App Inventor - Basic Components

1.Button

Buttons are components that users touch to perform some action in your app.

Buttons detect when users tap them. Many aspects of a button's appearance can be changed. You can use the Enabled property to choose whether a button can be tapped.

Properties:

BackgroundColor:
    Color for button background.

Enabled:
    If set, user can tap button to cause action.

FontBold:
    If set, button text is displayed in bold.

FontItalic:
    If set, button text is displayed in italics.

FontSize:
    Point size for button text.

FontTypeface:
    Font family for button text.

Height:
    Button height (y-size).

Width:
    Button width (x-size).

Image:
    Image to display on button.

Shape (designer only):

Specifies the button's shape (default, rounded, rectangular, oval). The shape will not be visible if an Image is being displayed.

ShowFeedback:

Specifies if a visual feedback should be shown for a button that has an image as background.

Text:

Text to display on button.

TextAlignment:

Left, center, or right.

TextColor:

Color for button text.

Visible:

Specifies whether the component should be visible on the screen. Value is true if the component is showing and false if hidden.

Events:

Click():

Indicates a user has clicked on the button.

GotFocus():

Indicates the cursor moved over the button so it is now possible to click it.

LongClick():

Indicates a user has long clicked on the button.

LostFocus():

Indicates the cursor moved away from the button so it is now no longer possible to click it.

Methods:

None

2.Canvas

A two-dimensional touch-sensitive rectangular panel on which drawing can be done and sprites can be moved.

The **BackgroundColor**, **PaintColor**, **BackgroundImage**, **Width**, and **Height** of the Canvas can be set in either the Designer or in the Blocks Editor. The **Width** and **Height** are measured in pixels and must be positive.

Any location on the Canvas can be specified as a pair of (X, Y) values, where

- X is the number of pixels away from the left edge of the Canvas
- Y is the number of pixels away from the top edge of the Canvas

There are events to tell when and where a Canvas has been touched or a Sprite (**ImageSprite** or **Ball**) has been dragged. There are also methods for drawing points, lines, and circles.

Properties:

BackgroundColor:
The color of the canvas background.

BackgroundImage:
The name of a file containing the background image for the canvas.

FontSize:
The font size of text drawn on the canvas.

Height:
The height of the canvas.

LineWidth:
The width of lines drawn on the canvas.

PaintColor:
The color in which lines are drawn

TextAlignment (designer only):

Visible:

Specifies whether the component should be visible on the screen. Value is true if the component is showing and false if hidden.

Width:

The width of the canvas.

Events:

Dragged (number startX, number startY, number prevX, number prevY, number currentX, number currentY, boolean draggedSprite):

When the user does a drag from one point (prevX, prevY) to another (x, y). The pair (startX, startY) indicates where the user first touched the screen, and "draggedSprite" indicates whether a sprite is being dragged.

Flung (number x, number y, number speed, number heading, number xvel, number yvel, boolean flungSprite):

When a fling gesture (quick swipe) is made on the canvas: provides the (x,y) position of the start of the fling, relative to the upper left of the canvas. Also provides the speed (pixels per millisecond) and heading (0-360 degrees) of the fling, as well as the x velocity and y velocity components of the fling's vector. The value "flungSprite" is true if a sprite was located near the the starting point of the fling gesture.

TouchDown (number x, number y):

When the user begins touching the canvas (places finger on canvas and leaves it there): provides the (x,y) position of the touch, relative to the upper left of the canvas

TouchUp(number x, number y):

When the user stops touching the canvas (lifts finger after a TouchDown event): provides the (x,y) position of the touch, relative to the upper left of the canvas

Touched(number x, number y, boolean touchedSprite):

When the user touches the canvas and then immediately lifts finger: provides the (x,y) position of the touch, relative to the upper left of the canvas. TouchedSprite is true if the same touch also touched a sprite, and false otherwise.

Methods:

Clear():

Clears anything drawn on this Canvas but not any background color or image.

DrawCircle(number x, number y, number r):

Draws a circle (filled in) at the given coordinates on the canvas, with the given radius.

DrawLine(number x1, number y1, number x2, number y2):

Draws a line between the given coordinates on the canvas.

DrawPoint(number x, number y):

Draws a point at the given coordinates on the canvas.

DrawText(text text, number x, number y):

Draws the specified text relative to the specified coordinates using the values of the FontSize and TextAlignment properties.

DrawTextAtAngle(text text, number x, number y, number angle):

Draws the specified text starting at the specified coordinates at the specified angle using the values of the FontSize and TextAlignment properties.

number GetBackgroundPixelColor(number x, number y):

Gets the color of the specified point. This includes the background and any drawn points, lines, or circles but not sprites.

number GetPixelColor(number x, number y):

Gets the color of the specified point.

text Save():

Saves a picture of this Canvas to the device's external storage. If an error occurs, the Screen's ErrorOccurred event will be called.

text SaveAs(text fileName):

Saves a picture of this Canvas to the device's external storage in the file named fileName. fileName must end with one of ".jpg", ".jpeg", or ".png" (which determines the file type: JPEG, or PNG).

SetBackgroundPixelColor(number x, number y, number color):

Sets the color of the specified point. This differs from DrawPoint by having an argument for color.

---

3. CheckBox

Check box components can detect user taps and can change their boolean state in response.

A check box component raises an even when the user taps it. There are many properties affecting its appearance that can be set in the Designer or Blocks Editor.

Properties:

BackgroundColor:

Color for check box background.

Checked:

True if the box is checked, false otherwise.

Enabled:

If set, user can tap check box to cause action.

FontBold (designer only):
FontItalic (designer only):
FontSize:
FontTypeface (designer only):
Height:

Check box height (y-size).

Text:

Text to display on check box.

TextColor:

Color for check box text.

Visible:

If set, check box is visible.

Width:

Check box width (x-size).

Events:

Changed():

User tapped and released check box.

GotFocus():

Check box became the focused component.

LostFocus():

Check box stopped being the focused component.

Methods:

None

---

4. Clock

Use a clock component to create a time that signals events at regular intervals. The clock component also does various conversions and manipulations with time units.

One use of the clock component is a *timer*: set the time interval, and the timer will fire repeatedly at the interval, signaling a timer event.

A second use of the clock component is to manipulate time, and express time in various units. The internal time format used by the clock is called an *instant*. The clock's **Now** method returns the current time as an instant. The clock provides methods to manipulate instants, for example, return an instant that is several seconds, or months, or years from the given instant. It also provides methods to show the second, minute, hour, day, …, corresponding to a given instant.

Properties:

TimerAlwaysFires:

If true, the timer will fire even if the application is not showing on the screen

TimerEnabled:

If true, then the timer will fire

TimerInterval:

Timer interval in milliseconds

Events:

Timer():

This event is signaled when the timer fired

Methods:

InstantInTime AddDays(InstantInTime instant, number days):

An instant in time some days after the argument

InstantInTime AddHours(InstantInTime instant, number hours):

An instant in time some hours after the argument

InstantInTime AddMinutes(InstantInTime instant, number minutes):

An instant in time some minutes after the argument

InstantInTime AddMonths(InstantInTime instant, number months):

An instant in time some months after the argument

InstantInTime AddSeconds(InstantInTime instant, number seconds):

An instant in time some seconds after the argument

InstantInTime AddWeeks(InstantInTime instant, number weeks):

An instant in time some weeks after the argument

InstantInTime AddYears(InstantInTime instant, number years):

An instant in time some years after the argument

number DayOfMonth(InstantInTime instant):

The day of the month

number Duration(InstantInTime start, InstantInTime end):

Milliseconds between instants

text FormatDate(InstantInTime instant):

Text describing the date of an instant

text FormatDateTime(InstantInTime instant):

Text describing the date and time of an instant

text FormatTime(InstantInTime instant):

Text describing time time of an instant

number GetMillis(InstantInTime instant):

The instant in time measured as milliseconds since 1970.

number Hour(InstantInTime instant):

The hour of the day

InstantInTime MakeInstant(text from):

An instant specified by MM/DD/YYYY hh:mm:ss or MM/DD/YYYY or hh:mm

InstantInTime MakeInstantFromMillis(number millis):

An instant in time specified by the milliseconds since 1970.

number Minute(InstantInTime instant):

The minute of the hour

number Month(InstantInTime instant):

The month of the year, a number from 1 to 12)

text MonthName(InstantInTime instant):

The name of the month

InstantInTime Now():

The instant in time read from phone's clock

number Second(InstantInTime instant):

The second of the minute

number SystemTime():

The phone's internal time in milliseconds number Weekday(InstantInTime instant):

The day of the week. a number from 1 (Sunday) to 7 (Saturday)

text WeekdayName(InstantInTime instant):

The name of the day of the week

number Year(InstantInTime instant):

The year

---

5. Image

You use image components to represent images that users select and manipulate.

An image component displays an image. You can specify the picture to display and other aspects of the image's appearance in the Designer or in the Blocks Editor.

Properties:

Animation Height:

Image height (y-size).

Picture:

Picture displayed by this image.

Visible:

If true, image is displayed.

Width

Image width (x-size).

Events:

None

Methods:

None

---

6. Label

Labels are components used to show text.

A label displays text which is specified by the Text property. Other properties, all of which can be set in the Designer or Blocks Editor, control the appearance and placement of the text.

Events:

None

Methods:

None

---

7. ListPicker

Users tap a list picker component to select one item from a list of text strings.

When a user taps a list picker, it displays a list of text items for the user to choose from. The text items can be specified through the Designer or Blocks Editor by setting the **ElementsFromString** property to their comma-separated concatenation (for example, **choice 1, choice 2, choice 3**) or by setting the **Elements** property to a **List** in the Blocks Editor.

Other properties, including **TextAlignment** and **BackgroundColor**, affect the appearance of the button and whether it can be tapped (**Enabled**).

Properties:

Selection:

Selected list element.

Items:

Comma-separated list of items to display in component.

ElementsFromString:

BackgroundColor:

Color for list picker background.

FontBold:

If set, list picker text is displayed in bold.

FontItalic:

If set, list picker text is displayed in italics.

FontSize:

Point size for list picker text.

FontTypeface:

Font family for list picker text.

Height:

List picker height (y-size).

Width:

List picker width (x-size).

Text:

Title text to display on list picker.

TextAlignment:

Left, center, or right.

TextColor:

Color for list picker text.

Visible:

If set, list picker is visible.

Events:

AfterPicking():

User selected an item from the list picker.

BeforePicking():

User has tapped the list picker but hasn't yet selected an item.

GotFocus():

List picker became the focused component.

LostFocus():

List picker is no longer the focused component.

Methods:

Open():

Opens the picker, as though the user clicked on it.

---

8. PasswordTextBox

Users enter passwords in a password text box component, which hides the text that has been typed in it.

A password text box is the same as the ordinary **TextBox** component, except that it does not display the characters typed by the user.

You can get or set the value of the text in the box with the Text property. If **Text** is blank, you can use the **Hint** property to provide the user with a suggestion of what to type. The **Hint** appears as faint text in the box.

Password text box components are usually used with a button component. The user taps the button after entering text.

Properties:

BackgroundColor:

Color for text box background.

Enabled:

If set, user can enter a password in the box.

FontBold:

If set, text is displayed in bold.

FontItalic:

If set, text is displayed in italics.

FontSize:

Point size for text.

FontTypeface:

Font family for text.

Height:

Box height (y-size).

Width:

Box width (x-size).

Text:

The text in the input box, which can be set by the programmer in the Designer or Blocks Editor, or it can be entered by the user (unless the **Enabled** property is false).

TextAlignment:

Left, center, or right.

TextColor:

Color for text.

Visible:

Specifies whether the component should be visible on the screen. Value is true if the component is showing and false if hidden.

Hint:

Password hint.

Events:

GotFocus():

Box became the focused component.

LostFocus():

Box is no longer the focused component.

Methods:

None

---

9. Screen

The screen does not appear in the palette like other components, but it comes automatically with the project. Each project starts with one screen, named Screen1. This name cannot be changed. More screens can be added.

Properties:

AlignHorizontal:

A number that encodes how contents of the screen are aligned horizontally. The choices are: 1 = left aligned, 2 = horizontally centered, 3 = right aligned.

AlignVertical:

A number that encodes how the contents of the arrangement are aligned vertically. The choices are: 1 = aligned at the top, 2 = vertically centered, 3 = aligned at the bottom. Vertical alignment has no effect if the screen is scrollable.

BackgroundColor:

Color for screen background.

BackgroundImage:

An image that forms the screen's background.

Height:

Screen height (y-size).

Icon:

An image to be used as the icon for the installed application on the phone. This should be a PNG or a JPG image; 48x48 is a good size. *Warning:* Specifying images other than PNG or JPG, for example GIF or .ico files, may prevent App Inventor from being able to package the application.

ScreenOrientation:

The requested screen orientation. Commonly used values are "unspecified", "landscape", "portrait", "sensor", and "user".

Scrollable:

This is set by a checkbox in the designer. When checked, there will be a vertical scrollbar on the screen, and the height of the application can exceed the physical height of the device. When unchecked, the application height is constrained to the height of the device.

VersionCode (designer only - main screen only):

VersionName (designer only - main screen only):

Title:

Title for the screen (text). This will appear at the upper left of the phone when the application runs. A natural choice for the title is the title of the App, but you could make it something else, or even change it while the app is running.

Width:

Screen width (x-size).

Events:

BackPressed():

Device back button pressed.

Initialize():

Signaled when the application starts. It can be used setting initial values and performing other setup operations.

ErrorOccurred(component component, text functionName, number errorNumber, text message):

Signaled when an error occurs. The ErrorOccurred event is currently used for a small set of errors including:

- Errors that occur in the LEGO MINDSTORMS Nxt* components
- Errors that occur in the Bluetooth components
- Errors that occur in the Twitter component
- Errors that occur in the SoundRecorder component
- ActivityStarter - when StartActivity is called, but there is no activity that corresponds to the activity properties.
- LocationSensor - when LatitudeFromAddress or LongitudeFromAddress fails
- Player - when setting the Source property fails
- Sound - when setting the Source property fails or when the Play function fails
- VideoPlayer - when setting the Source property fails

For those errors, the system will show a notification by default, with an error number and a message. You can use this event handler to prescribe an error behavior different than the default, by testing errorNumber and taking the appropriate action.

OtherScreenClosed(text otherScreenName, any result):

Event raised when another screen has closed and control has returned to this screen.

ScreenOrientationChanged():

Screen orientation changed

Methods:

CloseScreenAnimation(text animType):

Sets the animation for closing current screen and returning to the previous screen. Valid options are default, fade, zoom, slidehorizontal, slidevertical, and none

OpenScreenAnimation(text animType):

Sets the animation for switching to another screen. Valid options are default, fade, zoom, slidehorizontal, slidevertical, and none

10. Slider

A Slider is a progress bar that adds a draggable thumb. You can touch the thumb and drag left or right to set the slider thumb position. As the Slider thumb is dragged, it will trigger the PositionChanged event, reporting the position of the Slider thumb. The reported position of the Slider thumb can be used to dynamically update another component attribute, such as the font size of a TextBox or the radius of a Ball.

Properties:

ColorLeft:

The color of slider to the left of the thumb.

ColorRight:

The color of slider to the left of the thumb.

MaxValue:

Sets the maximum value of slider. Changing the maximum value also resets Thumbposition to be halfway between the minimum and the (new) maximum. If the new maximum is less than the current minimum, then minimum and maximum will both be set to this value. Setting MaxValue resets the thumb position to halfway between MinValue and MaxValue and signals the PositionChanged event.

MinValue:

Sets the minimum value of slider. Changing the minimum value also resets Thumbposition to be halfway between the (new) minimum and the maximum. If the new minimum is greater than the current maximum, then minimum and maximum will both be set to this value. Setting MinValue resets the thumb position to halfway between MinValue and MaxValue and signals the PositionChanged event.

ThumbPosition:

Sets the position of the slider thumb. If this value is greater than MaxValue, then it will be set to same value as MaxValue. If this value is less than MinValue, then it will be set to same value as MinValue.

Visible:

Specifies whether the component should be visible on the screen. Value is true if the component is showing and false if hidden.

Width:

Events:

PositionChanged(number thumbPosition):

Indicates that position of the slider thumb has changed.

Methods:

None

---

11. TextBox

Users enter text in a text box component.

The initial or user-entered text value in a text box component is in the Text property. If **Text** is blank, you can use the **Hint** property to provide the user with a suggestion of what to type. The **Hint** appears as faint text in the box.

The **MultiLine** property determines if the text can have more than one line. For a single line text box, the keyboard will close automatically when the user presses the Done key. To close the keyboard for multiline text boxes, the app should use the HideKeyboard method or rely on the user to press the Back key.

The **NumbersOnly** property restricts the keyboard to accept numeric input only.

Other properties affect the appearance of the text box (**TextAlignment, BackgroundColor**, etc.) and whether it can be used (**Enabled**).

Text boxes are usually used with the **Button** component, with the user clicking on the button when text entry is complete.

If the text entered by the user should not be displayed, use **PasswordTextBox** instead.

Properties:

BackgroundColor:

The background color of the input box. You can choose a color by name in the Designer or in the Blocks Editor. The default background color is 'default' (shaded 3-D look).

Enabled:

Whether the user can enter text into this input box. By default, this is true.

FontBold (designer only):

Whether the font for the text should be bold. By default, it is not.

FontItalic (designer only):

Whether the text should appear in italics. By default, it does not.

FontSize:

The font size for the text. By default, it is 14.0 points.

FontTypeface (designer only):

The font for the text. The value can be changed in the Designer.

Height:

Hint:

Text that should appear faintly in the input box to provide a hint as to what the user should enter. This can only be seen if the **Text** property is empty.

MultiLine:

If true, then this text box accepts multiple lines of input, which are entered using the return key. For single line text boxes there is a Done key instead of a return key, and pressing Done hides the keyboard. The app should call the HideKeyboard method to hide the keyboard for a multiline text box.

NumbersOnly:

If true, then this text box accepts only numbers as keyboard input. Numbers can include a decimal point and an optional leading minus sign. This applies to keyboard input only. Even if NumbersOnly is true, you can use [set Text to] to enter any text at all.

Text:

The text in the input box, which can be set by the programmer in the Designer or Blocks Editor, or it can be entered by the user (unless the **Enabled** property is false).

TextAlignment (designer only):

Whether the text should be left justified, centered, or right justified. By default, text is left justified.

TextColor:

The color for the text. You can choose a color by name in the Designer or in the Blocks Editor. The default text color is black.

Visible:

Whether the component is visible

Width:

Events:

GotFocus():

Event raised when this component is selected for input, such as by the user touching it.

LostFocus():

Event raised when this component is no longer selected for input, such as if the user touches a different text box.

Methods:

HideKeyboard():

Hide the keyboard. Only multiline text boxes need this. Single line text boxes close the keyboard when the users presses the Done key.

---

12. TinyDB

Use a TinyDB component to store data that will be available each time the app runs.

TinyDB is a non-visible component.

Apps created with App Inventor are initialized each time they run. If an app sets the value of a variable and the user then quits the app, the value of that variable will not be remembered the next time the app is run. TinyDB is a *persistent* data store for the app, that is, the data stored there will be available each time the app is run. An example might be a game that saved the high score, and retrieved it each time the game is played.

Data items are stored under *tags*. To store a data item, you specify the tag it should be stored under. Subsequently, you can retrieve the data item that was stored under a given tag. If there is no value stored under a tag, then the value returned is the empty text. Consequently, to see if a tag has a value stored under it, test whether the return value is equal to the empty text (i.e., a text box with no text filled in).

There is only one data store per app. If you have multiple TinyDB components, they will use the same data store. To get the effect of separate stores, use different keys. Also each app has its own data store. You cannot use TinyDB to pass data between two different apps on the phone.

Properties:

None

Events:

None

Methods:

StoreValue(text tag, valueToStore):

Store the value under the given tag. The tag must be a text string; the value can be a string or a list.

GetValue(text tag):

Gets the value that was stored under the given tag. If no value was stored, returns the empty text.

Note: To clear out the data base for an app, go on the phone under Settings → Applications → Manage Applications, the pick the app, and press "Clear Data".

The data in TinyDB is persistent only when you have packaged and downloading your app. If you are developing connected to the phone, and you restart the App Inventor application, or if you disconnect and reconnect the phone, then the data base will start fresh. This is a case where the application is not merely being stopped and restarted; it is being removed from the phone and then reloaded.