Complete SnackbarX Package

Code Guide for Beginners

Understanding Every Line of Code

From Models to Widgets to Manager Logic

Flutter Package Development Masterclass

Complete SnackbarX Package Code Guide for Beginners

Table of Contents

1. Package Overview
2. File Structure Explained
3. Core Models
4. Widget Components
5. Manager System
6. Main Package Interface
7. Example Application
8. How Everything Works Together

Package Overview

The SnackbarX package is a complete Flutter solution for showing beautiful notification messages. Think of it like a notification system for your phone, but inside your Flutter app.

What the package does:

- Shows temporary messages to users (success, error, info)
- Handles animations and positioning automatically
- Manages memory and cleanup properly
- Provides a simple API for developers

Package Architecture:

```
SnackbarX Package
├── Models (Data structures)
│   ├── SnackbarType (Success/Error/Info)
│   └── SnackbarConfig (Customization options)
├── Widgets (Visual components)
│   ├── BaseSnackbar (Basic appearance)
│   └── SnackbarContainer (Animation wrapper)
├── Manager (Business logic)
│   └── SnackbarXManager (Controls everything)
└── Main Interface
    └── SnackbarX (Public API)
```

File Structure Explained

The Package Layout

```
lib/
├── snackbarx.dart                ← Main entry point
└── src/
    ├── models/
    │   ├── snackbar_type.dart     ← Defines message types
    │   └── snackbar_config.dart   ← Customization options
    ├── widgets/
    │   ├── base_snackbar.dart     ← Visual appearance
    │   └── snackbar_container.dart ← Animation handling
    └── snackbar_x_manager.dart    ← Core business logic
```

Why this structure? - `lib/`: Main package code - `src/`: Internal implementation (users don't see this directly) - `models/`: Data structures and configurations - `widgets/`: Visual components - **Main file**: Public interface for users

---

Core Models

1. SnackbarType (snackbar_type.dart)

This file defines what types of messages we can show.

```
/// Enum defining the type of snackbar to display
enum SnackbarType {
  /// Success snackbar for positive confirmations
  success,

  /// Error snackbar for errors and failures
  error,

  /// Info snackbar for neutral information
  info,
}
```

What this means: - `enum` is like a multiple choice list - you can pick one option - We have 3 types: success (good news), error (bad news), info (neutral news) - Each type will have different colors and icons

Extension Methods for SnackbarType

```
extension SnackbarTypeExtension on SnackbarType {
  /// Get the default background color for this snackbar type
  Color get backgroundColor {
    switch (this) {
      case SnackbarType.success:
        return const Color(0xFF4CAF50); // Green
      case SnackbarType.error:
        return const Color(0xFFF44336); // Red
      case SnackbarType.info:
        return const Color(0xFF2196F3); // Blue
    }
  }
```

What extensions do: - Add extra functionality to existing types - Like adding new features to your phone through apps - Here we add color and icon properties to each message type

Color explanation: - `Color(0xFF4CAF50)`: Hex color code for green - `0xFF` means fully opaque (not transparent) - `4CAF50` is the green color value

Complete Type Properties

```
  /// Get the default text color for this snackbar type
  Color get textColor {
    switch (this) {
      case SnackbarType.success:
      case SnackbarType.error:
      case SnackbarType.info:
        return Colors.white;
    }
  }

  /// Get the default icon for this snackbar type
  IconData get icon {
    switch (this) {
      case SnackbarType.success:
        return Icons.check_circle_outline;
      case SnackbarType.error:
        return Icons.error_outline;
      case SnackbarType.info:
        return Icons.info_outline;
    }
  }

  /// Get the default icon color for this snackbar type
  Color get iconColor {
    return Colors.white.withOpacity(0.9);
  }
```

What each property does: - `textColor`: Color of the message text (white for all types) - `icon`: Symbol shown next to message (checkmark, error, info) - `iconColor`: Color of the icon (slightly transparent white)

2. SnackbarConfig (snackbar_config.dart)

This file defines all the customization options for snackbars.

```
/// Configuration options for customizing snackbars
class SnackbarConfig {
  /// The duration for which the snackbar will be displayed
  final Duration duration;

  /// The position of the snackbar on the screen
  final SnackbarPosition position;

  /// The background color of the snackbar
  /// If null, will use type-specific default colors
  final Color? backgroundColor;
```

Understanding the properties: - `Duration`: How long the message stays visible - `SnackbarPosition`: Where on screen to show it - `Color?`: The `?` means optional - can be null

All Configuration Options

```
class SnackbarConfig {
  final Duration duration;
  final SnackbarPosition position;
  final Color? backgroundColor;
  final Color? textColor;
  final IconData? icon;
  final Color? iconColor;
  final String? actionLabel;
  final VoidCallback? onActionPressed;
  final Color? actionTextColor;
  final BorderRadius borderRadius;
  final EdgeInsets padding;
  final EdgeInsets margin;
  final Duration animationDuration;
  final double? maxWidth;
  final double? minWidth;
  final double elevation;
  final bool showCloseButton;
  final SnackbarAnimationType animationType;
```

Property explanations: - `duration`: Auto-dismiss timer (default 3 seconds) - `position`: top, center, or bottom of screen - `backgroundColor`: Custom color (overrides type default) - `textColor`: Custom text color - `icon`: Custom icon (overrides type default) - `actionLabel`: Text for action button (like "RETRY") - `onActionPressed`: What happens when action button tapped - `borderRadius`: How rounded the corners are - `padding`: Space inside the snackbar - `margin`: Space around the snackbar - `elevation`: Shadow depth - `showCloseButton`: Whether to show X button

Constructor with Defaults

```
const SnackbarConfig({
  this.duration = const Duration(seconds: 3),
  this.position = SnackbarPosition.bottom,
  this.backgroundColor,
  this.textColor,
  this.icon,
  this.iconColor,
  this.actionLabel,
  this.onActionPressed,
  this.actionTextColor,
  this.borderRadius = const BorderRadius.all(Radius.circular(8)),
  this.padding = const EdgeInsets.symmetric(horizontal: 16, vertical: 12),
  this.margin = const EdgeInsets.all(16),
  this.animationDuration = const Duration(milliseconds: 250),
  this.maxWidth = 600,
  this.minWidth,
  this.elevation = 6,
  this.showCloseButton = false,
  this.animationType = SnackbarAnimationType.slideUp,
});
```

Default values explained: - **3 seconds**: Good balance between readable and not annoying - **Bottom position**: Follows mobile app conventions - **8px border radius**: Slightly rounded, modern look - **16px horizontal, 12px vertical padding**: Comfortable spacing - **250ms animation**: Fast enough to feel responsive - **600px max width**: Prevents snackbars from being too wide on large screens

Enums for Position and Animation

```
/// Enum defining the position of the snackbar on screen
enum SnackbarPosition {
  /// Display at the top of the screen
  top,

  /// Display at the bottom of the screen
  bottom,

  /// Display in the center of the screen
  center,
}

/// Enum defining the animation type for snackbar entry and exit
enum SnackbarAnimationType {
  /// Slide up from the bottom (or down from the top)
  slideUp,

  /// Fade in/out
  fade,

  /// Scale up/down
  scale,

  /// Slide + fade combined
  slideAndFade,
}
```

Widget Components

1. BaseSnackbar (base_snackbar.dart)

This widget handles the visual appearance of the snackbar.

```
/// Base widget for all snackbar variants
class BaseSnackbar extends StatelessWidget {
  /// The message to display
  final String message;

  /// The type of snackbar
  final SnackbarType type;

  /// Configuration options
  final SnackbarConfig config;

  /// Callback when dismiss is requested
  final VoidCallback onDismiss;

  const BaseSnackbar({
    Key? key,
    required this.message,
    required this.type,
    required this.config,
    required this.onDismiss,
  }) : super(key: key);
```

Understanding StatelessWidget: - A widget that doesn't change after it's built - Like a printed sign - the content doesn't change - Perfect for displaying static content

Required parameters: - `message`: The text to show - `type`: Success, error, or info - `config`: Customization options - `onDismiss`: Function to call when closing

Building the Visual Layout

```
@override
Widget build(BuildContext context) {
  final backgroundColor = config.backgroundColor ?? type.backgroundColor;
  final textColor = config.textColor ?? type.textColor;
  final iconData = config.icon ?? type.icon;
  final iconColor = config.iconColor ?? type.iconColor;

  return Material(
    elevation: config.elevation,
    borderRadius: config.borderRadius,
    color: backgroundColor,
    child: Container(
      constraints: BoxConstraints(
        maxWidth: config.maxWidth ?? double.infinity,
        minWidth: config.minWidth ?? 0,
      ),
      margin: config.margin,
      padding: config.padding,
      child: Row(
        mainAxisSize: MainAxisSize.min,
        children: [
          // Icon, Message, Action Button, Close Button
        ],
      ),
    ),
  );
}
```

Understanding the null-coalescing operator (`??`):

```
final backgroundColor = config.backgroundColor ?? type.backgroundColor;
```

This means: "Use config.backgroundColor if it exists, otherwise use type.backgroundColor"

Layout structure: - **Material**: Provides elevation (shadow) and rounded corners - **Container**: Applies size constraints, margin, and padding - **Row**: Arranges icon, text, and buttons horizontally - **MainAxisSize.min**: Row takes only the space it needs

Icon Section

```
// Icon
Icon(
  iconData,
  color: iconColor,
  size: 24,
),
const SizedBox(width: 12),
```

What this does: - Shows the appropriate icon (checkmark, error, info) - Sets color and size - Adds 12 pixels of space after the icon

Message Section

```
// Message
Expanded(
  child: Text(
    message,
    style: TextStyle(
      color: textColor,
      fontSize: 14,
      fontWeight: FontWeight.w500,
    ),
    overflow: TextOverflow.ellipsis,
    maxLines: 2,
  ),
),
```

Understanding Expanded: - Takes up all remaining space in the Row - Prevents overflow when message is long

Text styling: - **fontSize: 14**: Readable size - **FontWeight.w500**: Medium weight (between normal and bold) - **TextOverflow.ellipsis**: Shows "..." if text is too long - **maxLines: 2**: Limits to 2 lines maximum

Action Button Section

```
// Action button if provided
if (config.actionLabel != null && config.onActionPressed != null) ...[
  const SizedBox(width: 8),
  TextButton(
    onPressed: config.onActionPressed,
    style: TextButton.styleFrom(
      foregroundColor: config.actionTextColor ?? textColor,
      padding: const EdgeInsets.symmetric(horizontal: 8),
      tapTargetSize: MaterialTapTargetSize.shrinkWrap,
    ),
    child: Text(
      config.actionLabel!,
      style: const TextStyle(
        fontWeight: FontWeight.bold,
      ),
    ),
  ),
],
```

Understanding conditional rendering: - `if (condition) ...[ widgets ]` means "only show these widgets if condition is true" - Only shows action button if both label and callback are provided

TextButton styling: - **tapTargetSize.shrinkWrap**: Reduces touch area to content size - **foregroundColor**: Color of button text - **FontWeight.bold**: Makes action text stand out

Close Button Section

```
// Close button if enabled
if (config.showCloseButton) ...[
  const SizedBox(width: 4),
  IconButton(
    icon: Icon(Icons.close, color: textColor.withOpacity(0.7), size: 20),
    padding: EdgeInsets.zero,
    constraints: const BoxConstraints(
      minWidth: 32,
      minHeight: 32,
    ),
    onPressed: onDismiss,
  ),
],
```

Close button details: - **withOpacity(0.7)**: Makes close icon slightly transparent - **padding: EdgeInsets.zero**: Removes default button padding - **constraints**: Sets minimum touch area for accessibility

2. SnackbarContainer (snackbar_container.dart)

This widget handles all animations and positioning.

```
/// Container widget that handles positioning and animation of snackbars
class SnackbarContainer extends StatefulWidget {
  final String message;
  final SnackbarType type;
  final SnackbarConfig config;
  final AnimationController animationController;
  final VoidCallback onDismiss;

  const SnackbarContainer({
    Key? key,
    required this.message,
    required this.type,
    required this.config,
    required this.animationController,
    required this.onDismiss,
  }) : super(key: key);

  @override
  State<SnackbarContainer> createState() => _SnackbarContainerState();
}
```

Why StatefulWidget here? - Animations require state management - State can change during the animation lifecycle - Like a video player - the state changes as it plays

Animation State Management

```
class _SnackbarContainerState extends State<SnackbarContainer> {
  // Animations
  late Animation<double> _fadeAnimation;
  late Animation<Offset> _slideAnimation;
  late Animation<double> _scaleAnimation;

  @override
  void initState() {
    super.initState();

    // Set up animations based on the animation type
    _setupAnimations();
  }
```

Understanding `late` keyword: - Tells Dart "I promise to initialize this before using it" - Used when we can't initialize immediately but will do so in initState

Animation types: - **Fade**: Changes opacity from 0 (invisible) to 1 (visible) - **Slide**: Moves from off-screen to final position - **Scale**: Changes size from small to normal

Setting Up Different Animation Types

```
void _setupAnimations() {
  // Configure animations based on type
  switch (widget.config.animationType) {
    case SnackbarAnimationType.slideUp:
      _fadeAnimation = Tween<double>(begin: 1.0, end: 1.0)
          .animate(widget.animationController);

      final beginOffset = widget.config.position == SnackbarPosition.top
          ? const Offset(0, -1)
          : const Offset(0, 1);

      _slideAnimation = Tween<Offset>(
        begin: beginOffset,
        end: Offset.zero,
      ).animate(CurvedAnimation(
        parent: widget.animationController,
        curve: Curves.easeOutCubic,
        reverseCurve: Curves.easeInCubic,
      ));

      _scaleAnimation = Tween<double>(begin: 1.0, end: 1.0)
          .animate(widget.animationController);
      break;
```

Understanding Tween: - "Between" - defines start and end values - `Tween<double>(begin: 0.0, end: 1.0)` means animate from 0 to 1

Understanding Offset: - `Offset(0, -1)` means "one screen height above normal position" - `Offset(0, 1)` means "one screen height below normal position" - `Offset.zero` means "normal position"

Understanding Curves: - **easeOutCubic**: Starts fast, slows down at end (feels natural) - **easeInCubic**: Starts slow, speeds up at end (for reverse animation)

Fade Animation Setup

```
case SnackbarAnimationType.fade:
  _fadeAnimation = Tween<double>(begin: 0.0, end: 1.0)
      .animate(CurvedAnimation(
    parent: widget.animationController,
    curve: Curves.easeOut,
  ));

  _slideAnimation = Tween<Offset>(
    begin: Offset.zero,
    end: Offset.zero,
  ).animate(widget.animationController);

  _scaleAnimation = Tween<double>(begin: 1.0, end: 1.0)
      .animate(widget.animationController);
  break;
```

For fade animation: - Only fade changes (0.0 to 1.0) - Slide and scale stay constant - Simpler but elegant effect

Scale Animation Setup

```
case SnackbarAnimationType.scale:
  _fadeAnimation = Tween<double>(begin: 0.0, end: 1.0)
      .animate(CurvedAnimation(
    parent: widget.animationController,
    curve: Curves.easeOut,
  ));

  _slideAnimation = Tween<Offset>(
    begin: Offset.zero,
    end: Offset.zero,
  ).animate(widget.animationController);

  _scaleAnimation = Tween<double>(begin: 0.8, end: 1.0)
      .animate(CurvedAnimation(
    parent: widget.animationController,
    curve: Curves.easeOutCubic,
  ));
  break;
```

For scale animation: - Fade in from invisible to visible - Scale from 80% size to normal size - Creates a "pop in" effect

Combined Slide and Fade

```
case SnackbarAnimationType.slideAndFade:
  _fadeAnimation = Tween<double>(begin: 0.0, end: 1.0)
      .animate(CurvedAnimation(
    parent: widget.animationController,
    curve: Curves.easeOut,
  ));

  final beginOffset = widget.config.position == SnackbarPosition.top
      ? const Offset(0, -0.3)
      : const Offset(0, 0.3);

  _slideAnimation = Tween<Offset>(
    begin: beginOffset,
    end: Offset.zero,
  ).animate(CurvedAnimation(
    parent: widget.animationController,
    curve: Curves.easeOutCubic,
    reverseCurve: Curves.easeInCubic,
  ));

  _scaleAnimation = Tween<double>(begin: 1.0, end: 1.0)
      .animate(widget.animationController);
  break;
```

Combined effect: - Slides in from 30% off-screen (smaller distance than pure slide) - Fades in simultaneously - Creates smooth, polished effect

Building the Animated Widget

```
@override
Widget build(BuildContext context) {
  return AnimatedBuilder(
    animation: widget.animationController,
    builder: (_, child) {
      return FadeTransition(
        opacity: _fadeAnimation,
        child: SlideTransition(
          position: _slideAnimation,
          child: ScaleTransition(
            scale: _scaleAnimation,
            child: SafeArea(
              child: Material(
                color: Colors.transparent,
                child: _positionSnackbar(),
              ),
            ),
          ),
        ),
      );
    },
  );
}
```

Understanding AnimatedBuilder: - Rebuilds the widget whenever the animation value changes - Like refreshing the screen 60 times per second during animation

Transition hierarchy: 1. **FadeTransition**: Controls opacity 2. **SlideTransition**: Controls position 3. **ScaleTransition**: Controls size 4. **SafeArea**: Avoids system UI (status bar, notch) 5. **Material**: Provides transparent background

Positioning Logic

```
Widget _positionSnackbar() {
  final snackbar = BaseSnackbar(
    message: widget.message,
    type: widget.type,
    config: widget.config,
    onDismiss: widget.onDismiss,
  );

  switch (widget.config.position) {
    case SnackbarPosition.top:
      return Align(
        alignment: Alignment.topCenter,
        child: snackbar,
      );
    case SnackbarPosition.center:
      return Center(child: snackbar);
    case SnackbarPosition.bottom:
      return Align(
        alignment: Alignment.bottomCenter,
        child: snackbar,
      );
  }
}
```

Positioning explained: - **Align**: Positions child within available space - **Alignment.topCenter**: Top middle of screen - **Center**: Exact center of screen - **Alignment.bottomCenter**: Bottom middle of screen

---

Manager System

SnackbarXManager (snackbar_x_manager.dart)

This is the brain of the entire system. I'll explain the key parts:

Singleton Pattern Implementation

```
class SnackbarXManager {
  /// Singleton instance
  static final SnackbarXManager instance = SnackbarXManager._();

  /// Private constructor
  SnackbarXManager._();
```

Why singleton? - Only one manager should exist - Prevents conflicts between multiple snackbars - Global access from anywhere

State Variables

```
  /// Global key to maintain reference to the navigator
  GlobalKey<NavigatorState>? _navigatorKey;

  /// Current overlay entry
  OverlayEntry? _currentOverlayEntry;

  /// Timer for auto-dismissal
  Timer? _dismissTimer;

  /// Animation controller
  AnimationController? _animationController;

  /// TickerProvider for animations
  TickerProvider? _tickerProvider;

  /// Whether the snackbar system has been initialized
  bool _isInitialized = false;
```

State management: - **_navigatorKey**: Way to find the overlay - **_currentOverlayEntry**: The snackbar currently showing - **_dismissTimer**: Countdown to auto-remove - **_animationController**: Controls smooth animations - **_tickerProvider**: Provides animation timing - **_isInitialized**: Safety check

Initialization Method

```
void init({GlobalKey<NavigatorState>? navigatorKey, TickerProvider? tickerProvider}) {
  _navigatorKey = navigatorKey;
  _tickerProvider = tickerProvider;
  _isInitialized = true;
}
```

Setup process: 1. Store the navigator key (for finding overlay) 2. Store the ticker provider (for animations) 3. Mark as ready to use

The Main Show Method

```
void show({
  required String message,
  required SnackbarType type,
  required SnackbarConfig config,
  BuildContext? context,
  TickerProvider? tickerProvider,
}) {
  if (!_isInitialized) {
    throw Exception('SnackbarX not initialized. Call SnackbarX.init() first.');
  }

  // Make sure any previous snackbars are dismissed
  _dismissCurrentSnackbar();
```

Safety first: - Check if system is initialized - Remove any existing snackbar (one at a time rule)

Finding the Overlay

```
  // Get the overlay state - try multiple approaches
  OverlayState? overlayState;

  // First try using provided context if available
  if (context != null) {
    try {
      overlayState = Overlay.of(context);
    } catch (e) {
      print('SnackbarX: Could not get overlay from context: $e');
    }
  }

  // Next try using the navigator key if available
  if (overlayState == null && _navigatorKey?.currentContext != null) {
    try {
      overlayState = Overlay.of(_navigatorKey!.currentContext!);
    } catch (e) {
      print('SnackbarX: Could not get overlay from navigatorKey: $e');
    }
  }

  // If we still couldn't find an overlay, throw an error
  if (overlayState == null) {
    throw Exception(
      'No Overlay found. Please provide a valid context in the show method or pass a navigatorKey
      'Make sure you\'re calling this method after MaterialApp has been created.'
    );
  }
```

Finding overlay (like finding a bulletin board): 1. **Method 1**: Use provided context (direct path) 2. **Method 2**: Use stored navigator key (backup path) 3. **Method 3**: Give up and show error (safety net)

Creating the Snackbar

```
// Get the ticker provider
final TickerProvider vsync = tickerProvider ??
                            _tickerProvider ??
                            _createSimpleTickerProvider();

// Create a new controller for this snackbar
_animationController = AnimationController(
  vsync: vsync,
  duration: config.animationDuration,
);

// Create and insert the overlay entry
_currentOverlayEntry = OverlayEntry(
  builder: (context) => SnackbarContainer(
    message: message,
    type: type,
    config: config,
    animationController: _animationController!,
    onDismiss: dismiss,
  ),
);

overlayState.insert(_currentOverlayEntry!);

// Start the animation
_animationController!.forward();

// Set up auto-dismiss timer if duration > 0
if (config.duration.inMilliseconds > 0) {
  _dismissTimer = Timer(config.duration, () {
    dismiss();
  });
}
```

Creation process: 1. Get animation timing provider 2. Create animation controller 3. Package snackbar in overlay entry 4. Insert into overlay (make visible) 5. Start entrance animation 6. Set auto-dismiss timer

Dismissal Process

```
/// Dismisses the current snackbar if one is visible
void dismiss() {
  if (_currentOverlayEntry != null && _animationController != null) {
    _animationController!.reverse().then((_) {
      _dismissCurrentSnackbar();
    });
  }
}

/// Helper method to clean up resources when dismissing a snackbar
void _dismissCurrentSnackbar() {
  // Cancel the dismiss timer if it's active
  _dismissTimer?.cancel();
  _dismissTimer = null;

  // Remove the overlay entry if it exists
  _currentOverlayEntry?.remove();
  _currentOverlayEntry = null;

  // Dispose the animation controller
  _animationController?.dispose();
  _animationController = null;
}
```

Dismissal steps: 1. Start exit animation 2. Wait for animation to complete 3. Cancel any timers 4. Remove from overlay 5. Free animation resources 6. Reset all variables

---

Main Package Interface

SnackbarX (snackbarx.dart)

This is what users actually interact with - the public API.

```
/// Main class for showing customizable snackbars without requiring BuildContext
class SnackbarX {
  /// Private constructor to prevent instantiation
  SnackbarX._();

  /// A global navigator key that can be used with MaterialApp
  static final GlobalKey<NavigatorState> navigatorKey = GlobalKey<NavigatorState>();
```

Why private constructor? - Prevents users from creating instances - All methods are static (class-level, not instance-level) - Like a utility class - tools you use, not objects you create

Convenience Methods

```
/// Shows a success snackbar with the given [message]
static void showSuccess(
  String message, {
  BuildContext? context,
  SnackbarConfig? config,
  TickerProvider? tickerProvider,
}) {
  SnackbarXManager.instance.show(
    message: message,
    type: SnackbarType.success,
    config: config ?? const SnackbarConfig(),
    context: context,
    tickerProvider: tickerProvider,
  );
}

/// Shows an error snackbar with the given [message]
static void showError(
  String message, {
  BuildContext? context,
  SnackbarConfig? config,
  TickerProvider? tickerProvider,
}) {
  SnackbarXManager.instance.show(
    message: message,
    type: SnackbarType.error,
    config: config ?? const SnackbarConfig(),
    context: context,
    tickerProvider: tickerProvider,
  );
}

/// Shows an info snackbar with the given [message]
static void showInfo(
  String message, {
  BuildContext? context,
  SnackbarConfig? config,
  TickerProvider? tickerProvider,
}) {
  SnackbarXManager.instance.show(
    message: message,
    type: SnackbarType.info,
    config: config ?? const SnackbarConfig(),
    context: context,
    tickerProvider: tickerProvider,
  );
}
```

Convenience methods: - Simple wrappers around the manager - Pre-set the type (success/error/info) - Provide defaults for optional parameters - Make the API easy to use

Generic Show Method

```
/// Shows a custom snackbar with the given [message] and [type]
static void show(
  String message,
  SnackbarType type, {
  BuildContext? context,
  SnackbarConfig? config,
  TickerProvider? tickerProvider,
}) {
  SnackbarXManager.instance.show(
    message: message,
    type: type,
    config: config ?? const SnackbarConfig(),
    context: context,
    tickerProvider: tickerProvider,
  );
}
```

Generic method: - Allows any type to be specified - More flexible than convenience methods - Still provides defaults

Initialization and Control

```
/// Initializes the snackbar service
static void init({
  GlobalKey<NavigatorState>? navigatorKey,
  TickerProvider? tickerProvider,
}) {
  SnackbarXManager.instance.init(
    navigatorKey: navigatorKey ?? SnackbarX.navigatorKey,
    tickerProvider: tickerProvider,
  );
}

/// Dismisses any currently showing snackbar
static void dismiss() {
  SnackbarXManager.instance.dismiss();
}
```

Control methods: - **init**: Set up the system - **dismiss**: Force close any snackbar

Example Application

Main App Setup (example/lib/main.dart)

```
import 'package:flutter/material.dart';
import 'package:snackbarx/snackbarx.dart';

void main() {
  WidgetsFlutterBinding.ensureInitialized();
  runApp(const MyApp());
}

class MyApp extends StatelessWidget {
  const MyApp({Key? key}) : super(key: key);

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'SnackbarX Demo',
      navigatorKey: SnackbarX.navigatorKey,
      theme: ThemeData(
        primarySwatch: Colors.blue,
        useMaterial3: true,
      ),
      home: const HomePageState(),
    );
  }
}
```

Key setup points: - **WidgetsFlutterBinding.ensureInitialized()**: Ensures Flutter is ready - **navigatorKey: SnackbarX.navigatorKey**: Connects the overlay system - **Material3**: Modern design system

StatefulWidget with TickerProvider

```
class HomePageState extends StatefulWidget {
  const HomePageState({Key? key}) : super(key: key);

  @override
  State<HomePageState> createState() => _HomePageStateState();
}

class _HomePageStateState extends State<HomePageState> with TickerProviderStateMixin {
  @override
  void initState() {
    super.initState();
    // Initialize the snackbar with both navigator key and ticker provider
    SnackbarX.init(
      navigatorKey: SnackbarX.navigatorKey,
      tickerProvider: this,
    );
  }
```

TickerProviderStateMixin: - Provides the "heartbeat" for animations - Required for smooth animation timing - Mixed into the State class

Example Usage Buttons

```
ElevatedButton(
  onPressed: () {
    SnackbarX.showSuccess(
      'Operation completed successfully!',
      context: context,
      tickerProvider: this,
    );
  },
  child: const Text('Success Snackbar'),
),

ElevatedButton(
  onPressed: () {
    SnackbarX.showError(
      'Connection lost',
      context: context,
      tickerProvider: this,
      config: SnackbarConfig(
        actionLabel: 'RETRY',
        onActionPressed: () {
          SnackbarX.showInfo(
            'Retrying connection...',
            context: context,
            tickerProvider: this,
          );
        },
        animationType: SnackbarAnimationType.slideAndFade,
      ),
    );
  },
  child: const Text('With Action Button'),
),
```

Usage patterns: - Always pass context and tickerProvider for best results - Use configuration for customization - Action buttons can trigger other snackbars

---

How Everything Works Together

The Complete Flow

1. **User taps button** `dart SnackbarX.showSuccess('File saved!', context: context);`

2. **SnackbarX class receives call** `dart static void showSuccess(String message, {BuildContext? context, ...}) { SnackbarXManager.instance.show( message: message, type: SnackbarType.success, // ... ); }`

3. **Manager processes the request** `dart void show({required String message, required SnackbarType type, ...}) { // 1. Check initialization // 2. Dismiss existing snackbar // 3. Find`

`overlay // 4. Create animation controller // 5. Create and insert overlay entry // 6. Start animation // 7. Set timer }`

4. **SnackbarContainer handles animation** `dart Widget build(BuildContext context) { return AnimatedBuilder(animation: widget.animationController, builder: (_, child) { return FadeTransition(opacity: _fadeAnimation, child: SlideTransition(position: _slideAnimation, child: ScaleTransition(scale: _scaleAnimation, child: // BaseSnackbar),),); },); }`

5. **BaseSnackbar renders the content** `dart Widget build(BuildContext context) { return Material( child: Container( child: Row( children: [ Icon(), // Type-specific icon Text(), // User message TextButton(), // Optional action IconButton(), // Optional close ], ), ), ); }`

6. **Timer automatically dismisses** `dart Timer(config.duration, () { dismiss(); // Starts exit animation and cleanup });`

Data Flow Diagram

```
User Action
    ↓
SnackbarX (Public API)
    ↓
SnackbarXManager (Business Logic)
    ↓
OverlayEntry (Positioning System)
    ↓
SnackbarContainer (Animation Handler)
    ↓
BaseSnackbar (Visual Renderer)
    ↓
Screen Display
```

Component Responsibilities

- **SnackbarX**: Simple, user-friendly API
- **SnackbarXManager**: Business logic, state management, overlay handling
- **SnackbarContainer**: Animation orchestration, positioning
- **BaseSnackbar**: Visual rendering, user interaction
- **Models**: Data structures and configuration

Memory Management Flow

```
Show Snackbar:
1. Create AnimationController
2. Create OverlayEntry
3. Start Timer

Dismiss Snackbar:
1. Cancel Timer
2. Remove OverlayEntry
3. Dispose AnimationController
4. Reset all references to null
```

This comprehensive architecture ensures: - **Reliability**: Proper initialization and error handling - **Performance**: Efficient memory management and smooth animations - **Usability**: Simple API with powerful customization - **Maintainability**: Clear separation of concerns and organized code structure

The package demonstrates advanced Flutter concepts like overlay management, animation coordination, singleton patterns, and memory lifecycle management, all wrapped in a beginner-friendly API.