# UltraCanvas TextStyle Programmer's Guide

## Introduction

The **TextStyle** function provides a simple way to access both system-defined and custom text styles in UltraCanvas applications. It returns a `TextStyle` structure that can be directly used with the existing `SetTextStyle()` function, eliminating the need to manually set individual font, color, and formatting properties.

The system automatically handles OS theme integration, ensuring your applications look native while maintaining consistent behavior across platforms. Simply call `TextStyle("StyleName")` to get a complete text configuration, then apply it with `SetTextStyle()`.

**Key Benefits:**

- **One-line text styling** - `SetTextStyle(TextStyle("ButtonText"))`
- **Automatic OS theme integration** - Looks native on every platform
- **Zero setup required** - Automatically initializes on first use
- **Custom styles supported** - Add your own application-specific styles

---

### Automatic Style Selection (NEW!)

**`TextStyle TextStyleAuto(const ElementType* element)`**

**Purpose:** Automatically select the appropriate text style based on the element type.

**Parameters:**

- `element` - Pointer to the UI element (button, label, etc.)

**Returns:** `TextStyle` configured for the specific element type

**Usage:**

```cpp
class UltraCanvasButton : public UltraCanvasElement {
    void Render() override {
        // OLD WAY: Manual style selection
        SetTextStyle(TextStyle("ButtonText"));
        DrawText(buttonText, position);

        // NEW WAY: Automatic style selection
        SetTextStyle(TextStyleAuto(this));  // Automatically uses "ButtonText"
        DrawText(buttonText, position);
    }
};

class UltraCanvasLabel : public UltraCanvasElement {
    void Render() override {
        // Automatically selects "LabelText" style
        SetTextStyle(TextStyleAuto(this));
        DrawText(labelText, position);
    }
};

class UltraCanvasTextInput : public UltraCanvasElement {
    void Render() override {
        // Automatically selects "InputText" style
        SetTextStyle(TextStyleAuto(this));
        DrawText(inputText, position);
    }
};
```

**Element Type Mapping:**

| Element Class | Auto-Selected Style |
|---|---|
| `UltraCanvasButton` | `"ButtonText"` |
| `UltraCanvasLabel` | `"LabelText"` |
| `UltraCanvasTextInput` | `"InputText"` |
| `UltraCanvasTextArea` | `"InputText"` |
| `UltraCanvasMenu` | `"MenuText"` |
| `UltraCanvasDropdown` | `"MenuText"` |
| `UltraCanvasWindow` (title) | `"WindowTitle"` |
| `UltraCanvasTooltip` | `"TooltipText"` |
| `UltraCanvasStatus*` | `"StatusText"` |
| `UltraCanvasHeader*` | `"HeaderText"` |
| `UltraCanvasCaption*` | `"CaptionText"` |
| `*Link*` | `"LinkText"` |
| Other elements | `"BodyText"` (default) |

**Benefits:**

- **Zero configuration** - Elements automatically get appropriate styling
- **Consistent appearance** - All buttons use ButtonText, all labels use LabelText
- **OS theme compliance** - Auto-styles respect system theme settings
- **Maintainable** - Change system style once, affects all elements of that type

---

## Application Developer Functions

| Function | Description |
|---|---|
| `TextStyle(name)` | Get complete text style by name - ready for `SetTextStyle()` |
| `AddTextStyle()` | Add custom text style for your application |
| `ClearAppStyles()` | Remove all styles created by your application |
| `StyleExists()` | Check if a style name is available |

### Convenience Usage Patterns

| Pattern | Description |
|---|---|
| `SetTextStyle(TextStyle("ButtonText"))` | Apply system button style |
| `SetTextStyle(TextStyle("MyCustomStyle"))` | Apply your custom style |

---

## Function Documentation

### Primary Usage Function

`TextStyle TextStyle(const std::string& styleTitle)`

**Purpose:** Get a complete text style configuration ready for immediate use with `SetTextStyle()`.

**Parameters:**

- `styleTitle` - Name of the style (e.g., "ButtonText", "HeaderText", "MyAppTitle")

**Returns:**

- `TextStyle` struct with all formatting properties set
- If style not found, returns default style and shows error dialog

**Usage:**

```cpp
// Simple one-line text styling
SetTextStyle(TextStyle("ButtonText"));
DrawText("OK", buttonPosition);

// For headers
SetTextStyle(TextStyle("HeaderText"));
DrawText("Application Settings", headerPosition);

// For your custom styles
SetTextStyle(TextStyle("MyAppTitle"));
DrawText("My Application", titlePosition);
```

**Before (Wrong Way):**

```cpp
// DON'T: Manual property setting
TextStyleConfig* style = ULTRA_SYSTEM_STYLE(ButtonText);
SetFont(style->fontFamily, style->fontSize);
SetTextColor(style->textColor);
// ... more manual setup
DrawText("OK", position);
```

**After (Correct Way):**

```cpp
// DO: Direct style application
SetTextStyle(TextStyle("ButtonText"));
DrawText("OK", position);
```

---

## Custom Style Management

**`bool AddTextStyle(const std::string& styleTitle, const TextStyle& style, const std::string& ownerApp)`**

**Purpose:** Add a custom text style that your application can use.

**Parameters:**

- `styleTitle` - Unique name for your style
- `style` - Complete `TextStyle` configuration
- `ownerApp` - Your application name (enables automatic cleanup)

**Returns:**

- `true` if style was added successfully
- `false` if style name already exists (shows error dialog)

**Usage:**

```cpp
// Create your custom text style
TextStyle myTitleStyle;
myTitleStyle.fontFamily = "Roboto";
myTitleStyle.fontSize = 18.0f;
myTitleStyle.fontWeight = FontWeight::Bold;
myTitleStyle.textColor = Color(32, 64, 128, 255);
myTitleStyle.alignment = TextAlign::Center;

// Add to the system
ThemeControl& theme = ThemeControl::GetInstance();
bool success = theme.AddTextStyle("ProductTitle", myTitleStyle, "MyApplication");

if (success) {
    // Now use it anywhere in your app:
    SetTextStyle(TextStyle("ProductTitle"));
    DrawText("My Product", titlePosition);
}
```

---

**`int ClearAppStyles(const std::string& appName)`**

**Purpose:** Remove all custom text styles created by your application.

**Parameters:**

- `appName` - Your application name

**Returns:** Number of styles removed

**Usage:**

```cpp
// Call during application shutdown
void MyApplication::Shutdown() {
    ThemeControl& theme = ThemeControl::GetInstance();
    int removed = theme.ClearAppStyles("MyApplication");
    std::cout << "Cleaned up " << removed << " styles" << std::endl;
}
```

---

**`bool StyleExists(const std::string& styleTitle)`**

**Purpose:** Check if a style is available before using it.

**Usage:**

```cpp
// Safe style usage with fallback
if (StyleExists("MyCustomStyle")) {
    SetTextStyle(TextStyle("MyCustomStyle"));
} else {
    SetTextStyle(TextStyle("BodyText"));  // Fallback to system style
}
DrawText("Content", position);
```

---

## Predefined Text Styles (Always Available)

### Standard UI Elements

| Style Name | Purpose | Example Usage |
|---|---|---|
| **"WindowTitle"** | Window title bars | `SetTextStyle(TextStyle("WindowTitle"))` |
| **"MenuText"** | Menu items | `SetTextStyle(TextStyle("MenuText"))` |
| **"ButtonText"** | Button labels | `SetTextStyle(TextStyle("ButtonText"))` |
| **"LabelText"** | Form labels | `SetTextStyle(TextStyle("LabelText"))` |
| **"InputText"** | Text input fields | `SetTextStyle(TextStyle("InputText"))` |
| **"TooltipText"** | Hover help text | `SetTextStyle(TextStyle("TooltipText"))` |
| **"StatusText"** | Status bar messages | `SetTextStyle(TextStyle("StatusText"))` |

### Content Hierarchy

| Style Name | Purpose | Example Usage |
|---|---|---|
| **"HeaderText"** | Main headings | `SetTextStyle(TextStyle("HeaderText"))` |
| **"SubHeaderText"** | Secondary headings | `SetTextStyle(TextStyle("SubHeaderText"))` |
| **"BodyText"** | Main content text | `SetTextStyle(TextStyle("BodyText"))` |
| **"CaptionText"** | Small captions | `SetTextStyle(TextStyle("CaptionText"))` |
| **"SmallText"** | Fine print | `SetTextStyle(TextStyle("SmallText"))` |

### Interactive & Message Styles

| Style Name | Purpose | Example Usage |
|---|---|---|
| **"LinkText"** | Clickable links | `SetTextStyle(TextStyle("LinkText"))` |
| **"ErrorText"** | Error messages | `SetTextStyle(TextStyle("ErrorText"))` |
| **"WarningText"** | Warning messages | `SetTextStyle(TextStyle("WarningText"))` |
| **"SuccessText"** | Success messages | `SetTextStyle(TextStyle("SuccessText"))` |
| **"InfoText"** | Information messages | `SetTextStyle(TextStyle("InfoText"))` |

---

## Complete Application Example

cpp

```cpp
#include "UltraCanvasThemeControl.h"
#include "UltraCanvasRenderInterface.h"

class MyApplicationWindow : public UltraCanvasWindow {
public:
    MyApplicationWindow() {
        SetupCustomStyles();
    }

    ~MyApplicationWindow() {
        // Automatic cleanup of custom styles
        ThemeControl::GetInstance().ClearAppStyles("MyApplication");
    }

private:
    void SetupCustomStyles() {
        ThemeControl& theme = ThemeControl::GetInstance();

        // Create application title style
        TextStyle appTitleStyle;
        appTitleStyle.fontFamily = "Roboto";
        appTitleStyle.fontSize = 24.0f;
        appTitleStyle.fontWeight = FontWeight::Bold;
        appTitleStyle.textColor = Color(32, 64, 128, 255);
        appTitleStyle.alignment = TextAlign::Center;

        theme.AddTextStyle("AppTitle", appTitleStyle, "MyApplication");

        // Create data display style
        TextStyle dataStyle;
        dataStyle.fontFamily = "Consolas";
        dataStyle.fontSize = 11.0f;
        dataStyle.textColor = Color(0, 128, 0, 255);

        theme.AddTextStyle("DataDisplay", dataStyle, "MyApplication");
    }

    void Render() override {
        UltraCanvasWindow::Render();

        // Render application title
        SetTextStyle(TextStyle("AppTitle"));
        DrawText("My Application", Point2D(400, 50));
```

```
        // Render standard UI elements
        SetTextStyle(TextStyle("ButtonText"));
        DrawText("OK", Point2D(350, 300));
        DrawText("Cancel", Point2D(420, 300));

        // Render status information
        SetTextStyle(TextStyle("StatusText"));
        DrawText("Ready", Point2D(10, 450));

        // Render data with custom style
        SetTextStyle(TextStyle("DataDisplay"));
        DrawText("Data: 12,345 records", Point2D(10, 100));

        // Render different message types
        SetTextStyle(TextStyle("ErrorText"));
        DrawText("Error: File not found", Point2D(10, 150));

        SetTextStyle(TextStyle("WarningText"));
        DrawText("Warning: Unsaved changes", Point2D(10, 180));

        SetTextStyle(TextStyle("SuccessText"));
        DrawText("Success: File saved", Point2D(10, 210));
    }
};
```

## Key Advantages of Auto Styling

### ✅ Before vs After Comparison

**BEFORE: Manual styling in each element**

```cpp
class UltraCanvasButton : public UltraCanvasElement {
    void Render() override {
        ULTRACANVAS_RENDER_SCOPE();

        // Manual style creation (15+ lines)
        TextStyle buttonTextStyle;
        buttonTextStyle.fontFamily = style.fontFamily;
        buttonTextStyle.fontSize = style.fontSize;
        buttonTextStyle.fontWeight = style.fontWeight;
        buttonTextStyle.textColor = GetCurrentTextColor();
        buttonTextStyle.alignment = TextAlign::Center;
        buttonTextStyle.baseline = TextBaseline::Middle;
        SetTextStyle(buttonTextStyle);

        DrawTextInRect(text, bounds);
    }
};
```

**AFTER: Automatic styling**

```cpp
class UltraCanvasButton : public UltraCanvasElement {
    void Render() override {
        ULTRACANVAS_RENDER_SCOPE();

        // Automatic styling (1 line) - gets OS-appropriate "ButtonText" style
        SetTextStyle(TextStyleAuto(this));

        DrawTextInRect(text, GetBounds());
    }
};
```

## ✅ Benefits:

- **90% less code** - One line instead of manual TextStyle construction
- **OS theme compliance** - Automatically uses system fonts and colors
- **Consistency** - All buttons look the same across the application
- **Maintainable** - Change system style once, affects all buttons
- **No manual property mapping** - Framework handles everything

### ✅ Real-World Element Integration

```cpp
// UltraCanvasLabel automatically uses "LabelText"
class UltraCanvasLabel : public UltraCanvasElement {
    void Render() override {
        ULTRACANVAS_RENDER_SCOPE();

        SetTextStyle(TextStyleAuto(this));  // Gets "LabelText" style
        DrawTextInRect(labelText, GetTextArea());
    }
};

// UltraCanvasMenu automatically uses "MenuText"
class UltraCanvasMenu : public UltraCanvasElement {
    void RenderMenuItem(const MenuItemData& item, const Rect2D& itemBounds) {
        SetTextStyle(TextStyleAuto(this));  // Gets "MenuText" style
        DrawTextInRect(item.label, itemBounds);
    }
};

// UltraCanvasTextInput automatically uses "InputText"
class UltraCanvasTextInput : public UltraCanvasElement {
    void RenderText(const Rect2D& area, const Color& color) {
        UltraCanvas::TextStyle autoStyle = TextStyleAuto(this);  // Gets "InputText" style
        autoStyle.textColor = color;  // Override color if needed
        SetTextStyle(autoStyle);

        DrawText(GetDisplayText(), area.GetPosition());
    }
};

// UltraCanvasWindow title rendering
class UltraCanvasWindow : public UltraCanvasBaseWindow {
    void RenderTitleBar() {
        SetTextStyle(TextStyle("WindowTitle"));  // Explicit for window titles
        DrawTextInRect(GetTitle(), GetTitleBarArea());
    }
};
```

## Best Practices

### ✅ Recommended Usage Patterns

cpp

```cpp
// 1. Use Auto styling for standard elements (RECOMMENDED)
class MyButton : public UltraCanvasButton {
    void Render() override {
        ULTRACANVAS_RENDER_SCOPE();

        // Draw button background first
        DrawButton();

        // Use automatic text styling - gets "ButtonText" with OS theme
        SetTextStyle(TextStyleAuto(this));
        DrawTextInRect(GetText(), GetBounds());
    }
};

// 2. Use explicit styles for special cases
class HeaderLabel : public UltraCanvasLabel {
    void Render() override {
        ULTRACANVAS_RENDER_SCOPE();

        // Explicitly use header style instead of default label style
        SetTextStyle(TextStyle("HeaderText"));
        DrawTextInRect(GetText(), GetBounds());
    }
};

// 3. Modify auto style when needed
class ColoredButton : public UltraCanvasButton {
    void Render() override {
        ULTRACANVAS_RENDER_SCOPE();

        DrawButton();

        // Get auto style but override color
        UltraCanvas::TextStyle autoStyle = TextStyleAuto(this);
        autoStyle.textColor = customTextColor;
        SetTextStyle(autoStyle);

        DrawTextInRect(GetText(), GetBounds());
    }
};

// 4. Custom styles for branding
void SetupBrandStyles() {
    UltraCanvas::TextStyle brandStyle;
```

```
    UltraCanvas::TextStyle brandStyle;
    brandStyle.fontFamily = "YourBrandFont";
    brandStyle.fontSize = 18.0f;
    brandStyle.textColor = Color(255, 100, 50, 255);
    ThemeControl::GetInstance().AddTextStyle("BrandTitle", brandStyle, "YourApp");
}

class BrandLabel : public UltraCanvasLabel {
    void Render() override {
        ULTRACANVAS_RENDER_SCOPE();

        SetTextStyle(TextStyle("BrandTitle"));  // Use custom brand style
        DrawTextInRect(GetText(), GetBounds());
    }
};
```

## ❌ Avoid These Patterns

```cpp
// DON'T: Manual font/color setting
class BadButton : public UltraCanvasButton {
    void Render() override {
        SetFont("Arial", 12.0f);          // Hard-coded, no OS theme
        SetTextColor(Color(0, 0, 0, 255)); // Hard-coded, no OS theme
        DrawTextInRect(GetText(), GetBounds());
    }
};

// DON'T: Manual TextStyle construction when Auto works
class BadLabel : public UltraCanvasLabel {
    void Render() override {
        TextStyle manual;
        manual.fontFamily = "Arial";      // Should use OS font
        manual.fontSize = 12.0f;          // Should use OS size
        manual.textColor = Colors::Black;  // Should use OS color
        SetTextStyle(manual);
        DrawTextInRect(GetText(), GetBounds());
    }
};

// DO: Use Auto styling instead
class GoodLabel : public UltraCanvasLabel {
    void Render() override {
        SetTextStyle(TextStyleAuto(this));  // Gets OS-appropriate "LabelText"
        DrawTextInRect(GetText(), GetBounds());
    }
};
```

---

## Theme Control Related System Functions

*These functions are for advanced users and framework developers. Most application programmers won't need these directly.*

### System Management Functions

| Function | Usage Level | Description |
|---|---|---|
| `SetThemeMode()` | Framework | Switch between OS theme and UltraOS mode |
| `RefreshFromOSTheme()` | Framework | Update when OS theme changes |
| `GetSystemStyleNames()` | Debug | List all predefined system styles |
| `GetCustomStyleNames()` | Debug | List custom application styles |
| `RegisterAddCallback()` | Advanced | Monitor when styles are added |
| `RegisterClearCallback()` | Advanced | Monitor when styles are removed |

## Platform Integration

The TextStyle system automatically adapts to each platform:

- **Windows**: Uses `NONCLIENTMETRICS` and `GetSysColor()` for native appearance
- **Linux**: Integrates with GTK themes via `gsettings` and style contexts
- **macOS**: Uses `NSFont` and `NSAppearance` system APIs
- **Result**: Same style names, native appearance on every platform

**Example:** `TextStyle("ButtonText")` returns:

- **Windows**: "Segoe UI 9pt" with Windows system colors
- **Linux**: "Ubuntu Regular 11pt" with GTK theme colors
- **macOS**: "SF Pro Text 13pt" with macOS system colors

---

*This guide covers the essential TextStyle functions for application development. The key principle is: **get a style, apply it with SetTextStyle(), draw your text** - that's it!*