



# Developer Guide

Version 1.4

**.NET Clix**

*Copyright 2011 Thomas W. Holtquist*

[www.skewworks.com](http://www.skewworks.com)

# Table of Contents

<b>1 Introduction</b> .....	<b>4</b>
<b>2 About this Document</b> .....	<b>5</b>
2.1 Change Log.....	5
2.2 Intended Audience.....	8
<b>3 Getting Started</b> .....	<b>9</b>
3.1 Hello World (in forms).....	9
3.2 Multiple Forms.....	11
3.3 LCD Calibration.....	13
<b>4 Graphics Class</b> .....	<b>16</b>
4.1 Graphics Properties.....	16
4.1.1 ActiveContainer .....	16
4.1.2 AvailableDrives.....	16
4.1.3 KeyboardAttached .....	16
4.1.4 KeyboardLayout .....	16
4.1.5 ProductName .....	16
4.1.6 Touching.....	16
4.2 Graphics Methods.....	17
4.2.1 Bounce .....	17
4.2.2 CalibrateScreen.....	17
4.2.3 CalibrateScreen_Blocking .....	17
4.2.4 GetTouchPoint .....	17
4.2.5 ModalBlock .....	17
4.2.6 PauseTouch.....	17
4.2.7 QuietRender.....	17
4.2.8 Grahpics.ReleaseCurrentBlock.....	17
4.2.9 RemoveCurrentBlock .....	18
4.2.10 ResetTouch .....	18
4.2.11 ResumeTouch.....	18
4.2.12 ScreenTransition .....	18

4.3 Graphics Read-Only Variables.....	19
4.3.1 Screen .....	19
4.3.2 ScreenBPP .....	19
4.3.3 ScreenHeight.....	19
4.3.4 ScreenRotation .....	19
4.3.5 ScreenWidth .....	19
<b>5 Helper Classes .....</b>	<b>20</b>
5.1 Colors .....	20
5.2 Dialogs.....	20
5.2.1 Dialogs.AbortModals .....	20
5.2.2 Dialogs.OpenFile .....	20
5.2.3 Dialogs.Prompt.....	20
5.2.4 Dialogs.SaveFile.....	20
5.3 Fonts.....	20
5.4 SettingsManager .....	21
5.5 Strings.....	21
5.5.1 Strings.AbsolutePath.....	21
5.5.2 Strings.NormalizeDirectory .....	21
5.5.3 Strings.RelativePath .....	21
5.5.4 Strings.Replace.....	21
5.6 StringSorter .....	21
5.7 VirtualKeyboard .....	21
5.8 VirtualNumberPad .....	21
<b>6 Controls .....</b>	<b>22</b>
6.1 Control Base Class .....	22
6.2 Container Base Class .....	24
6.3 Checkbox.....	25
6.4 Combobox.....	26
6.5 Command Button.....	27
6.6 ContextMenu .....	28
6.7 Filebox.....	29
6.8 Form .....	30

6.9 Label .....	31
6.10 Listbox .....	32
6.11 MenuItem .....	33
6.12 MenuStrip .....	34
6.13 NavList.....	35
6.14 NumericUpDown.....	36
6.15 Panel.....	37
6.16 PictureBox .....	38
6.17 ProgressBar .....	39
6.18 RadioButton .....	40
6.19 RichTextLabel .....	41
6.20 Scrollbar .....	42
6.21 Slider .....	43
6.22 Tab.....	44
6.23 TabDialog .....	45
6.24 Textbox.....	46
6.25 Treeview.....	47
6.26 TreeviewNode .....	48
6.26 VirtualKeys .....	49

## 1 Introduction

*.NET Clix* is a collection of GUI controls and utilities that allow rapid development of graphical applications using the .NET Micro Framework (NETMF) native graphics. The small footprint, about 131Kb, allows *.NET Clix* to run on nearly any NETMF that supports native graphics.

Since *.NET Clix* supports the NETMF emulator (which is provided free by Microsoft in their Express edition of C#) you can try *.NET Clix* without having to invest in any hardware. The emulator can also be modified to support different screen sizes so you can test multiple pieces of hardware without ever having to purchase them.

## 2 About this Document

### 2.1 Change Log

#### **March 31st, 2012**

Fixed remove child issue with Containers

Added AddItems to Listbox

Added SET ability to Listbox.SelectedItem

Fixed issue where tapping Listbox could set selection outside range

Updated CommandButton to allow multi-line text.

#### **January 8<sup>th</sup>, 2012**

Added TextAlignment property to Label

Fixed sizing issue with PictureBox

Added Press and Release events to controls

Added Style property to CommandButton

Reduced minimum height of Progressbar from 30 to 5

Added GripSize property to Slider

Improved touch handling for Slider

Fixed Height/Width properties for Label

#### **December 26<sup>th</sup>, 2011**

Improved touch handling and added a message queue

Improved touch calibration

Increased calibration timeout from 10 seconds to 30 seconds

Added VirtualKeys control

Merged VirtualNumberPad with VirtualKeyboard

Added Numeric to KeyboardLayouts enumeration

Changed render method for CommandButtons

No longer call GC every render

Fixed render glitch on file dialogs with no directories

Fixed positioning of children after moving a Form

Added Position property to Forms (allowing updating of X & Y in a single call/render)

Added BackgroundScaleMode property to Forms

### **December 6<sup>th</sup>, 2011**

Moved controls & dialogs to Skewwors.NETClix.GUI

Improved caret on multiple line textboxes

Improved scrolling on multiple line textboxes

Added "bounces"

Added "swipes"

Moved VirtualKeyboard into static class

Added AZERTY support to Virtual Keyboard

Added "clear" button to Virtual Keyboard

Updated buttons on Virtual Keyboard to auto-size to screen

Added smart capitalization to Virtual Keyboard

Added top & bottom gradient properties to buttons

Added image to buttons

Removed Input from Graphics (to avoid conflicts)

Exposed Input's methods and events through Graphics directly

Added Virtual Number Pad

Changed progressbar background from image tile to gradient

Changed default font from Seqoe to Verdana (for special character support)

Added KeyboardLayout property to Graphics

**Nov 22<sup>nd</sup>, 2011** Initial release, no changes

## 2.2 Intended Audience

This document is aimed at users with a basic knowledge of C# and the .NET Micro Framework (NETMF). Users should have a working understanding of how to create NETMF applications, deploy to the emulator, and deploy to devices.

## 3 Getting Started

### 3.1 Hello World (in forms)

Adding .NET Clix to a NETMF application is quick and simple. To begin start a new *Console Application* under Visual C# > Micro Framework in Visual Studio.

You can call it ClixHelloWorld for ease of following along. Next add a reference to *Skewworks.NETClix* by right-clicking References in the Solution Explorer and selecting "Add Reference...". Once done, copy and paste the code below. Note no Sleep statement is required to keep the program from exiting.

```
using System;
using Microsoft.SPOT;

using Skewworks.NETClix;

namespace ClixHelloWorld
{
    public class Program
    {
        public static void Main()
        {
            // Create a new form; with default bgcolor
            // Forms automatically occupy the entire screen
            // So there's no need to size it
            // Or even know how big the screen is
            Form frmMain = new Form();

            // We don't need to do anything with this label
            // So we'll add it straight to the form
            frmMain.AddChild(new Label("Hello World!", 4, 4));

            // We're going to add a button as well
            // And we're going to subscribe to an event
            // So we won't add it straight in
            CommandButton cmdPrompt = new CommandButton("Show Prompt", frmMain.Width - 99, frmMain.Height
- 29, 94, 25);

            // Subscribe to the Tap event so we know when
            // the user has pressed the button
            cmdPrompt.Tap += cmdPrompt_Tap;

            // Now we'll add it to the form
            frmMain.AddChild(cmdPrompt);

            // Finally we need to activate the form
            Graphics.ActiveContainer = frmMain;
        }

        private static void cmdPrompt_Tap(object sender, point e)
        {
            // When the button is tapped we want to show a dialog
            // Dialogs block all other actions while they're up
            // This is useful for gathering information
            Dialogs.Prompt("You just tapped a button!", "Hello World!", PromptType.OKOnly);
        }
    }
}
```

Once your code is ready go ahead and press F5 to start your application inside the emulator and you should see a screen like the one below.



Notice how the “Show Prompt” button is nicely at the bottom right of the screen. This is because *.NET Clix* knows the size of your screen automatically and forms take up the full screen by default. In this way we were able to position our button easily.

Go ahead and click it to see what a Prompt dialog looks like.

## 3.2 Multiple Forms

*.NET Clix* can support as many forms as you like, however only one can be **active** at a time. Let's start another project called *HelloMultipleForms* to see exactly how this is handled. Again you'll need to add a reference to *Skewworks.NETClix*.

In this example we'll introduce you to *transitions* as well as multiple forms. *Transitions* are an **optional** way to switch between active forms, so we'll cover switch both with and without them.

Copy the code below and press F5 to see the example in action.

```
using System;
using Microsoft.SPOT;

using Skewworks.NETClix;

namespace HelloMultipleForms
{
    public class Program
    {
        // We need to keep a reference to our first form
        // in order to switch back to it when we're done
        private static Form frmMain;
        private static Form frmPage2; // We'll keep page 2 as well to prevent multiple instancing

        public static void Main()
        {
            // Let's have fun with our first form
            // We'll make it Orange and assign a big font
            // All controls get their font from their parent
            // Unless you provide a new font for them
            frmMain = new Form(Colors.Orange, Fonts.Seqoe12Bold);

            // This will have the form's font
            frmMain.AddChild(new Label("Transition Type:", 4, 4));

            // This will have the font we specify
            Combobox cboTrans = new Combobox(4, 34, frmMain.Width - 8, Fonts.Seqoe10, new string[] {
"None", "Crossfade", "Fade Black", "Fade White", "Slide Down", "Slide Left", "Slide Right", "Slide Up"
});
            cboTrans.SelectedIndex = 1;
            frmMain.AddChild(cboTrans);

            // Transition on button tap
            CommandButton cmdNext = new CommandButton("Continue >", frmMain.Width - 99, frmMain.Height -
29, 94, 25);
            cmdNext.Tap += new OnTap((object sender, point e) => TransScreen(cboTrans.SelectedItem));
            frmMain.AddChild(cmdNext);

            // Activate the form
            Graphics.ActiveContainer = frmMain;
        }

        /// <summary>
        /// Transitions between main form and new screen
        /// </summary>
        /// <param name="TransType">Type of transition</param>
        private static void TransScreen(string TransType)
        {
            // Create a basic form
            if (frmPage2 == null)
            {
                frmPage2 = new Form();
            }
        }
    }
}
```

```

    frmPage2.AddChild(new Label("Look, it's a new form!", 4, 4));

    CommandButton cmdBack = new CommandButton("< Go Back", 4, frmMain.Height - 29, 94, 25);
    cmdBack.Tap += new OnTap((object sender, point e) => Graphics.ActiveContainer = frmMain);
    frmPage2.AddChild(cmdBack);
}

Transitions t = Transitions.None;
switch (TransType.ToLower())
{
    case "crossfade":
        t = Transitions.Crossfade;
        break;
    case "fade black":
        t = Transitions.FadeBlack;
        break;
    case "fade white":
        t = Transitions.FadeWhite;
        break;
    case "slide down":
        t = Transitions.SlideDown;
        break;
    case "slide left":
        t = Transitions.SlideLeft;
        break;
    case "slide right":
        t = Transitions.SlideRight;
        break;
    case "slide up":
        t = Transitions.SlideUp;
        break;
    default:
        Graphics.ActiveContainer = frmPage2;
        return;
}

Graphics.ScreenTransition(frmPage2, t);
}
}
}

```

As you can see subscribing to controls and switching between forms is quite simple; even when using advanced options like *transitions*.

### 3.3 LCD Calibration

Some NETMF boards support multiple screen sizes and so built-in touch calibration may not always be right for the screen you or your client is using. To help with that we've added a tool that will not only calibrate the screen but also save the calibration data using Extended Weak References (EWR).

Create a new project named *CalibrationDemo*, add a reference to *Skewworks.NETClix*, and copy the code below to see how calibration works.

```
using System;
using Microsoft.SPOT;

using Skewworks.NETClix;

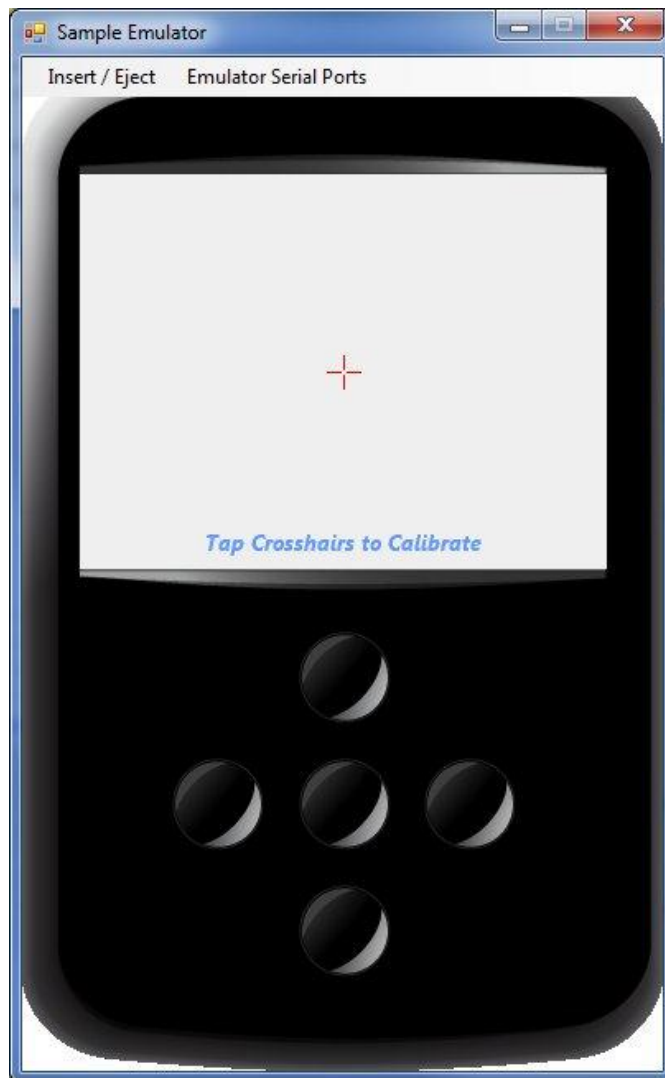
namespace CalibrationDemo
{
    public class Program
    {
        public static void Main()
        {
            if (SettingsManager.LCD == null)
            {
                CalibrateLCD();
                return;
            }
            else if (SettingsManager.LCD.calibrateLCD == ScreenCalibration.Restore)
                SettingsManager.RestoreLCDCalibration();

            Form frmCali = new Form();
            frmCali.AddChild(new Label("Calibration has already been completed and restored.", 4, 4));
            Graphics.ActiveContainer = frmCali;
        }

        /// <summary>
        /// Calibrate the LCD
        /// Displays a form afterwards
        /// </summary>
        private static void CalibrateLCD()
        {
            // Use the blocking method
            // This way we won't show our form before calibration completes
            Graphics.CalibrateScreen_Blocking();

            Form frmDone = new Form();
            frmDone.AddChild(new Label("Calibration complete, re-run code to see what happens.", 4, 4));
            Graphics.ActiveContainer = frmDone;
        }
    }
}
```

After running the code you should see a screen like the one below.



Click all the crosshairs and you'll be brought to a second screen which asks you to confirm your calibration. This helps prevent mis-calibrations.



Once you have finished you can stop and restart the program to see that calibration data has been saved and restored.

## 4 Graphics Class

Graphics is a static class available throughout your application that manages active forms, the touch screen and a few other helpful items.

### 4.1 Graphics Properties

#### 4.1.1 ActiveContainer

This property gets and sets the active container. When you set a new active container it will automatically be rendered to the screen; so don't set it until you're ready for it to be visible.

#### 4.1.2 AvailableDrives

While some manufacturers have their own way of mounting file systems, such as GHI's USB & PersistentStorage classes, all drives should show up in the VolumeInfo collection. This property returns a list of the root of all volumes found in this collection in the form of a string array.

Paths returned by this method will **always** end in a '\ ' character.

#### 4.1.3 KeyboardAttached

Currently this property will always return false.

#### 4.1.4 KeyboardLayout

Gets/Sets the default keyboard layout to use when opening the Virtual Keyboard.

#### 4.1.5 ProductName

An easy way to provide titles for your *dialogs* is to set the product name here and then read it later on. By default this property returns ".NET Clix" until set to another value.

#### 4.1.6 Touching

Returns true when a touch is being received on the screen.

## 4.2 Graphics Methods

### 4.2.1 Bounce

Performs a type of screen transition that stay on the currently active container; this method “bounces” the screen by 20 pixels, in the direction supplied, and then returns it to its original position. This is useful when at the end of scrollable content.

### 4.2.2 CalibrateScreen

This method will begin LCD touch calibration however, it will not block other functions so be sure to stop any other activities while it is running.

### 4.2.3 CalibrateScreen\_Blocking

Performs the same calibration as *CalibrateScreen* but blocks all other actions so you don’t have to worry about overriding its actions.

### 4.2.4 GetTouchPoint

This Input Manager exposed method directly collects touch signals for you in a blocking manner. If called with a timeout, supplied in milliseconds, the method will return at the end of that time whether a touch has been received or not.

### 4.2.5 ModalBlock

Calling this method starts a modal block that will stop all other thread activity (outside of events) until the *ReleaseCurrentBlock* method is called. This method is stackable, so you can have multiple blocks at the same time; which is useful for dialogs.

### 4.2.6 PauseTouch

Pauses touch collection from the screen; this can be useful when managing your own modals but is not recommended in normal use.

### 4.2.7 QuietRender

When calling this method the provided container is quietly rendered to the screen buffer, but not flushed to the screen. This is used to help with *transitions*.

### 4.2.8 Grahpics.ReleaseCurrentBlock

Releases the last block created by calling *ModalBlock*.

#### **4.2.9 RemoveCurrentBlock**

After a modal block has been released it still needs to be removed from the list of blocks in order to prevent it from interfering with other blocks.

#### **4.2.10 ResetTouch**

Resets the touch collection; not recommended for normal use.

#### **4.2.11 ResumeTouch**

If you have previously called `PauseTouch`, you will need to call this method in order to resume normal touch collection.

#### **4.2.12 ScreenTransition**

Instead of calling `ActiveContainer` you can call this method to perform a graphic transition between the currently active screen and the next one you wish to display. Once the transition has completed the supplied screen will automatically be activated.

## **4.3 Graphics Read-Only Variables**

### **4.3.1 Screen**

This is a reference to the buffer bitmap used by containers and controls to render to the screen. You can render directly to this image to make any sort of transitions or effects you may wish to add.

### **4.3.2 ScreenBPP**

Holds the BitsPerPixel of the current LCD.

### **4.3.3 ScreenHeight**

Returns the height of the LCD in pixels.

### **4.3.4 ScreenRotation**

This is the rotation of the screen in degrees, normally the value is 0.

### **4.3.5 ScreenWidth**

Returns the width of the LCD in pixels.

## 5 Helper Classes

### 5.1 Colors

*Colors* is a static class, available throughout the application, which provides a set of predefined colors available for your use. Of course, you can continue to create your own colors with *Microsoft.SPOT.Media.Presentation.ColorUtility*.

### 5.2 Dialogs

The *Dialogs* class is a static class that provides several methods to help you gather information from the user. All *dialogs* are modal and block all other activities until complete.

#### 5.2.1 Dialogs.AbortModals

This method will stop **all** modals currently being displayed. This method exists as an emergency out for errant programs and should not be called in normal operations.

#### 5.2.2 Dialogs.OpenFile

*OpenFile* is an overloaded method that provides a file selection dialog to the user. You can supply any or even none of the following parameters: Title, Start Directory, and Extensions. If the user cancels the dialog this method will return an empty string, otherwise it will contain the path to the file the user selected.

#### 5.2.3 Dialogs.Prompt

Probably the most frequently used *dialog*, *Prompt* displays a message to the user and waits for them to respond. There are 4 response values available: OK, Cancel, Yes, and No.

#### 5.2.4 Dialogs.SaveFile

Similar to *OpenFile*, this method allows the user to select a place to *save* a file. The user is automatically prompted with a confirmation dialog if they attempt to overwrite an existing file, unless otherwise specified when calling this method.

### 5.3 Fonts

*Fonts* is another static class containing predefined values for your use. In this case it is several fonts. Each font is tied to a property and is not loaded until it is called.

## 5.4 SettingsManager

The *Settings Manager* is used to save and restore touch calibration data from EWR.

## 5.5 Strings

The static *Strings* class contains several helpful methods for working with strings.

### 5.5.1 Strings.AbsolutePath

Takes a base directory and a relative path and returns an absolute path based on the supplied variables.

### 5.5.2 Strings.NormalizeDirectory

Ensures that the last character is a '\ ' character.

### 5.5.3 Strings.RelativePath

Takes a base directory and target path or filename and returns a relative path to that target.

### 5.5.4 Strings.Replace

This method extends the *Replace* method available in NETMF allowing you to replace entire strings instead of just characters.

## 5.6 StringSorter

This class will sort strings into an array. It is capable of being supplied a starting array, adding items dynamically, and sorting ascending, descending, case sensitive or case insensitive.

## 5.7 VirtualKeyboard

The VirtualKeyboard allows you to collect text information from the user without attaching an external keyboard. Currently VirtualKeyboard support QWERTY (default for text controls), AZERTY layouts, and Numeric (default for numeric controls).

## 5.8 VirtualNumberPad

This class has been merged with VirtualKeyboard; please update code accordingly.

## 6 Controls

Controls in *.NET Clix* all inherit from a base class of either Container or Control.

### 6.1 Control Base Class

The Control class contains all of the methods needed to render and use a control inside of *.NET Clix*, which allows you to create your own custom controls.

#### Events

Name	Description
<b>DoubleTap</b>	This event is fired when the user taps the control twice within a set time frame; ~0.5 seconds
<b>GotFocus</b>	GotFocus is raised when a user taps the control and it is given focus from another control. This event will not fire again until after the control has lost and then regained focus
<b>Lost Focus</b>	If a control has focus and a user taps on another control this event will raise alerting you that the control has lost its focus
<b>Pressed</b>	Fired when finger/stylus press down on control
<b>Released</b>	Fired when finger/stylus is removed from the screen
<b>Tap</b>	Generally this event is raised every time a user taps a control. Some controls have exceptions to this such as when a user taps the up or down arrow on a NumericUpDown control
<b>TapHold</b>	If a user taps a control and does not lift their stylus/finger for more than ~0.5 seconds this event is fired

#### Properties

All controls have 2 sets of coordinates: literal and relative. The literal coordinates are the exact position on the screen and are handled by ScreenBounds, Left, and Top. The relative coordinates are the coordinates of a control inside of its parent and handled by Bounds, X, and Y.

Name	Description
<b>Bounds</b>	The X, Y, Width and Height of a control inside of its parent
<b>Enabled</b>	Controls should not respond to taps or events when this property is set to false
<b>Height</b>	The height of a control
<b>Left</b>	The literal X coordinate on the screen of a control
<b>Parent</b>	This property gets/sets the parent of a control. This applies to all controls <b>except</b> the Form control which does not have a parent
<b>PenDown</b>	Returns true when the pen is down inside of the control
<b>ScreenBounds</b>	The X, Y, Width and Height of a control literally positioned on the screen
<b>Suspended</b>	While this is true the control should respond to all commands and events but should not draw to the screen

Name	Description
<b>Tag</b>	Optional object tag assigned to any control
<b>Top</b>	The literal Y position of a control on the screen
<b>TopLevelContainer</b>	Containers allow a control to be multiple levels down form a Form; this property will always return the top-most parent, which should always be a Form
<b>Visible</b>	Controls with their visible property set to false should never render and will not receive touch information
<b>Width</b>	The width of a control
<b>X</b>	The relative X position of a control inside its parent
<b>Y</b>	The relative Y position of a control inside its parent

## Methods

Each control that inherits from the Control class will have the following methods available.

Name	Description
<b>Blur</b>	This method is generally only called by the control's parent and lets the control know it has lost focus
<b>Focus</b>	Also generally called only by the parent this method informs the control that it has been focused
<b>Flush</b>	The Flush method is generally reserved and when called will cause the screen buffer to flush the control's area to the LCD. Note if called directly the flush could contain outdated information
<b>HitTest</b>	Returns true if the pen is down inside the control
<b>Render</b>	This method caused the control to draw itself onto the screen buffer. If the optional parameter "flush" is set to true that area will immediately be flushed to the screen
<b>SetOffset</b>	Reserved. This should never be called except for by a Container
<b>TouchDown</b>	Calling this method will force the control to think it has received a TouchDown message from the system
<b>TouchMove</b>	Forces the control to believe it has received a TouchMove message from the system
<b>TouchUp</b>	Forces the control to think it has received a TouchUp message from the system

## 6.2 Container Base Class

The Container class inherits from the Control class and therefore has all of the properties, events and methods as a control. Additionally it provides methods for containing children.

### Events

Name	Description
<b>Children</b>	Read-Only: provides an array list of the children contained
<b>Font</b>	This property is <b>important</b> as any text based control that does <i>not</i> have a font property set will attempt to inherit its font from the containing parent
<b>InstancingDomain</b>	This method is useful when working across domains, it returns the name of the domain the container was instanced inside of

### Methods

Each control that inherits from the Container class will have the following methods available.

Name	Description
<b>AddChild</b>	Adds a child to the container
<b>ClearChildren</b>	Removes all children from a container
<b>GetChildAt</b>	Returns a child at the given point inside the array list
<b>RemoveChild</b>	Removes a child by its reference
<b>RemoveChildAt</b>	Removes a child at the given point inside the array list

## 6.3 Checkbox

**Inherits** Control  
**Description** Provides a simple Boolean response from the user by way of the Value property.  
**Added Events** *None*

### Constructors

Checkbox(**int** x, **int** y)  
Checkbox(**int** x, **int** y, **bool** selected)

### Properties

Name	Description
<b>Height</b>	Read-Only; checkboxes must always be 17x17
<b>Width</b>	Read-Only; checkboxes must always be 17x17
<b>Value</b>	Returns true when the checkbox is checked. Updating this value will cause the control to render itself

### Events

## 6.4 Combobox

<b>Inherits</b>	Control
<b>Description</b>	Allows the user to select a single item from a list of items. When tapped a modal window will be displayed allowing the user to select from the options listed.
<b>Added Events</b>	SelectedIndexChanged

### Constructors

```
Combobox(int X, int Y, int Width)
Combobox(int X, int Y, int Width, Font Font)
Combobox(int X, int Y, int Width, string[] Items)
Combobox(int X, int Y, int Width, Font Font, string[] Items)
```

### Properties

Name	Description
<b>Font</b>	The font used to render text inside the Combobox
<b>ForeColor</b>	Color to use when rendering text
<b>GradientBottom</b>	Second color to use when drawing the gradient background
<b>GradientTop</b>	First color to use when drawing the gradient background
<b>Height</b>	Read-Only: height is determined by the size of the font
<b>Length</b>	Read-Only: returns the number of items inside the Combobox
<b>SelectedIndex</b>	Gets/Sets the selected index
<b>SelectedItem</b>	Read-Only: returns the string value of the selected item

### Methods

Name	Description
<b>AddItem</b>	Adds a single string to the end of the list
<b>AddItems</b>	Adds an array of strings to the end of the list
<b>Clear</b>	Clear all items from the list
<b>Item</b>	Returns a single item at a given point
<b>RemoveItem</b>	Removes a value by its reference
<b>RemoveItemAt</b>	Removes a value from a specified point in the list
<b>Sort</b>	Uses string sorter to sort the list (Ascending, Case-Sensitive)

## 6.5 Command Button

<b>Inherits</b>	Control
<b>Description</b>	Provides a simple tap interface for the user
<b>Added Events</b>	<i>None</i>

### Constructors

```
CommandButton(string text, int x, int y)
CommandButton(string text, Font font, int x, int y)
CommandButton(string text, int x, int y, int width, int height)
CommandButton(string text, Font font, Color ForeColor, int x, int y, int width, int height)
```

### Properties

Name	Description
<b>Color</b>	Color to use when rendering text
<b>Font</b>	The font used to render text
<b>GradientBottom</b>	Second color to use when drawing the gradient background
<b>GradientTop</b>	First color to use when drawing the gradient background
<b>Image</b>	Image to render on the CommandButton; images are centered
<b>Style</b>	When set to Default the CommandButton will render with a gradient; default. When set to Flat CommandButton will render solid with the GradientTop color normally and with the GradientBottom color when pressed.
<b>Text</b>	Text to display

## 6.6 ContextMenu

**Inherits** MenuOwner, Control  
**Description** Provides a pop-up context menu for the user  
**Added Events** *None*

### Constructors

ContextMenu(**Font** Font)  
ContextMenu(**MenuItem**[] MenuItems, **Font** Font)  
ContextMenu(**string**[] Items, **Font** Font)

### Properties

Name	Description
<b>Font</b>	The font used to render text
<b>Parent</b>	Top-level container to display the ContextMenu on
<b>Height</b>	Read-Only
<b>Items</b>	Gets/Sets array of MenuItem's inside the ContextMenu

### Methods

Name	Description
<b>Add</b>	Adds a MenuItem to the ContextMenu
<b>Clear</b>	Removes all MenuItem's
<b>Remove</b>	Removes a specified MenuItem
<b>RemoveAt</b>	Removes a MenuItem at a specified point in the array

## 6.7 Filebox

<b>Inherits</b>	Control
<b>Description</b>	Displays a list of folders and files at a specified point of a volume
<b>Added Events</b>	SelectedFileChanged

### Constructors

```
Filebox(string path, int x, int y, int width, int height)  
Filebox(string path, string[] extensions, int x, int y, int width, int height)
```

### Properties

Name	Description
<b>Height</b>	Minimum of 30 pixels is required and enforced
<b>Path</b>	The path inside the volume to display files and folders for
<b>SelectedFile</b>	Full path to the selection
<b>SelectionIsFile</b>	Returns false when the selected item is a folder

### Methods

Name	Description
<b>Refresh</b>	Refreshes the list of files and folders

## 6.8 Form

<b>Inherits</b>	Control
<b>Description</b>	This is the main container used inside of <i>.NET Clix</i> ; all other controls and containers should be placed inside of a Form.
<b>Added Events</b>	<i>None</i>

### Constructors

```
Form()  
Form(Color BackColor)  
Form(Color BackColor, Font Font)  
Form(Color BackColor, Font Font, int X, int Y, int Width, int Height)  
Form(Color BackColor, Font Font, int X, int Y, int Width, int Height, WindowType Type)
```

### Properties

Name	Description
<b>ActiveControl</b>	Gets/Sets the currently focused control on the Form
<b>AutoScroll</b>	When true the form will automatically provide scrollbars when controls take up more room than the size of the form. False by default
<b>BackColor</b>	The background color of the Form; LightGray by default
<b>BackgroundImage</b>	The image to display on the background of the Form; null by default
<b>BackgroundScaleMode</b>	Determines how the background image is rendered on the form
<b>Position</b>	Allows setting of the Form's X & Y positions in a single call

### Methods

Name	Description
<b>AddChild</b>	Adds Control or Container to the Form
<b>ClearChildren</b>	Removes all children
<b>Dispose</b>	Disposes of the form
<b>GetChildAt</b>	Returns a child at a specific point of the array
<b>RemoveChild</b>	Removes a specified child
<b>RemoveChildAt</b>	Removes a child from a specified point in the array
<b>ShowPopup</b>	Displays a specified ContextMenu at a specified point on the From

## 6.9 Label

<b>Inherits</b>	Control
<b>Description</b>	Provides read-only text display to the user
<b>Added Events</b>	<i>None</i>

### Constructors

```
Label(string text, int x, int y, bool TransparentBackground = true)
Label(string text, Font font, int x, int y, bool TransparentBackground = true)
Label(string text, int x, int y, int width, int height, bool TransparentBackground = true)
Label(string text, Font font, int x, int y, int width, int height, bool TransparentBackground = true)
Label(string text, Font font, Color ForeColor, int x, int y, int width, int height, bool TransparentBackground = true)
Label(string text, Color ForeColor, int x, int y, int width, int height, bool TransparentBackground = true)
```

### Properties

Name	Description
<b>AutoSize</b>	When this is set to true the label will automatically adjust its width to fit the text. <b>Note</b> in this mode the label's parent will need to re-render every time text is changed
<b>BackColor</b>	BackColor used when rendering the label; LightGray by default. This is ignored if TransparentBackground is set to true
<b>Color</b>	Color used when rendering the text; Charcoal by default
<b>Font</b>	Font used when rendering text
<b>Text</b>	Text to display
<b>TextAlignment</b>	Sets the alignment of text rendered on the control; left by default.
<b>TransparentBackground</b>	When this is set to true the background color will not be drawn

To avoid the parent re-rendering itself and all children you will need to set TransparentBackground and AutoSize to false.

## 6.10 Listbox

<b>Inherits</b>	Control
<b>Description</b>	Provides selectable list of items
<b>Added Events</b>	SelectedIndexChanged

### Constructors

```
Listbox(int x, int y, int width, int height, bool RadioSelect = true)
Listbox(int x, int y, int width, int height, string[] items, bool RadioSelect = true)
Listbox(int x, int y, int width, int height, Font Font, bool RadioSelect = true)
Listbox(int x, int y, int width, int height, string[] items, Font Font, bool RadioSelect = true)
```

### Properties

Name	Description
<b>Font</b>	Font used when rendering items
<b>Height</b>	Minimum requirement of 30 pixels to allow for scrollbars
<b>Length</b>	Number of items in list
<b>RadioSelect</b>	When set to true RadioButtons will be present on the left side of the text to indicate selection. When set to false a gradient background will denote the selected item
<b>SelectedIndex</b>	Index of the selected item
<b>SelectedItem</b>	String value of the selected item

### Methods

Name	Description
<b>AddItem</b>	Adds a string to the list
<b>Clear</b>	Removes all items from the list
<b>InsertAt</b>	Inserts a string at a specific point in the list
<b>RemoveItem</b>	Removes a specified item
<b>RemoveItemAt</b>	Removes an item at the specified index
<b>Sort</b>	Uses string sorter to sort the list (Ascending, Case-Sensitive)

## 6.11 MenuItem

**Inherits** Control  
**Description** Provides a clickable item inside of a ContextMenu or MenuStrip  
**Added Events** None

### Constructors

MenuItem()  
MenuItem(string Text)

### Properties

Name	Description
<b>Expanded</b>	When true all sub-items will be displayed
<b>Height</b>	Read-Only: based on the font used and the content of the MenuItem
<b>Items</b>	Array of MenuItems this MenuItem contains
<b>Text</b>	Text to display

### Methods

Name	Description
<b>Add</b>	Adds a MenuItem to the MenuItem's list of children
<b>Clear</b>	Removes all child MenuItems
<b>Remove</b>	Removes a specified MenuItem
<b>RemoveAt</b>	Removes a MenuItem at a specific point in the array

## 6.12 MenuStrip

<b>Inherits</b>	MenuOwner, Control
<b>Description</b>	Top-level container for MenuItems.
<b>Added Events</b>	<i>None</i>

### Constructors

```
MenuStrip()  
MenuStrip(Font Font)  
MenuStrip(int X, int Y, Font Font)  
MenuStrip(MenuItem[] MenuItems)  
MenuStrip(MenuItem[] MenuItems, Font Font)  
MenuStrip(int X, int Y, MenuItem[] MenuItems, Font Font)
```

### Properties

Name	Description
<b>Font</b>	Font to use when rendering
<b>Height</b>	Read-Only: based on the font used and the content of the MenuItems contained
<b>Items</b>	Array of MenuItems this MenuStrip contains

### Methods

Name	Description
<b>Add</b>	Adds a MenuItem to the MenuStrip's list of children
<b>Clear</b>	Removes all child MenuItems
<b>Remove</b>	Removes a specified MenuItem
<b>RemoveAt</b>	Removes a MenuItem at a specific point in the array

## 6.13 NavList

<b>Inherits</b>	Control
<b>Description</b>	Presents a list similar to Listbox, but intended for navigational use and does not support RadioButtons or scrolling.
<b>Added Events</b>	SelectedIndexChanged

### Constructors

```
NavList(int x, int y, int width, int height)
NavList(int x, int y, int width, int height, string[] items)
NavList(int x, int y, int width, int height, Font Font)
NavList(int x, int y, int width, int height, string[] items, Font Font)
```

### Properties

Name	Description
<b>Font</b>	Font to use when rendering
<b>Length</b>	Number of items in list
<b>SelectedIndex</b>	Index of the currently selected item
<b>SelectedItem</b>	Value of the currently selected item

### Methods

Name	Description
<b>AddItem</b>	Adds an item to the list
<b>Clear</b>	Removes all items
<b>RemoveItem</b>	Removes a specified item
<b>RemoveItemAt</b>	Removes an item at a specific point in the array
<b>Sort</b>	Uses string sorter to sort the list (Ascending, Case-Sensitive)

## 6.14 NumericUpDown

<b>Inherits</b>	MenuOwner, Control
<b>Description</b>	Presents a simple way to get a numeric value from the user within a specified range
<b>Added Events</b>	OnValueChanged

### Constructors

```
NumericUpDown(int X, int Y, int Width)
NumericUpDown(int X, int Y, int Width, Font Font)
NumericUpDown(int X, int Y, int Width, int Minimum, int Maximum)
NumericUpDown(int X, int Y, int Width, int Minimum, int Maximum, int Value)
NumericUpDown(int X, int Y, int Width, int Minimum, int Maximum, Font Font)
NumericUpDown(int X, int Y, int Width, int Minimum, int Maximum, int Value, Font Font)
```

### Properties

Name	Description
<b>Font</b>	Font to use when rendering
<b>ForceZeros</b>	When set to true this will make sure that the length of the value displayed always matches the length of the maximum value by appending zeros to the left of the value
<b>Minimum</b>	Minimum allowed value
<b>Maximum</b>	Maximum allowed value
<b>Value</b>	Currently selected value

**Note:** When Graphics.KeyboardAttached is set to false tapping this control will automatically bring up the VirtualNumberPad.

## 6.15 Panel

**Inherits** Container  
**Description** Provides a container to group other controls within.  
**Added Events** *None*

### Constructors

```
Panel(int x, int y, int width, int height)  
Panel(int x, int y, int width, int height, Color background)
```

### Properties

Name	Description
<b>Background</b>	Background color to use when rendering

### Methods

Name	Description
<b>AddChild</b>	Adds Control or Container
<b>ClearChildren</b>	Removes all children
<b>GetChildAt</b>	Returns a child at a specific point of the array
<b>RemoveChild</b>	Removes a specified child
<b>RemoveChildAt</b>	Removes a child from a specified point in the array

## 6.16 PictureBox

**Inherits** Control  
**Description** Displays an image  
**Added Events** *None*

### Constructors

```
PictureBox(Bitmap image, int x, int y)  
PictureBox(Bitmap image, int x, int y, int width, int height)  
PictureBox(Bitmap image, int x, int y, int width, int height, BorderStyle border,  
ScaleMode scale)
```

### Properties

Name	Description
<b>AutoSize</b>	When true the PictureBox will automatically be sized to fit the image
<b>Background</b>	Background color for the PictureBox
<b>ScaleMode</b>	Determines how(if at all) to scale the image displayed
<b>BorderStyle</b>	Type of border (if any) to display around the PictureBox
<b>Image</b>	Image to display

## 6.17 Progressbar

**Inherits** Control  
**Description** Visual display of progress  
**Added Events** *None*

### Constructors

Progressbar(`int` x, `int` y, `int` width)  
Progressbar(`int` x, `int` y, `int` width, `int` minvalue, `int` maxvalue, `int` value)

### Properties

Name	Description
<b>GradientBottom</b>	Second color to use when drawing progress gradient
<b>GradientTop</b>	First color to use when drawing progress gradient
<b>Minimum</b>	Minimum allowed value
<b>Maximum</b>	Maximum allowed value
<b>Value</b>	Current value

## 6.18 RadioButton

**Inherits** Control  
**Description** Selection from a set of values  
**Added Events** *None*

### Constructors

```
RadioButton(int x, int y)  
RadioButton(int x, int y, Color BackColor)  
RadioButton(int x, int y, string group)  
RadioButton(int x, int y, string group, Color BackColor)  
RadioButton(int x, int y, bool selected)  
RadioButton(int x, int y, bool selected, Color BackColor)  
RadioButton(int x, int y, string group, bool selected)  
RadioButton(int x, int y, string group, bool selected, Color BackColor)
```

### Properties

Name	Description
<b>Group</b>	The name of the group this RadioButton belongs to. When a RadioButton's value is set to true all other RadioButtons inside its group will have their values set to false. "defaultGroup" by default
<b>Value</b>	Current value

**Note** if a backcolor is not supplied at construction the RadioButton will have a transparent background. This cannot be changed at runtime. It is recommended to use a solid background since the RadioButton uses alpha-blended pixels and will create a buffer for fast rendering when a solid background is selected.

## 6.19 RichTextLabel

<b>Inherits</b>	Control
<b>Description</b>	Provides rich read-only text display to the user. This control allows <b>, <i>, <u>, <color>, and <font> to change the font being used.
<b>Added Events</b>	None

### Constructors

```
RichTextLabel(string text, int x, int y)
RichTextLabel(string text, int x, int y, int width, int height)
RichTextLabel(string text, Color ForeColor, int x, int y, int width, int height)
RichTextLabel(string text, Color ForeColor, string FontName, int x, int y, int width, int height)
RichTextLabel(string text, Color ForeColor, string FontName, int x, int y, int width, int height, bool enabled, bool visible)
```

### Properties

Name	Description
<b>AutoSize</b>	When this is set to true the label will automatically adjust its width to fit the text. <b>Note</b> in this mode the label's parent will need to re-render every time text is changed
<b>Color</b>	Color used when rendering the text; Charcoal by default
<b>FontName</b>	Name of the font to use (initially) when rendering; "Verdana10" by default
<b>Text</b>	Text to display

RichTextLabels *always* cause parent to re-render on text change and do not support solid backgrounds.

### Example usage:

```
rtl.Text = "<b>Bold</b>, <i>Italic</i>, <u>Underline</u>, <color '255,0,0'>Red</color>, <font 'Verdana8BoldItalic'>Verdana 8, bold and italic</font>";
```

## 6.20 Scrollbar

<b>Inherits</b>	Control
<b>Description</b>	Used for scrolling content
<b>Added Events</b>	ValueChanged

### Constructors

```
Scrollbar(int x, int y, int size, Orientation orientation)
Scrollbar(int x, int y, int size, Orientation orientation, int Minimum, int Maximum)
Scrollbar(int x, int y, int size, Orientation orientation, int Minimum, int Maximum, int value)
```

### Properties

Name	Description
<b>AutoRepeating</b>	When true if the user taps and holds the up/down/left/right arrow the scrollbar will continue to adjust its value until released; false by default
<b>Height</b>	Read-only
<b>LargeChange</b>	Amount to change value when tapping the scrollbar outside of the arrows or grip
<b>Minimum</b>	Minimum value
<b>Maximum</b>	Maximum value
<b>SmallChange</b>	Amount to change value when tapping an arrow
<b>Size</b>	When orientation is set to Horizontal size controls with Width, otherwise it controls the Height.
<b>Value</b>	Current value
<b>Width</b>	Read-only

## 6.21 Slider

<b>Inherits</b>	Control
<b>Description</b>	Similar to the Scrollbar the Slider allows the user to select a value by sliding a grip. However arrows are not provided.
<b>Added Events</b>	ValueChanged

### Constructors

```
Slider(int x, int y, int size, Orientation orientation)
Slider(int x, int y, int size, Orientation orientation, int Minimum, int Maximum)
Slider(int x, int y, int size, Orientation orientation, int Minimum, int Maximum, int value)
```

### Properties

Name	Description
<b>GripSize</b>	Determines the size of the grip to render; 12 by default
<b>Height</b>	Read-only
<b>Minimum</b>	Minimum value
<b>Maximum</b>	Maximum value
<b>Value</b>	Current value
<b>Width</b>	Read-only

## 6.22 Tab

**Inherits** Container  
**Description** Provides a container to group other controls within a TabDialog.  
**Added Events** *None*

### Constructors

Tab(`string` title)  
Tab(`string` title, `Color` background)

### Properties

Name	Description
<b>Background</b>	Background color to use when rendering
<b>Title</b>	Title to display on TabDialog

### Methods

Name	Description
<b>AddChild</b>	Adds Control or Container
<b>ClearChildren</b>	Removes all children
<b>GetChildAt</b>	Returns a child at a specific point of the array
<b>RemoveChild</b>	Removes a specified child
<b>RemoveChildAt</b>	Removes a child from a specified point in the array

## 6.23 TabDialog

**Inherits** Container  
**Description** Provides a tabbed interface for controls  
**Added Events** *None*

### Constructors

Tab(`string` title)  
Tab(`string` title, `Color` background)

### Properties

*None*

### Methods

Name	Description
<b>AddTab</b>	Adds a tab
<b>RemoveChild</b>	Removes a specified Tab
<b>RemoveChildAt</b>	Removes a Tab from a specified point in the array

## 6.24 Textbox

<b>Inherits</b>	Control
<b>Description</b>	Collects text input from the user
<b>Added Events</b>	TextChanged, VirtualKeyboardClosed

### Constructors

```
Textbox(string Text, int X, int Y, int Width, int Height)
Textbox(string Text, int X, int Y, int Width, int Height, bool Multiline)
Textbox(string Text, int X, int Y, int Width, int Height, Font Font)
Textbox(string Text, int X, int Y, int Width, int Height, bool Multiline, Font Font)
```

### Properties

Name	Description
<b>Caret</b>	Gets/Sets current location of the caret
<b>Color</b>	Color used when rendering font
<b>Font</b>	Font to use during render
<b>Height</b>	Minimum of Font.Height + 4 or 40 when multiline is true
<b>Multiline</b>	When true the textbox will allow multiple lines and scrolling
<b>PasswordChar</b>	When not ' ' the character supplied will be used to hide the text entered.
<b>ReadOnly</b>	When true the text will not be allowed to be edited by the user
<b>ShowCaret</b>	Reserved for use by VirtualKeyboard
<b>Text</b>	Current text to display

**Note:** When Graphics.KeyboardAttached is set to false tapping this control will automatically bring up the VirtualKeyboard.

## 6.25 Treeview

<b>Inherits</b>	Control
<b>Description</b>	Displays a collapsible/expandable list of nodes for the user
<b>Added Events</b>	NodeTapped, NodeCollapsed, NodeExpanded

### Constructors

Treeview(**int** x, **int** y, **int** width, **int** height)

### Properties

Name	Description
<b>Height</b>	Minimum of 30 pixels
<b>Length</b>	Number of top-level nodes
<b>SelectedNode</b>	Currently selected node

### Methods

Name	Description
<b>AddNode</b>	Adds a TreeviewNode
<b>ClearNodes</b>	Removes all nodes
<b>Node</b>	Returns a node at a specified point in the list
<b>RemoveNode</b>	Removes specified node
<b>RemoveNodeAt</b>	Removes a node at the specified index

## 6.26 TreeviewNode

<b>Inherits</b>	<i>None</i>
<b>Description</b>	Displays a single node inside the Treeview
<b>Added Events</b>	NodeTapped, NodeCollapsed, NodeExpanded

### Constructors

TreeviewNode([string](#) Text)  
TreeviewNode([string](#) Text, [object](#) Tag)

### Properties

Name	Description
<b>Bounds</b>	The X, Y, Width and Height of a control inside of its parent
<b>Expanded</b>	When true sub-nodes will be rendered
<b>Length</b>	Number of sub-nodes
<b>Parent</b>	This property gets/sets the parent of a control. This applies to all controls <b>except</b> the Form control which does not have a parent
<b>PenDown</b>	Returns true when the pen is down inside of the control
<b>Selected</b>	True when selected by the Treeview
<b>Tag</b>	Optional object tag assigned to any control
<b>Text</b>	Text to display

### Methods

Name	Description
<b>AddNode</b>	Adds a TreeviewNode
<b>ClearNodes</b>	Removes all nodes
<b>Node</b>	Returns a node at a specified point in the list
<b>RemoveNode</b>	Removes specified node
<b>RemoveNodeAt</b>	Removes a node at the specified index

## 6.26 VirtualKeys

**Inherits** Control  
**Description** Displays a set of Virtual Keys for gathering input from the user  
**Added Events** None

### Constructors

```
VirtualKeys(int X, int Y, int Width, int Height, Font Font, bool Multiline = false,  
string Text = null, string Title = null, KeyboardLayout Layout = KeyboardLayout.QWERTY,  
char PasswordCharacter = ' ')
```

### Properties

Name	Description
BackColor	Gets/Sets the background color of the control
Text	Text entered/displayed

### Methods

None