox-like behavior with rows and columns
- Automatic spacing and alignment
- Design-time preview of different breakpoints
- Runtime responsive behavior

CSS Classes

Container Classes

- `container` - Fixed-width responsive container
- `container-fluid` - Full-width container

Row Classes

- `row` - Creates a flex row
- `no-gutters` - Removes spacing between columns

Column Classes

- `col` - Auto-sized column
- `col-{n}` - Column width (1-12, e.g., col-6 = 50%)
- `col-{breakpoint}-{n}` - Responsive width (e.g., col-md-4)
- `col-auto` - Size to content

Spacing Classes

- `m-{n}` - Margin all sides (0-5)
- `mt-{n}`, `mb-{n}`, `ml-{n}`, `mr-{n}` - Margin sides
- `mx-{n}`, `my-{n}` - Margin horizontal/vertical
- `p-{n}` - Padding all sides
- `pt-{n}`, `pb-{n}`, `pl-{n}`, `pr-{n}` - Padding sides

- `px-{n}`, `py-{n}` - Padding horizontal/vertical

Sizing Classes

- `w-{percent}` - Width (25, 50, 75, 100)
- `h-{percent}` - Height (25, 50, 75, 100)
- `vh-{percent}` - Viewport height (25, 50, 75, 100) - sets height as percentage of available viewport
- `vh-{number}px` - Viewport height in pixels (e.g., vh-300px)
- `min-vh-100` - Minimum viewport height of 100%
- `min-w-{px}` - Minimum width
- `max-w-{px}` - Maximum width

Alignment Classes

- `justify-content-start`, `center`, `end`, `between` - Horizontal alignment
- `align-items-start`, `center`, `end`, `stretch` - Row alignment
- `align-top` - Align control to top of row
- `align-middle` - Vertically center control in row with equal top/bottom spacing
- `align-bottom` - Align control to bottom of row
- `align-baseline` - Align to baseline (same as top)

Usage Examples

Basic Row/Column Layout

```
Layout1.FlexFMXCSS.CSSClasses := 'container';
  Layout2.FlexFMXCSS.CSSClasses := 'row';
    Layout3.FlexFMXCSS.CSSClasses := 'col-6';
    Layout4.FlexFMXCSS.CSSClasses := 'col-6';
```

Responsive Layout

```
// Stacks on mobile, 2 columns on tablet, 3 columns on desktop
Layout1.FlexFMXCSS.CSSClasses := 'col-12 col-md-6 col-lg-4';
```

Centered Content

```
Layout1.FlexFMXCSS.CSSClasses := 'row justify-content-center align-items-center';
```

Vertical Alignment in Row

```
// Button will be vertically centered with equal top/bottom spacing
Button1.FlexFMXCSS.CSSClasses := 'col-4 align-middle';

// Or align to top/bottom
Button2.FlexFMXCSS.CSSClasses := 'col-4 align-top';
Button3.FlexFMXCSS.CSSClasses := 'col-4 align-bottom';
```

Viewport Height

```
// Make a panel take 50% of available viewport height
Panel1.FlexFMXCSS.CSSClasses := 'col-12 vh-50';

// Or use fixed pixel height
Panel2.FlexFMXCSS.CSSClasses := 'col-12 vh-300px';

// Minimum viewport height
ScrollBox1.FlexFMXCSS.CSSClasses := 'col-12 min-vh-100';
```

Breakpoints

- `xs` - Extra small (<576px) - Mobile phones
- `sm` - Small (≥576px) - Landscape phones
- `md` - Medium (≥768px) - Tablets
- `lg` - Large (≥992px) - Desktops
- `xl` - Extra large (≥1200px) - Large desktops

Properties

Grid Configuration

GridColumns *: Integer (default: 12)*

Number of columns in the grid system. Default is 12 for Bootstrap compatibility.

GutterWidth *: Single (default: 30)*

Horizontal spacing between columns in pixels.

SpacingUnit *: Single*

Base unit for spacing calculations (m-*, p-* classes).

RowSpacing *: Single*

Vertical spacing between rows in pixels.

Layout Behavior

EqualRowHeights *: Boolean (default: True)*

When True, all controls in a row have the same height (tallest control).

AutoHeight *: Boolean (default: False)*

Automatically adjust layout height to fit content.

MaxWidth *: Single*

Maximum width for container layouts. Used with 'container' CSS class.

Behaviour *: TBehaviour (default: bhLayout)*

Controls the layout's behavior mode. Options: bhLayout (standard grid layout) or bhGrid (data grid behavior).

SetParentHeight *: Boolean (default: False)*

When True, the layout sets its parent control's height to match its calculated height.

Default Column Widths

DefaultColumnsXS *: Integer (default: 12)*

Default column span for extra small screens (<576px).

DefaultColumnsSM *: Integer (default: 12)*

Default column span for small screens (≥576px).

DefaultColumnsMD *: Integer (default: 6)*

Default column span for medium screens (≥768px).

DefaultColumnsLG *: Integer (default: 4)*

Default column span for large screens (≥992px).

DefaultColumnsXL *: Integer (default: 3)*

Default column span for extra large screens (≥1200px).

Control Sizing

MinimumControlWidth *: Integer (default: 30)*

Minimum width for child controls in pixels.

DefaultControlHeight *: Single (default: 32.0)*

Default height for child controls.

ForceDefaultHeight *: Boolean (default: False)*

When True, all child controls are forced to use DefaultControlHeight.

Design-Time Features

ShowBreakpointIndicator *: Boolean (default: True)*

Show current breakpoint indicator at design-time.

ShowLayoutIndicators *: Boolean*

Display visual layout guides at design-time.

AllowDesignerResize *: Boolean (default: False)*

Allow resizing controls directly in the designer.

CurrentBreakpointInfo *: string (read-only)*

Displays current breakpoint information in the Object Inspector.

Advanced Features

FlexFMXCSS *: TFlexFMXCSS*

CSS class configuration for nested layouts. Allows a layout to be a child of another layout with its own responsive classes.

RestrictToFlexFMX *: Boolean (default: True)*

When True, only FlexFMX-aware controls participate in the grid layout.

AutoSnapToGrid *: Boolean (default: True)*

Automatically snap new controls to the grid layout.

LayoutIndex *: Integer (default: -1)*

Manual control over child control ordering. -1 indicates automatic ordering.

Mobile Support

AdjustForVirtualKeyboard *: Boolean*

Automatically adjust layout when virtual keyboard appears on mobile platforms.

OffsetTop : *Single*

Top offset adjustment for virtual keyboard avoidance.

OffsetBottom : *Single*

Bottom offset adjustment for virtual keyboard avoidance.

Styling

HoverColor : *TAlphaColor*

Background color when mouse hovers over the layout (bhGrid mode).

ClickedColor : *TAlphaColor*

Background color when layout is clicked (bhGrid mode).

EvenRowColour : *TAlphaColor (default: Null)*

Background color for even-numbered rows (bhGrid mode).

OddRowColour : *TAlphaColor (default: Null)*

Background color for odd-numbered rows (bhGrid mode).

StyleController : *TFlexFMXStyleController*

Reference to a style controller for centralized color scheme management.

Events

OnAutoHeight : *TAutoHeightEvent*

Fired when AutoHeight property calculates and applies a new height.

OnStyleChanged : *TNotifyEvent*

Fired when the style or color scheme changes.

Methods

Layout Management

UpdateLayout

Force FlexFMX layout update immediately. Useful for refreshing layout after property changes.

```
MyLayout.UpdateLayout;
```

RefreshLayout

Refresh the layout with current settings.

```
MyLayout.RefreshLayout;
```

RecalculateAutoHeight

Force AutoHeight recalculation immediately. Useful when child controls change size and AutoHeight doesn't seem to be updating.

```
MyLayout.RecalculateAutoHeight;
```

ApplyDefaultCSSClasses(Control: TControl; ForceOverride: Boolean)

Apply default responsive CSS classes to a control. ForceOverride determines if existing classes should be replaced.

```
MyLayout.ApplyDefaultCSSClasses(Button1, False);
```

Control Management

ReorderControlToIndex(Control: TControl; NewIndex: Integer)

Reorder a specific control to a new index position (0-based).

```
MyLayout.ReorderControlToIndex(Button1, 0); // Move to first position
```

UpdateTabOrder

Update tab order of child controls based on visual position (top-to-bottom, left-to-right).

```
MyLayout.UpdateTabOrder;
```

SetChildControlHeight(Control: TControl; RequestedHeight: Single)

Allow child controls to request specific heights from the layout.

```
MyLayout.SetChildControlHeight(Panel1, 150);
```

ClearChildControlHeight(Control: TControl)

Clear any previously set child control height.

```
MyLayout.ClearChildControlHeight(Panel1);
```

Design-Time Helpers

HighlightControl(Control: TControl)

Highlight a specific control with an orange border (for design-time editor integration).

```
MyLayout.HighlightControl(Button1);
```

ClearHighlight

Clear any control highlighting.

```
MyLayout.ClearHighlight;
```

SuspendLayout / ResumeLayout

Temporarily suspend layout updates (useful during batch operations), then resume.

```
MyLayout.SuspendLayout;
try
  // Make multiple changes...
  Button1.Width := 200;
  Button2.Height := 50;
finally
  MyLayout.ResumeLayout;
end;
```

Design-Time Features

FlexFMX Controls Layout Dialog

At design-time, right-click the TFlexFMXLayout component to access the **"FlexFMX Controls Layout..."** context menu option. This opens a visual editor that allows you to:

- View all child controls within the layout
- Reorder controls using **Move Up** and **Move Down** buttons
- Select controls to highlight them with an orange border in the designer
- See the control order changes immediately reflected in the form designer
- Manage complex nested layouts more easily

This design-time editor simplifies the management of control z-order and layout structure.

Notes

- Use the DesignModeBreakpoint property to preview different screen sizes at design-time
- Layouts automatically adjust at runtime based on actual form size
- The 12-column grid provides flexibility for most layout needs
- Combine multiple classes (e.g., 'col-md-6 p-3 mb-2') for complex layouts
- Right-click context menu provides quick access to the layout editor

← Back to Index

← Back to Index

TFlexFMXLetterSelector

Alphabetical index bar for quick navigation in long lists (like iOS Contacts).

Key Properties

Letters: string
Letters to display (default: 'ABCDEFGHIJKLMNOPQRSTUVWXYZ#').

SelectedLetter: Char
Currently selected letter.

TextColor: TAlphaColor
Letter text color.

SelectedColor: TAlphaColor
Selected letter highlight color.

FontSize: Single
Letter font size.

Orientation: TOrientation
Vertical (default) or Horizontal layout.

ShowPopupHint: Boolean
Display large letter popup while dragging.

HapticFeedback: Boolean
Vibrate on letter change (mobile).

Usage Examples

Contacts List Navigation

```
LetterSelector1.Align := alRight;
LetterSelector1.Width := 30;
LetterSelector1.ShowPopupHint := True;
LetterSelector1.HapticFeedback := True;

procedure TForm1.LetterSelector1Change(Sender: TObject);
begin
  ScrollContactsToLetter(LetterSelector1.SelectedLetter);
end;
```

Custom Letters

```
// Include numbers and special character
LetterSelector1.Letters := 'ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789#';

// Only vowels
LetterSelector1.Letters := 'AEIOU';
```

Integration with ListView

```
procedure TForm1.ScrollToLetter(Letter: Char);
var
  I: Integer;
begin
  for I := 0 to ListView1.Items.Count - 1 do
  begin
    if UpperCase(ListView1.Items[I].Text)[1] = Letter then
    begin
      ListView1.ScrollToItem(I);
      Break;
    end;
  end;
end;

procedure TForm1.LetterSelector1Change(Sender: TObject);
begin
  ScrollToLetter(LetterSelector1.SelectedLetter);
end;
```

Horizontal Layout

```
LetterSelector1.Orientation := Horizontal;
LetterSelector1.Align := alTop;
LetterSelector1.Height := 40;
```

Events

- `OnChange` - Selected letter changed
- `OnLetterClick` - Letter clicked (provides Letter: Char)
- `OnDragStart` - User begins dragging
- `OnDragEnd` - User stops dragging

Notes

- Perfect for long alphabetical lists (contacts, dictionaries, etc.)
- Popup hint shows large letter while scrolling (iOS style)
- Haptic feedback provides tactile confirmation on mobile
- Supports touch drag for quick scrubbing
- Automatically disables unavailable letters (optional)
- Use '#' for non-alphabetical entries
- ColorScheme integration for automatic theming

← Back to Index

← Back to Index

TFlexFMXMessageDialog

Modern Bootstrap-style modal dialog for displaying messages with customizable buttons.

Key Properties

Title: string
Dialog title text.

Message: string
Main message content.

MessageType: TMessageType
Type: mtInformation, mtWarning, mtError, mtConfirmation, mtCustom.

Buttons: TDialogButtons
Button configuration: dbOK, dbOKCancel, dbYesNo, dbYesNoCancel, dbCustom.

DefaultButton: TDialogButton
Which button has focus by default.

ShowIcon: Boolean
Display message type icon.

CustomButtons: TStringList
Custom button captions (when Buttons = dbCustom).

ModalResult: TModalResult
Result after dialog closes (mrOK, mrCancel, mrYes, mrNo, etc.).

Usage Examples

Simple Information

```
Dialog1.Title := 'Information';
Dialog1.Message := 'Settings have been saved';
Dialog1.MessageType := mtInformation;
Dialog1.Buttons := dbOK;
Dialog1.Execute;
```

Confirmation

```
Dialog1.Title := 'Confirm Delete';
Dialog1.Message := 'Are you sure you want to delete this item?';
Dialog1.MessageType := mtConfirmation;
Dialog1.Buttons := dbYesNo;
if Dialog1.Execute = mrYes then
  DeleteItem;
```

Error with Details

```
Dialog1.Title := 'Error';
Dialog1.Message := 'Connection failed: ' + E.Message;
Dialog1.MessageType := mtError;
Dialog1.Buttons := dbOK;
Dialog1.Execute;
```

Custom Buttons

```
Dialog1.Title := 'Choose Action';
Dialog1.Message := 'What would you like to do?';
Dialog1.Buttons := dbCustom;
Dialog1.CustomButtons.Clear;
Dialog1.CustomButtons.Add('Save');
Dialog1.CustomButtons.Add('Discard');
Dialog1.CustomButtons.Add('Cancel');
case Dialog1.Execute of
  mrYes: SaveChanges;      // First button
  mrNo: DiscardChanges;     // Second button
  mrCancel: Exit;           // Third button
end;
```

Methods

- `Execute` - Show dialog modally and return result
- `Show` - Show dialog non-modally
- `Hide` - Close dialog programmatically

Events

- `OnButtonClick` - Button clicked (before dialog closes)
- `OnShow` - Dialog appears
- `OnClose` - Dialog dismissed

Notes

- Modal dialogs block interaction with parent form
- Automatically centers on screen or parent control
- Supports Enter (default button) and Escape (cancel) shortcuts
- ColorScheme integration for automatic theming
- Icons automatically match message type

← Back to Index

← Back to Index

TFlexFMXOrgChart

An organizational chart component for visualizing hierarchical structures with expandable/collapsible nodes, HTML text formatting, connector lines, zoom controls, and search functionality.

Overview

TFlexFMXOrgChart displays hierarchical data in a visual tree structure, perfect for organizational charts, family trees, process diagrams, and any hierarchical data visualization. Key features include:

- **HTML Text Rendering** - Rich formatting in node labels (bold, italic, colors, font sizes)
- **Multiple Orientations** - Top-down, bottom-up, left-right, or right-left layouts
- **Expandable Nodes** - Show/hide child nodes with expand/collapse buttons
- **Connector Lines** - Visual connections between parent and child nodes
- **Zoom Controls** - Zoom in/out, zoom to fit, pinch-to-zoom on mobile
- **Search & Highlight** - Find nodes by text and navigate through results
- **Flexible Child Layouts** - Each node can arrange children horizontally or vertically
- **Mouse Interactions** - Click to select, hover highlights, smooth scrolling
- **Data Binding** - Attach custom objects to nodes

Architecture

Like TFlexFMXAdvancedTreeView, the component uses an efficient architecture:

- Inherits from `TRectangle`
- Connectors and buttons are drawn on the Canvas during `Paint`
- Node text is rendered using `TFlexFMXAdvancedHTMLLabel` children
- Positions are calculated based on orientation and node dimensions
- Optimized for large hierarchies with lazy rebuilding

Key Properties

Node Management

> **Nodes**: *TOrgChartNodes*
> The root collection of organizational chart nodes. Add top-level (root) nodes here.

Layout Properties

> **Orientation**: *TOrgChartOrientation*
> Controls the overall chart layout direction:
>
> - `ocoTopDown` - Traditional org chart (root at top, children below)
> - `ocoBottomUp` - Inverted (root at bottom, children above)
> - `ocoLeftRight` - Root on left, children to the right

- `ocoRightLeft` - Root on right, children to the left

Default: `ocoTopDown`

NodeWidth: *Single*
Width of each node box (default: 150px)

NodeHeight: *Single*
Height of each node box (default: 60px)

HorizontalSpacing: *Single*
Horizontal gap between sibling nodes (default: 20px)

VerticalSpacing: *Single*
Vertical gap between parent and child levels (default: 40px)

Connector Properties

ConnectorColor: *TAlphaColor*
Color of the connector lines between nodes (default: Gray)

ConnectorThickness: *Single*
Line thickness for connectors (default: 2.0)

ShowConnectorButtons: *Boolean*
When True, displays expand/collapse buttons on connector lines (default: True)

ConnectorButtonSize: *Single*
Size of the expand/collapse buttons (default: 16px)

Zoom Properties

ZoomScale: *Single*
Current zoom level: 1.0 = 100%, 0.5 = 50%, 2.0 = 200% (default: 1.0)

MinZoom: *Single*
Minimum allowed zoom level (default: 0.1 = 10%)

MaxZoom: *Single*
Maximum allowed zoom level (default: 5.0 = 500%)

Other Properties

ImageList: *TImageList*
Optional image list for node icons

TOrgChartNode Properties

Each node in the organizational chart has the following properties:

HTMLText: *string*
HTML-formatted text for the node label. Supports:

- `<b>Bold</b>`
- `<i>Italic</i>`
- `<font size="18" color="#0000FF">Styled text</font>`
- `<br>` for line breaks

Expanded: *Boolean*
Controls whether the node's children are visible (default: True)

Children: *TCollection*
Collection of child nodes

ChildOrientation: *TChildOrientation*
How this node's children are laid out:

- `coHorizontal` - Children arranged side-by-side (default)
- `coVertical` - Children stacked vertically

Tag: *Integer*
Integer value for storing custom data

NodeType: *Integer*
Custom type identifier for categorizing nodes

Data: *TObject*
Attach any TObject instance to the node for custom data

Events

OnNodeClick: *TNodeClickEvent*
Fired when a node is clicked. Use to respond to node selection.

OnNodeExpand: *TNodeExpandEvent*
Fired when a node is expanded or collapsed. The `Expanded` parameter indicates the new state.

Methods

Chart Management

```
procedure Refresh;
```

Marks the chart for rebuild on next paint (lazy rebuild).

```
procedure RebuildNow;
```

Forces an immediate rebuild of the entire chart layout.

Expand/Collapse

```
procedure ExpandAll;
```

Expands all nodes in the entire chart.

```
procedure CollapseAll;
```

Collapses all nodes in the chart.

Search & Find

```
function FindText(const SearchText: string; CaseSensitive: Boolean = False): Integer;
```

Searches for nodes containing the specified text. Returns the number of matches found. Found nodes are highlighted.

```
procedure ClearFind;
```

Clears the find results and removes highlighting.

```
function GoToNextFound: Boolean;
```

Navigates to the next found node and scrolls it into view. Returns True if successful.

```
function GoToPreviousFound: Boolean;
```

Navigates to the previous found node and scrolls it into view. Returns True if successful.

```
function GetFoundNodes: TList<TOrgChartNode>;
```

Returns the list of found nodes from the last search.

```
function GetFoundCount: Integer;
```

Returns the total number of found nodes.

```
function GetCurrentFoundIndex: Integer;
```

Returns the current found node index (0-based, -1 if none).

Zoom Controls

```
procedure ZoomIn(Percent: Single = 10);
```

Increases zoom by the specified percentage (default: 10%).

```
procedure ZoomOut(Percent: Single = 10);
```

Decreases zoom by the specified percentage (default: 10%).

```
procedure ZoomActual;
```

Resets zoom to 100% (actual size).

```
procedure ZoomToFit;
```

Automatically adjusts zoom to fit the entire chart in the visible area.

Node Lookup

```
function FindNodeByTag(ATag: Integer): TOrgChartNode;
```

Finds and returns a node by its Tag value.

Node Methods

```
function AddChild: TOrgChartNode;
```

Adds a child node to this node and returns it.

Code Examples

Basic Organizational Chart

```
 var
  CEO, VP1, VP2, Manager: TOrgChartNode;
begin
  // Create CEO (root node)
  CEO := OrgChart.Nodes.Add;
  CEO.HTMLText := '<b>John Smith</b><br>CEO';
  CEO.Tag := 1;

  // Add VP reporting to CEO
  VP1 := CEO.AddChild;
  VP1.HTMLText := '<b>Jane Doe</b><br>VP Sales';
  VP1.Tag := 10;

  // Add another VP
  VP2 := CEO.AddChild;
  VP2.HTMLText := '<b>Bob Johnson</b><br>VP Engineering';
  VP2.Tag := 20;

  // Add manager under VP Engineering
  Manager := VP2.AddChild;
  Manager.HTMLText := '<b>Alice Brown</b><br>Engineering Manager';
  Manager.Tag := 21;

  // Refresh the chart
  OrgChart.Refresh;
end;
```

Changing Orientation

```
 // Traditional top-down org chart
OrgChart.Orientation := ocoTopDown;

// Left-to-right flow chart
OrgChart.Orientation := ocoLeftRight;
```

Custom Child Layout

```
 var
  ParentNode, Child: TOrgChartNode;
begin
  ParentNode := OrgChart.Nodes.Add;
  ParentNode.HTMLText := '<b>Department</b>';

  // Arrange children vertically instead of horizontally
  ParentNode.ChildOrientation := coVertical;

  Child := ParentNode.AddChild;
  Child.HTMLText := 'Team 1';

  Child := ParentNode.AddChild;
  Child.HTMLText := 'Team 2';
end;
```

Zoom Controls

```
 // Zoom in by 20%
OrgChart.ZoomIn(20);

// Zoom out
OrgChart.ZoomOut(15);

// Fit entire chart to view
OrgChart.ZoomToFit;

// Reset to 100%
OrgChart.ZoomActual;
```

Search Functionality

```
 var
  FoundCount: Integer;
begin
  // Search for nodes containing "Manager"
  FoundCount := OrgChart.FindText('Manager', False);
  ShowMessage(Format('Found %d matches', [FoundCount]));

  // Navigate through results
  if FoundCount > 0 then
  begin
    // Go to first match (called automatically by FindText)
    // Navigate to next match
    OrgChart.GoToNextFound;

    // Navigate to previous match
    OrgChart.GoToPreviousFound;
  end;

  // Clear search results
  OrgChart.ClearFind;
end;
```

Handling Node Click Events

```
 procedure TForm1.OrgChartNodeClick(Sender: TObject; Node: TOrgChartNode);
begin
  if Assigned(Node) then
  begin
    ShowMessage('Clicked: ' + Node.HTMLText);

    // Use Tag to identify the employee
    LoadEmployeeDetails(Node.Tag);
  end;
end;
```

Expand/Collapse All

```
 // Collapse entire chart
OrgChart.CollapseAll;

// Expand entire chart
OrgChart.ExpandAll;
```

Finding Nodes by Tag

```
 var
  Node: TOrgChartNode;
begin
  Node := OrgChart.FindNodeByTag(42);
  if Assigned(Node) then
  begin
    // Expand this node
    Node.Expanded := True;
    OrgChart.Refresh;
  end;
end;
```

Customizing Appearance

```
 // Change node dimensions
OrgChart.NodeWidth := 200;
OrgChart.NodeHeight := 80;

// Adjust spacing
OrgChart.HorizontalSpacing := 30;
OrgChart.VerticalSpacing := 50;

// Customize connector lines
OrgChart.ConnectorColor := TAlphaColors.Blue;
OrgChart.ConnectorThickness := 2.5;

// Hide expand/collapse buttons
OrgChart.ShowConnectorButtons := False;
```

Best Practices

ScrollBox Container: Place the OrgChart inside a TVertScrollBox for best results. This enables automatic scrolling and smooth pan/zoom interactions.

Lazy Rebuilding: The chart uses lazy rebuilding for performance. Call `Refresh` to mark for rebuild, or `RebuildNow` for immediate updates after multiple changes.

Mobile Support: The component supports pinch-to-zoom gestures on mobile devices. Enable Touch.InteractiveGestures on the parent form.

Large Hierarchies: For very large charts (100+ nodes), consider using `CollapseAll` initially and letting users expand sections as needed.

HTML Formatting: Keep HTML text concise for best visual appearance. Use line breaks (
) to create multi-line labels.

Use Cases

- **Corporate Org Charts** - Visualize company hierarchy and reporting structures
- **Family Trees** - Display genealogical relationships
- **Project Dependencies** - Show task breakdown and dependencies
- **Process Flows** - Illustrate business or technical processes
- **Classification Trees** - Display taxonomies and categorization
- **Decision Trees** - Visualize decision-making logic

CSS Integration

TFlexFMXOrgChart supports FlexFMX CSS classes for responsive layout within a TFlexFMXLayout container:

```
OrgChart.FlexFMXCSS.CSSClasses := 'col-12 m-2 h-600px';
```

This allows the org chart to participate in responsive grid layouts with proper spacing and sizing.

← Back to Index

← Back to Index

TFlexFMXPassLock

PIN/passcode entry control with customizable length and visual feedback, ideal for app security.

Key Properties

PasscodeLength: Integer
Number of digits required (default: 4).

Value: string
Entered passcode value.

MaskCharacter: Char
Character displayed for each digit (default: '●').

ShowValue: Boolean
Display actual digits instead of mask characters.

ActiveColor: TAlphaColor
Color of filled/active digit circles.

InactiveColor: TAlphaColor
Color of empty digit circles.

ErrorColor: TAlphaColor
Color shown on validation error.

ShowKeyboard: Boolean
Display built-in numeric keyboard (mobile-friendly).

AllowBiometric: Boolean
Enable Face ID / Touch ID / Fingerprint option.

AutoSubmit: Boolean
Automatically validate when all digits entered.

Usage Examples

Basic PIN Entry

```
PassLock1.PasscodeLength := 4;
PassLock1.ShowKeyboard := True;
PassLock1.AutoSubmit := True;
PassLock1.ActiveColor := TAlphaColors.Blue;
```

6-Digit Code

```
PassLock1.PasscodeLength := 6;
PassLock1.MaskCharacter := '*';
PassLock1.OnComplete := PasscodeEnteredHandler;
```

Validation

```
procedure TForm1.PassLock1Complete(Sender: TObject);
begin
  if PassLock1.Value = '1234' then
  begin
    ShowMessage('Access granted');
    UnlockApp;
  end
  else
  begin
    PassLock1.ShowError;
    PassLock1.Clear;
  end;
end;
```

Biometric Option

```
PassLock1.AllowBiometric := True;
PassLock1.OnBiometricAuth := BiometricAuthHandler;

procedure TForm1.BiometricAuthHandler(Sender: TObject; Success: Boolean);
begin
  if Success then
    UnlockApp
  else
    ShowMessage('Authentication failed');
end;
```

Methods

- `Clear` - Reset all entered digits
- `ShowError` - Display error animation/color
- `Validate(ExpectedCode)` - Check if code matches
- `RequestBiometric` - Trigger biometric authentication

Events

- `OnComplete` - All digits entered
- `OnChange` - Digit added or removed
- `OnError` - Validation failed

- `OnBiometricAuth` - Biometric result (Success: Boolean)

Notes

- Built-in keyboard optimized for touch devices
- Shake animation on error (like iOS)
- Supports hardware keyboard input on desktop
- Biometric integration requires platform permissions
- Use for app locks, transaction confirmations, parental controls
- ColorScheme integration for automatic theming
- Consider adding "Forgot PIN?" option for user recovery

← Back to Index

← Back to Index

TFlexFMXPopup

Flexible popup/overlay component for menus, tooltips, popovers, and modal content.

Key Properties

PopupType: TPopupType
Type: ptDropdown, ptModal, ptTooltip, ptPopover.

PlacementTarget: TControl
Control to position popup relative to.

Placement: TPlacement
Position: pTop, pBottom, pLeft, pRight, pCenter.

ShowBackdrop: Boolean
Display semi-transparent backdrop behind popup.

BackdropColor: TAlphaColor
Backdrop overlay color.

CloseOnBackdropClick: Boolean
Dismiss popup when clicking backdrop.

ShowArrow: Boolean
Display arrow/pointer to target control.

AnimationType: TPopupAnimation
Entry/exit animation: paSlide, paFade, paZoom, paNone.

AnimationDuration: Single
Animation duration in seconds.

AutoSize: Boolean
Automatically size popup to fit content.

Usage Examples

Dropdown Menu

```
Popup1.PopupType := ptDropdown;
Popup1.PlacementTarget := Button1;
Popup1.Placement := pBottom;
Popup1.ShowArrow := True;
Popup1.CloseOnBackdropClick := True;

// Add menu items to popup
var MenuLayout := TFlexFMXLayout.Create(Popup1);
MenuLayout.Parent := Popup1;
// Add menu buttons...

Popup1.Popup; // Show popup
```

Modal Dialog

```
Popup1.PopupType := ptModal;
Popup1.Placement := pCenter;
Popup1.ShowBackdrop := True;
Popup1.BackdropColor := $80000000; // 50% black
Popup1.CloseOnBackdropClick := True;
Popup1.AnimationType := paZoom;

Popup1.Width := 400;
Popup1.Height := 300;
Popup1.Popup;
```

Tooltip

```
Popup1.PopupType := ptTooltip;
Popup1.PlacementTarget := EditControl;
Popup1.Placement := pTop;
Popup1.ShowArrow := True;
Popup1.AnimationType := paFade;

var Label := TLabel.Create(Popup1);
Label.Parent := Popup1;
Label.Text := 'Enter your email address';
Label.TextSettings.Font.Size := 12;

Popup1.AutoSize := True;
Popup1.Popup;
```

Context Menu

```
procedure TForm1.Image1MouseDown(Sender: TObject; Button: TMouseButton;
  Shift: TShiftState; X, Y: Single);
begin
  if Button = TMouseButton.mbRight then
  begin
    Popup1.PopupType := ptDropdown;
    Popup1.PlacementTarget := Image1;
    Popup1.Placement := pBottom;
    // Position at cursor
    Popup1.Left := X;
    Popup1.Top := Y;
    Popup1.Popup;
  end;
end;
```

Methods

- `Popup` - Show popup
- `IsOpen` - Check if popup is currently visible
- `Close` - Hide popup
- `PopupAtPoint(X, Y)` - Show at specific coordinates

Events

- `OnPopup` - Before popup appears
- `OnClose` - Popup dismissed
- `OnBackdropClick` - Backdrop area clicked

Design-Time Features

Execute Popup

At design-time, right-click the TFlexFMXPopup component to access the **"Execute Popup"** context menu option. This allows you to:

- Preview how the popup will appear at runtime
- Test popup positioning and placement
- Verify popup content layout
- Check animations and transitions

This helps you visualize and adjust the popup appearance without running the application.

Notes

- Popups float above all other controls
- Modal popups prevent interaction with parent form
- Automatically repositions if off-screen
- Arrow automatically points to target control
- Use for menus, tooltips, confirmations, custom dialogs
- ColorScheme integration for automatic theming

- Consider accessibility - provide keyboard shortcuts to dismiss
- Right-click "Execute Popup" at design-time to preview popup appearance

← Back to Index

← Back to Index

TFlexFMXProgressBar

Bootstrap-style horizontal progress bar with multiple styles and animation support.

Key Properties

Value: Single
Current progress value (0 to Max).

Min / Max: Single
Minimum and maximum range values (default: 0-100).

ShowText: Boolean
Display percentage text inside bar.

BarColor: TAlphaColor
Progress bar fill color.

BackgroundColor: TAlphaColor
Background track color.

Striped: Boolean
Show diagonal stripe pattern.

Animated: Boolean
Animate stripes (requires Striped = True).

BarHeight: Single
Height of the progress bar.

CornerRadius: Single
Rounded corners radius.

CSS Classes

- `progress-bar` - Base progress bar
- `progress-bar-success` - Green success bar
- `progress-bar-info` - Blue info bar
- `progress-bar-warning` - Yellow warning bar
- `progress-bar-danger` - Red danger bar
- `progress-bar-striped` - Striped pattern
- `progress-bar-animated` - Animated stripes

Usage Examples

Basic Progress

```
ProgressBar1.Max := 100;
ProgressBar1.Value := 45;
ProgressBar1.ShowText := True;
```

Success Bar

```
ProgressBar1.FlexFMXCSS.Text := 'progress-bar-success';
ProgressBar1.Value := 100;
```

Animated Striped

```
ProgressBar1.FlexFMXCSS.Text := 'progress-bar-striped progress-bar-animated';
ProgressBar1.Striped := True;
ProgressBar1.Animated := True;
```

Download Progress

```
procedure UpdateDownloadProgress(BytesReceived, TotalBytes: Int64);
begin
  ProgressBar1.Max := TotalBytes;
  ProgressBar1.Value := BytesReceived;
end;
```

Notes

- Percentage is automatically calculated from Value/Max ratio
- Use animated stripes for indeterminate progress
- Supports smooth value transitions via AnimateValue method
- ColorScheme integration for automatic theming

← Back to Index

← Back to Index

TFlexFMXProgressCircle

Circular progress indicator with percentage display and customizable appearance.

Key Properties

Value: Single
Current progress value (0 to Max).

Min / Max: Single
Minimum and maximum range values (default: 0-100).

ShowText: Boolean
Display percentage text in center.

CircleColor: TAlphaColor
Progress circle fill color.

BackgroundColor: TAlphaColor
Background circle color.

CircleWidth: Single
Thickness of the progress circle stroke.

StartAngle: Single
Starting angle in degrees (default: -90, starts at top).

ClockWise: Boolean
Progress direction (clockwise or counter-clockwise).

Animated: Boolean
Animate value changes.

AnimationDuration: Single
Animation duration in seconds.

Usage Examples

Basic Circle

```
ProgressCircle1.Max := 100;
ProgressCircle1.Value := 75;
ProgressCircle1.ShowText := True;
```

Custom Style

```
ProgressCircle1.CircleColor := TAlphaColors.Green;
ProgressCircle1.CircleWidth := 12;
ProgressCircle1.BackgroundColor := $20000000; // Semi-transparent
```

Animated Progress

```
ProgressCircle1.Animated := True;
ProgressCircle1.AnimationDuration := 0.5;
ProgressCircle1.Value := 90; // Smoothly animates to 90%
```

Loading Indicator

```
// For indeterminate loading, continuously update value
procedure TForm1.Timer1Timer(Sender: TObject);
begin
  ProgressCircle1.Value := (ProgressCircle1.Value + 5) mod 100;
end;
```

Notes

- Percentage text automatically scales with circle size
- Use transparent background for overlay effects
- Ideal for dashboards and KPI displays
- ColorScheme integration for automatic theming
- Can be used as a countdown timer (reverse progress)

← Back to Index

← Back to Index

TFlexFMXProgressDialog

Modal progress dialog for long-running operations with cancellation support.

Key Properties

Title: string
Dialog title text.

Message: string
Progress status message.

Progress: Integer
Current progress value (0-100).

Indeterminate: Boolean
Show spinning indicator instead of progress bar.

ShowPercentage: Boolean
Display percentage text.

Cancelable: Boolean
Allow user to cancel operation.

Cancelled: Boolean (read-only)
True if user clicked Cancel button.

CancelButtonText: string
Custom text for cancel button (default: "Cancel").

ShowElapsedTime: Boolean
Display elapsed time counter.

Usage Examples

File Processing

```
ProgressDialog1.Title := 'Processing Files';
ProgressDialog1.Message := 'Please wait...';
ProgressDialog1.Cancelable := True;
ProgressDialog1.ShowPercentage := True;
ProgressDialog1.Show;

for I := 1 to FileCount do
begin
  if ProgressDialog1.Cancelled then Break;

  ProcessFile(Files[I]);
  ProgressDialog1.Progress := Round((I / FileCount) * 100);
  ProgressDialog1.Message := Format('Processing file %d of %d', [I, FileCount]);
  Application.ProcessMessages;
end;

ProgressDialog1.Hide;
```

Indeterminate Progress

```
ProgressDialog1.Title := 'Connecting...';
ProgressDialog1.Message := 'Establishing connection to server';
ProgressDialog1.Indeterminate := True;
ProgressDialog1.Cancelable := True;
ProgressDialog1.Show;

try
  ConnectToServer;
finally
  ProgressDialog1.Hide;
end;
```

Download Progress

```
ProgressDialog1.Title := 'Downloading';
ProgressDialog1.ShowElapsedTime := True;
ProgressDialog1.ShowPercentage := True;
ProgressDialog1.Cancelable := True;
ProgressDialog1.Show;

procedure OnDownloadProgress(Sender: TObject; BytesReceived, TotalBytes: Int64);
begin
  if ProgressDialog1.Cancelled then
    AbortDownload;

  ProgressDialog1.Progress := Round((BytesReceived / TotalBytes) * 100);
  ProgressDialog1.Message := Format('%.1f MB / %.1f MB',
    [BytesReceived / 1048576, TotalBytes / 1048576]);
end;
```

Async Operation

```
ProgressDialog1.Title := 'Loading Data';
ProgressDialog1.Indeterminate := True;
ProgressDialog1.Show;

TTask.Run(procedure
begin
  var Data := LoadData; // Long operation

  TThread.Synchronize(nil, procedure
  begin
    ProgressDialog1.Hide;
    DisplayData(Data);
  end);
end);
```

Methods

- `Show` - Display progress dialog
- `Hide` - Close dialog
- `UpdateProgress(Value, Message)` - Update progress and message
- `Reset` - Reset progress to 0

Events

- `OnCancel` - User clicked Cancel button
- `OnShow` - Dialog displayed
- `OnHide` - Dialog closed

Notes

- Blocks user interaction with parent form (modal)
- Check Cancelled property frequently in loops
- Call Application.ProcessMessages to keep UI responsive
- Use indeterminate mode when progress cannot be measured
- Elapsed time automatically starts when shown
- ColorScheme integration for automatic theming
- Consider using threads for truly non-blocking operations

← Back to Index

← Back to Index

TFlexFMXPushNotification

A non-visual component that handles device registration and notification receipt for Firebase Cloud Messaging (Android / FCM) and Apple Push Notification Service (iOS / APNs). Drop it on a form, configure platform credentials, and handle the events.

Overview

- Supports Firebase FCM (Android) and Apple APNs (iOS)
- Auto-detects platform or override manually via `Platform`
- Optional automatic registration on component creation (`AutoRegister`)
- Provides the device token for server-side sending
- Rich notification data via `TFlexFMXPushData` including custom key/value pairs
- `IsSupported` class function for safe feature detection at run time

Published Properties

Enabled: *Boolean*
Activates the push notification service. Default: `False` . Set to `True` before calling `Register` .

Platform: *TFlexFMXPushPlatform*
`ppAutoDetect` (default) — chooses Firebase or APNs based on the target OS.
`ppFirebase` — force Firebase even on iOS (e.g., for testing).
`ppAPNS` — force APNs.

AutoRegister: *Boolean*
When `True` the component calls `Register` automatically during construction. Default: `False` .

GCMAppID: *string*
Your Firebase project's Sender ID / App ID. Required for Android push notifications.

APNSAppID: *string*
Your APNs Bundle ID. Required for iOS push notifications.

Read-Only Properties

DeviceToken: *string*
The FCM registration token or APNs device token received after a successful registration. Pass this to your backend to target this device.

DeviceID: *string*
Platform device identifier, useful for analytics or server-side device management.

Methods

Method	Description
`Register`	Request a push token from FCM / APNs. Result delivered via `OnTokenReceived` or `OnRegistrationError`.
`Unregister`	De-register this device from push notifications.
`IsSupported` (class function)	Returns `True` if push notifications are available on the current platform.

Events

Event	Signature	Description
`OnTokenReceived`	`procedure(Sender: TObject; const Token: string)`	Fired when a new device token is issued. Send this token to your server.
`OnRegistrationSuccess`	`TNotifyEvent`	Fired when registration completes without error.
`OnRegistrationError`	`procedure(Sender: TObject; const ErrorMessage: string)`	Fired when registration fails.
`OnNotificationReceived`	`procedure(Sender: TObject; NotificationData: TFlexFMXPushData)`	Fired when a push notification arrives while the app is in the foreground.

TFlexFMXPushData

The object passed to `OnNotificationReceived`:

Field	Type	Description
`Title`	string	Notification title
`Message`	string	Notification body text
`BadgeNumber`	Integer	Badge count (iOS)
`Sound`	string	Sound filename (or 'default')
`RawData`	string	Raw JSON payload string
`CustomData`	`TDictionary<string, string>`	Arbitrary key/value pairs from the notification payload

Quick-Start Example

```
// 1. Drop TFlexFMXPushNotification on form
// 2. Set GCMAppID = 'your-firebase-sender-id'
// 3. Set Enabled = True

procedure TForm1.FormCreate(Sender: TObject);
begin
  if TFlexFMXPushNotification.IsSupported then
    PushNotification1.Register;
end;

procedure TForm1.PushNotification1TokenReceived(Sender: TObject;
  const Token: string);
begin
  // Send Token to your back-end
  MyServer.RegisterDevice(Token);
end;

procedure TForm1.PushNotification1NotificationReceived(Sender: TObject;
  NotificationData: TFlexFMXPushData);
begin
  ShowMessage(NotificationData.Title + #13#10 + NotificationData.Message);
  // Access custom payload: NotificationData.CustomData['order_id']
end;
```

Tip: Always check `TFlexFMXPushNotification.IsSupported` before registering — it returns `False` on Windows and other platforms where push is unavailable, preventing runtime errors during desktop testing.

Tip: On iOS, APNs registration requires the user to grant permission. The OS permission dialog is shown automatically on the first call to `Register`. Handle `OnRegistrationError` to detect when the user denies permission.

← Back to Index

TFlexFMXRoundRect

A FlexFMX-enabled wrapper around the standard FMX `TRoundRect` that adds CSS class support and layout-grid integration. Use it wherever you need a styled rounded rectangle that participates in a `TFlexFMXLayout`.

Overview

- All standard `TRoundRect` properties: `Fill`, `Stroke`, `XRadius`, `YRadius`, `Corners`, `CornerType`, `Sides`
- Adds `FlexFMXCSS` for Bootstrap-style column sizing and spacing
- Adds `LayoutIndex` for explicit control ordering in a `TFlexFMXLayout`
- Cross-platform (all FMX targets)

Key Properties

FlexFMXCSS: *TFlexFMXCSS*
CSS classes that control this control's column width, spacing and visibility within a `TFlexFMXLayout`. Example: `col-12 col-md-6 m-2`.

LayoutIndex: *Integer*
Explicit position in the parent layout's ordering. Lower index = rendered first.

XRadius / **YRadius**: *Single*
Horizontal and vertical corner radii in pixels. Both default to 5.

Corners: *TCorners*
Set of corners to round: `crTopLeft`, `crTopRight`, `crBottomLeft`, `crBottomRight`. Default: all corners.

CornerType: *TCornerType*
Shape of each rounded corner: `ctRound`, `ctBevel`, `ctInnerRound`, `ctInnerLine`.

Fill: *TBrush*
Background fill. Supports solid colour, gradient and bitmap fills.

Stroke: *TStrokeBrush*
Border brush — set `Stroke.Kind = bkNone` to remove the border.

Usage Example

```
// Designer: drop inside a TFlexFMXLayout
// Set FlexFMXCSS.CSSClasses = 'col-12 col-md-4 m-2'
// Fill with a gradient for a coloured info card

// Code:
RoundRect1.Fill.Kind       := TBrushKind.Gradient;
RoundRect1.Fill.Gradient.Color  := TAlphaColorRec.Cornflowerblue;
RoundRect1.Fill.Gradient.Color1 := TAlphaColorRec.Navy;
RoundRect1.XRadius := 10;
RoundRect1.YRadius := 10;
RoundRect1.Stroke.Kind := TBrushKind.None;
```

Tip: Use `TFlexFMXRoundRect` as a background decoration layer behind other controls — set `HitTest = False` so it doesn't intercept touch/mouse events.

Tip: For pill-shaped tags set `XRadius` and `YRadius` to half the control height (e.g., Height = 28, radii = 14).

← Back to Index

TFlexFMXScrollingFlowLayout

A `TFlowLayout` descendant that automatically resizes its child `TLayout` controls to fill a configurable number of columns based on the current container width (Small / Medium / Large breakpoints). The layout adjusts its own height to fit all children and resets the parent scroll box to the top after each resize.

Overview

- Responsive column counts for Small (≤399 px), Medium (400–800 px) and Large (>800 px) widths
- Automatically sizes child `TLayout` controls to fill the available width
- Self-adjusts height to tightly wrap all visible children
- Optional minimum and maximum child height constraints
- Integrates with `TVertScrollBox` — resets scroll position to top on resize
- Designed to be placed directly inside a `TVertScrollBox`

Key Properties

ColumnCounts: *TControlColumns*
Sub-properties defining how many columns to use at each breakpoint:

- `Small` — columns when layout width ≤ 399 px (default 1)
- `Medium` — columns when layout width 400–800 px (default 2)
- `Large` — columns when layout width > 800 px (default 3)

MinLayoutHeight: *Integer*
Minimum height of the layout in pixels. The layout will not shrink below this value even if content is sparse. Set to `0` to disable (default).

MaxLayoutHeight: *Integer*
Maximum height for each individual child control. Set to `0` to allow children to use their natural height (default).

How It Works

On every `Resize` event the layout calls `ResizeCards` which:

1. Calculates `CardWidth` from the current container width divided by the appropriate column count
2. Sets every child `TLayout` to that width (with 5 px margins)
3. Optionally caps child height at `MaxLayoutHeight`
4. Sums the heights of all visible children to set its own `Height`
5. Resets the parent `TVertScrollBox` viewport to Y = 0

Typical Setup

```
// FMX Designer steps
// 1. Drop a TVertScrollBox on the form, Align = Client
// 2. Drop TFlexFMXScrollingFlowLayout inside it, Align = Top
// 3. Set ColumnCounts.Small = 1, ColumnCounts.Medium = 2, ColumnCounts.Large = 3
// 4. At run time, add TLayout children for each card:

procedure TForm1.LoadCards;
var
  Card: TLayout;
  I: Integer;
begin
  ScrollingFlowLayout1.BeginUpdate;
  try
    for I := 1 to 12 do
    begin
      Card := TLayout.Create(Self);
      Card.Parent := ScrollingFlowLayout1;
      Card.Height := 160;
      // ... populate card contents
    end;
  finally
    ScrollingFlowLayout1.EndUpdate;
    ScrollingFlowLayout1.ResizeCards;
  end;
end;
```

Tip: Call `ResizeCards` manually after dynamically adding or removing children in code.

Tip: On Windows the parent `TVertScrollBox` scroll bar is forced visible (`ShowScrollBars := True`) to avoid layout miscalculations — this is handled automatically.

← Back to Index

TFlexFMXSpacer

A lightweight invisible placeholder control used to pad out a row in a `TFlexFMXLayout`. Visible at design time (grey fill, dashed border) so you can select and size it. Completely invisible and non-interactive at run time.

Overview

- Visible only in the IDE designer — transparent at run time
- Non-interactive (`HitTest = False`)
- Participates in the FlexFMX layout grid via `FlexFMXCSS` and `LayoutIndex`
- Useful for pushing controls to specific columns or adding explicit gaps

Key Properties

FlexFMXCSS: *TFlexFMXCSS*
CSS classes that control the spacer's column width within a `TFlexFMXLayout`. For example `col-6` reserves half a row.

LayoutIndex: *Integer*
Explicit position of this spacer in the parent layout's control ordering (0-based).

Typical Usage

```
// Push a button to the right half of a row
// Drop TFlexFMXSpacer with CSS col-6 first, then TFlexFMXAdvancedButton with col-6

// Designer approach:
// 1. Inside TFlexFMXLayout, drop TFlexFMXSpacer
// 2. Set FlexFMXCSS.CSSClasses = 'col-6'
// 3. Drop your control with FlexFMXCSS.CSSClasses = 'col-6'
// Both fill half the row; the spacer takes the left half invisibly.
```

Tip: The grey dashed outline at design time makes it easy to spot and resize spacers without accidentally selecting the invisible area at run time.

Tip: For simple right-alignment of a single button on a row, use a spacer with `col-9` followed by the button with `col-3`.

← Back to Index

TFlexFMXStyleController

Global style management component for applying Bootstrap color schemes and themes across all FlexFMX components.

Key Properties

ColorScheme: TColorScheme
Active color scheme: csLight, csDark, csAuto (follows system).

PrimaryColor: TAlphaColor
Primary brand color (Bootstrap primary blue by default).

SecondaryColor: TAlphaColor
Secondary accent color.

SuccessColor: TAlphaColor
Success state color (green).

DangerColor: TAlphaColor
Error/danger state color (red).

WarningColor: TAlphaColor
Warning state color (yellow/orange).

InfoColor: TAlphaColor
Informational color (cyan/blue).

LightColor: TAlphaColor
Light background color.

DarkColor: TAlphaColor
Dark text/background color.

CustomCSS: TStringList
Custom CSS class definitions and overrides.

AutoApply: Boolean
Automatically apply styles to all FlexFMX controls on form.

Usage Examples

Dark Mode

```
StyleController1.ColorScheme := csDark;
// All FlexFMX controls automatically update
```

Custom Brand Colors

```
StyleController1.PrimaryColor := $FF6B4C93; // Purple
StyleController1.SecondaryColor := $FF48C9B0; // Turquoise
StyleController1.ApplyStyles; // Update all controls
```

Auto Dark Mode (Follow System)

```
StyleController1.ColorScheme := csAuto;
// Automatically switches between light/dark based on OS settings
```

Custom CSS Class

```
StyleController1.CustomCSS.Add('.my-button {');
StyleController1.CustomCSS.Add('  background-color: #FF5733;');
StyleController1.CustomCSS.Add('  color: white;');
StyleController1.CustomCSS.Add('  border-radius: 20px;');
StyleController1.CustomCSS.Add('}');
StyleController1.ApplyStyles;

// Use in button
Button1.FlexFMXCSS.Text := 'my-button';
```

Material Design Theme

```
StyleController1.PrimaryColor := $FF6200EE; // Material Purple
StyleController1.SecondaryColor := $FF03DAC6; // Material Teal
StyleController1.SuccessColor := $FF4CAF50;
StyleController1.DangerColor := $FFF44336;
StyleController1.WarningColor := $FFFF9800;
StyleController1.ApplyStyles;
```

Methods

- `ApplyStyles` - Apply current style settings to all controls
- `LoadTheme(FileName)` - Load theme from JSON file
- `SaveTheme(FileName)` - Save current theme to JSON file
- `ResetToDefaults` - Restore Bootstrap default colors

Events

- `OnColorSchemeChange` - Color scheme changed (light/dark)
- `OnThemeApplied` - After theme applied to controls

Notes

- Place one StyleController per form or use a global instance
- Changes propagate to all FlexFMX controls automatically
- CSS classes map to Bootstrap 5 conventions
- Supports runtime theme switching without restart
- Auto mode detects system dark mode preference
- Custom CSS uses simplified syntax (key-value pairs)

← Back to Index

← Back to Index

TFlexFMXTileList

Responsive tile/card grid with automatic layout, perfect for image galleries, product catalogs, and dashboards.

Key Properties

Items: TTileItems
Collection of tile items.

TileWidth / TileHeight: Single
Default tile dimensions.

ColumnsCount: Integer
Number of columns (0 = auto-calculate based on width).

Spacing: Single
Gap between tiles.

AutoHeight: Boolean
Automatically adjust tile heights to fit content.

SelectionMode: TSelectionMode
smNone, smSingle, smMultiple.

ShowHeaders: Boolean
Display tile header area.

ShowFooters: Boolean
Display tile footer area.

Tile Item Properties

Title: string
Tile title text.

Description: string
Tile description/subtitle.

ImageBitmap: TBitmap
Tile image.

FooterText: string
Footer text.

BadgeText: string
Optional badge on tile.

Selected: Boolean
Selection state.

Tag: NativeInt
Custom data tag.

Usage Examples

Photo Gallery

```
TileList1.TileWidth := 200;
TileList1.TileHeight := 200;
TileList1.ColumnsCount := 0; // Auto
TileList1.Spacing := 10;

with TileList1.Items.Add do
begin
  Title := 'Sunset';
  ImageBitmap.LoadFromFile('photo1.jpg');
  FooterText := 'July 2024';
end;

with TileList1.Items.Add do
begin
  Title := 'Mountain';
  ImageBitmap.LoadFromFile('photo2.jpg');
  FooterText := 'August 2024';
end;
```

Product Catalog

```
TileList1.SelectionMode := smSingle;
TileList1.ShowFooters := True;

with TileList1.Items.Add do
begin
  Title := 'Laptop Pro';
  Description := 'High-performance laptop';
  ImageBitmap.LoadFromFile('laptop.png');
  FooterText := '$1,299';
  BadgeText := 'NEW';
end;

with TileList1.Items.Add do
begin
  Title := 'Wireless Mouse';
  Description := 'Ergonomic design';
  ImageBitmap.LoadFromFile('mouse.png');
  FooterText := '$49';
end;
```

Dashboard Cards

```
TileList1.TileWidth := 150;
TileList1.TileHeight := 150;
TileList1.ColumnsCount := 3;

with TileList1.Items.Add do
begin
  Title := 'Sales';
  Description := '$45,320';
  BadgeText := '+12%';
end;

with TileList1.Items.Add do
begin
  Title := 'Orders';
  Description := '234';
  BadgeText := '+5%';
end;
```

Methods

- `Clear` - Remove all tiles
- `AddItem(Title, Description)` - Quick add tile
- `GetSelectedItems` - Get array of selected tiles
- `SelectAll / DeselectAll` - Bulk selection
- `RecalculateLayout` - Update tile positions

Events

- `OnItemClick` - Tile clicked (provides Index, Item)
- `OnItemDblClick` - Tile double-clicked
- `OnSelectionChange` - Selection changed
- `OnItemRender` - Before tile rendered (customize)

Design-Time Features

Reorder Tiles Dialog

At design-time, right-click the TFlexFMXTileList component to access the **"Reorder Tiles..."** context menu option. This opens an interactive dialog that allows you to:

- See a list of all tiles/child controls in the TileList
- Select a tile to highlight it with an orange border in the designer
- Use **Move Up** and **Move Down** buttons to reorder tiles
- See changes reflected in real-time in the form designer
- Navigate with keyboard (arrow keys) or mouse
- Press **ESC** to close the dialog

This makes it easy to adjust the z-order and visual arrangement of tiles without manually editing code or using the Structure View.

Notes

- Automatically reflows tiles on resize
- Touch-friendly with swipe scrolling
- Supports virtual scrolling for large datasets
- Use with TFlexFMXLayout for responsive grids
- ColorScheme integration for automatic theming
- Tiles can have custom layouts via OnItemRender
- Design-time reordering dialog provides visual feedback with orange highlighting

← Back to Index

← Back to Index

TFlexFMXTimeline

Vertical or horizontal timeline component for displaying chronological events with icons and descriptions.

Key Properties

Items: TTimelineItems
Collection of timeline event items.

Orientation: TOrientation
Layout: Vertical or Horizontal.

LineColor: TAlphaColor
Timeline connector line color.

LineWidth: Single
Timeline line thickness.

IconSize: Single
Size of event icons/markers.

ItemSpacing: Single
Space between timeline items.

AlternateAlignment: Boolean
Alternate items left/right (vertical) or top/bottom (horizontal).

Timeline Item Properties

Title: string
Event title/heading.

Description: string
Event details text.

DateTime: TDateTime
Event date and time.

IconName: string
Icon identifier or emoji.

IconColor: TAlphaColor
Icon/marker color.

CustomData: TObject
Optional custom data object.

Usage Examples

Vertical Timeline

```
Timeline1.Orientation := Vertical;
Timeline1.LineColor := TAlphaColors.Lightgray;

with Timeline1.Items.Add do
begin
  Title := 'Project Started';
  Description := 'Initial planning phase';
  DateTime := EncodeDate(2024, 1, 15);
  IconName := '□';
  IconColor := TAlphaColors.Blue;
end;

with Timeline1.Items.Add do
begin
  Title := 'Development';
  Description := 'Core features implemented';
  DateTime := EncodeDate(2024, 3, 1);
  IconName := '□□';
  IconColor := TAlphaColors.Orange;
end;

with Timeline1.Items.Add do
begin
  Title := 'Launch';
  Description := 'Product released to public';
  DateTime := EncodeDate(2024, 6, 1);
  IconName := '□';
  IconColor := TAlphaColors.Green;
end;
```

Order History

```
Timeline1.Orientation := Vertical;

with Timeline1.Items.Add do
begin
  Title := 'Order Placed';
  Description := 'Order #12345';
  DateTime := Now - 2;
  IconName := '□';
end;

with Timeline1.Items.Add do
begin
  Title := 'Shipped';
  Description := 'Package sent via UPS';
  DateTime := Now - 1;
  IconName := '□';
end;

with Timeline1.Items.Add do
begin
  Title := 'Delivered';
  Description := 'Signed by recipient';
  DateTime := Now;
  IconName := '□';
end;
```

Horizontal Timeline

```
Timeline1.Orientation := Horizontal;
Timeline1.Height := 120;
// Add events similar to above
```

Methods

- `AddItem(Title, Description, DateTime)` - Quick add item
- `Clear` - Remove all timeline items
- `ScrollToItem(Index)` - Scroll to specific item
- `SortByDate` - Sort items chronologically

Events

- `OnItemClick` - Timeline item clicked (provides Index, Item)
- `OnItemRender` - Before item rendered (customize appearance)

Notes

- Ideal for activity feeds, order tracking, project milestones
- Automatically formats dates based on locale
- Supports emoji icons or custom image icons
- Scrollable for long timelines
- AlternateAlignment creates zigzag layout
- ColorScheme integration for automatic theming

← Back to Index

← Back to Index

TFlexFMXToast

Temporary notification message that appears at screen edge with automatic dismissal.

Key Properties

Message: string
Toast notification text.

Title: string
Optional toast title/heading.

ToastType: TToastType
Type: ttSuccess, ttInfo, ttWarning, ttError, ttDefault.

Duration: Integer
Display duration in milliseconds (0 = manual dismiss).

Position: TToastPosition
Screen position: tpTopLeft, tpTopCenter, tpTopRight, tpBottomLeft, tpBottomCenter, tpBottomRight.

ShowCloseButton: Boolean
Display close (X) button.

ShowIcon: Boolean
Display type icon (checkmark, info, warning, error).

AnimationType: TToastAnimation
Entry/exit animation: taSlide, taFade, taZoom.

Usage Examples

Simple Success Toast

```
Toast1.Message := 'Settings saved successfully!';
Toast1.ToastType := ttSuccess;
Toast1.Duration := 3000; // 3 seconds
Toast1.Show;
```

Error Notification

```
Toast1.Title := 'Error';
Toast1.Message := 'Failed to connect to server';
Toast1.ToastType := ttError;
Toast1.Duration := 5000;
Toast1.Position := tpTopCenter;
Toast1.Show;
```

Custom Position

```
Toast1.Message := 'New message received';
Toast1.ToastType := ttInfo;
Toast1.Position := tpBottomRight;
Toast1.AnimationType := taSlide;
Toast1.Show;
```

Manual Dismiss

```
Toast1.Message := 'Processing...';
Toast1.Duration := 0; // No auto-dismiss
Toast1.ShowCloseButton := True;
Toast1.Show;
// Later...
Toast1.Hide;
```

Methods

- `Show` - Display the toast notification
- `Hide` - Manually dismiss the toast
- `ShowToast(Message, ToastType, Duration)` - Quick show method

Events

- `OnShow` - Toast appears on screen
- `OnHide` - Toast dismissed
- `OnClick` - User clicks toast

Notes

- Multiple toasts can stack vertically
- Non-blocking - doesn't interrupt user workflow
- Automatically adjusts for mobile and desktop layouts
- Use for feedback messages, confirmations, and notifications
- ColorScheme integration for automatic theming

← Back to Index