COMPONENTS - APP INVENTOR FOR ANDROID

COMPONENT REFERENCE

COMPONENT TYPES

This document describes the components you can use in App Inventor to build your apps.

Each component can have methods, events, and properties. Most properties can be changed by apps — these properties have blocks you can use to get and set the values. Some properties can't be changed by apps — these only have blocks you can use to get the values, not set them. In this document, read-only properties are shown in *italics* . A few properties are only available in the Designer.

USER INTERFACE COMPONENTS - APP INVENTOR FOR ANDROID

TABLE OF CONTENTS

BUTTON

Button with the ability to detect clicks. Many aspects of its appearance can be changed, as well as whether it is clickable (`Enabled`), can be changed in the Designer or in the Blocks Editor.

PROPERTIES

`BackgroundColor`

Returns the button's background color

`Enabled`

If set, user can tap check box 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` (designer only)

Font family for button text.

`Height`

Button height (y-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 as an image as background.

`Text`

Text to display on button.

`TextAlignment` (designer only)

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.

`Width`

Button width (x-size).

EVENTS

`Click()`

User tapped and released the button.

`GotFocus()`

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

`LongClick()`

User held the button down.

`LostFocus()`

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

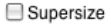
`TouchDown()`

Indicates that the button was pressed down.

`TouchUp()`

Indicates that a button has been released.

CHECKBOX



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

A check box component raises an event 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.

`Height`

Check box height (y-size).

`Width`

Check box width (x-size).

`Text`

Text to display on check box.

`TextColor`

Color for check box text.

`Visible`

If set, check box is visible.

EVENTS

`Click()`

User tapped and released check box.

`GotFocus()`

Check box became the focused component.

`LostFocus()`

Check box stopped being the focused component.

DATEPICKER

A button that, when clicked on, launches a popup dialog to allow the user to select a date.

PROPERTIES

`BackgroundColor`

Returns the button's background color

`DAY`

the Day of the month that was last picked using the DatePicker.

`Enabled`

If set, user can tap check box 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` (designer only)

Font family for button text.

`Height`

`Image`

Image to display on button.

`MONTH`

the number of the Month that was last picked using the DatePicker. Note that months start in 1 = January, 12 = December.

`MONTHINTEXT`

Returns the name of the Month that was last picked using the DatePicker, in textual format.

`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 as an image as background.

`Text`

Text to display on button.

`TextAlignment` (designer only)

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.

`Width`

`YEAR`

the Year that was last picked using the DatePicker

EVENTS

`AfterDateSet()`

Event that runs after the user chooses a Date in the dialog

`GotFocus()`

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

`LostFocus()`

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

`TouchDown()`

Indicates that the button was pressed down.

`TouchUp()`

Indicates that a button has been released.

METHODS

`LaunchPicker()`

Launches the DatePicker popup.

`SetDateToDisplay(number year, number month, number day)`

Allows the user to set the date to be displayed when the date picker opens. Valid values for the month field are 1-12 and 1-31 for the day field.

IMAGE

Component for displaying images. The picture to display, and other aspects of the Image's appearance, can be specified in the Designer or in the Blocks Editor.

PROPERTIES

`Animation`

This is a limited form of animation that can attach a small number of motion types to images. The allowable motions are ScrollRightSlow, ScrollRight, ScrollRightFast, ScrollLeftSlow, ScrollLeft, ScrollLeftFast, and Stop

`Height`

`Picture`

`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

none

METHODS

none

LABEL

Shiny

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.

PROPERTIES

`BackgroundColor`

Color for label background.

`FontBold`

If set, label text is displayed in bold.

`FontItalic`

If set, label text is displayed in italics.

`FontSize`

Point size for label text.

`FontTypeface`

Font family for label text.

`HasMargins`

Reports whether or not the label appears with margins. All four margins (left, right, top, bottom) are the same. This property has no effect in the designer, where labels are always shown with margins.

`Height`

Label height (y-size).

`Width`

Label width (x-size).

`Text`

Text to display on label.

`TextAlignment`

Left, center, or right.

`TextColor`

Color for label text.

`Visible`

If set, label is visible.

LISTPICKER

A button that, when clicked on, displays a list of texts for the user to choose among. The texts can be specified through the Designer or Blocks Editor by setting the`ElementsFromString` property to their string-separated concatenation (for example, CHOICE 1, CHOICE 2, CHOICE 3) or by setting the `Elements` property to a List in the Blocks editor.
Setting property ShowFilterBar to true, will make the list searchable. Other properties affect the appearance of the button (`TextAlignment`, `BackgroundColor`, etc.) and whether it can be clicked on (`Enabled`).

PROPERTIES

`BackgroundColor`

Returns the button's background color

`Elements`

List of Choices to Display (as a list)

`ElementsFromString`

Comma separated list of choices to use

`Enabled`

Whether the ListPicker can be tapped

`FontBold` (designer only)

If set, list picker text is displayed in bold.

`FontItalic` (designer only)

If set, list picker text is displayed in italics.

`FontSize` (designer only)

Point size for list picker text.

`FontTypeface` (designer only)

Font family for list picker text.

`Height`

Box height (y-size).

`Image`

Specifies the path of the button's image. If there is both an Image and a BackgroundColor, only the Image will be visible.

`Selection`

The selected item. When directly changed by the programmer, the SelectionIndex property is also changed to the first item in the ListPicker with the given value. If the value does not appear, SelectionIndex will be set to 0.

`SelectionIndex`

The index of the currently selected item, starting at 1. If no item is selected, the value will be 0. If an attempt is made to set this to a number less than 1 or greater than the number of items in the ListPicker, SelectionIndex will be set to 0, and Selection will be set to the empty text.

`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 as an image as background.

`ShowFilterBar`

Returns current state of ShowFilterBar indicating if Search Filter Bar will be displayed on ListPicker or not

`Text`

Title text to display on list picker.

`TextAlignment` (designer only)

Left, center, or right.

`TextColor`

Color for text.

`Title`

Optional title displayed at the top of the list of choices.

`Visible`

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

`Width`

Box width (x-size).

`ItemTextColor`

The text color of the ListPicker items.

`ItemBackgroundColor`

The background color of the ListPicker items.

EVENTS

`AfterPicking()`

Event to be raised after the picker activity returns its result and the properties have been filled in.

`BeforePicking()`

Event to raise when the button of the component is clicked or the list is shown using the Open block. This event occurs before the list of items is displayed, and can be used to prepare the list before it is shown.

`GotFocus()`

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

`LostFocus()`

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

METHODS

`Open()`

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

LISTVIEW

This is a visible component that allows to place a list of text elements in your Screen to display.
The list can be set using the ElementsFromString property or using the Elements block in the blocks editor.
Warning: This component will not work correctly on Screens that are scrollable.

PROPERTIES

`BackgroundColor`

The color of the listview background.

`Elements`

List of text elements to build your list.

`ElementsFromString`

Build a list with a series of text elements separated by commas such as: Cheese,Fruit,Bacon,Radish. Each word before the comma will be an element in the list.

`Height`

Determines the height of the list on the view.

`Selection`

Returns the text last selected in the ListView.

`SelectionIndex`

The index of the currently selected item, starting at 1. If no item is selected, the value will be 0. If an attempt is made to set this to a number less than 1 or greater than the number of items in the ListView, SelectionIndex will be set to 0, and Selection will be set to the empty text.

`ShowFilterBar`

Sets visibility of ShowFilterBar. True will show the bar, False will hide it.

`TextColor`

The text color of the listview items.

`Visible`

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

`Width`

Determines the width of the list on the view.

EVENTS

`AfterPicking()`

Simple event to be raised after the an element has been chosen in the list. The selected element is available in the Selection property.

METHODS

none

NOTIFIER

The Notifier component displays alert dialogs, messages, and temporary alerts, and creates Android log entries through the following methods:

- ShowMessageDialog: displays a message which the user must dismiss by pressing a button.
- ShowChooseDialog: displays a message two buttons to let the user choose one of two responses, for example, yes or no, after which the AfterChoosing event is raised.
- ShowTextDialog: lets the user enter text in response to the message, after which the AfterTextInput event is raised.
- ShowAlert: displays a temporary alert that goes away by itself after a short time.
- ShowProgressDialog: displays an alert with a loading spinner that cannot be dismissed by the user. It can only be dismissed by using the DismissProgressDialog block.
- DismissProgressDialog: Dismisses the progress dialog displayed by ShowProgressDialog.
- LogError: logs an error message to the Android log.
- LogInfo: logs an info message to the Android log.
- LogWarning: logs a warning message to the Android log.
- The messages in the dialogs (but not the alert) can be formatted using the following HTML tags:<b>, <big>, <blockquote>,
, <cite>, <dfn>, , , <small>, , <sub>, <sup>, <tt>. <u>
- You can also use the font tag to specify color, for example, <font color="blue">. Some of the available color names are aqua, black, blue, fuchsia, green, grey, lime, maroon, navy, olive, purple, red, silver, teal, white, and yellow

PROPERTIES

`BackgroundColor`

Specifies the background color for alerts (not dialogs).

`NotifierLength` (designer only)

specifies the length of time that the alert is shown -- either "short" or "long".

`TextColor`

Specifies the text color for alerts (not dialogs).

EVENTS

`AfterChoosing(text choice)`

Event after the user has made a selection for ShowChooseDialog.

`AfterTextInput(text response)`

Event raised after the user has responded to ShowTextDialog.

METHODS

`DismissProgressDialog()`

Dismiss a previously displayed ProgressDialog box

`LogError(text message)`

Writes an error message to the Android system log. See the Google Android documentation for how to access the log.

`LogInfo(text message)`

Writes an information message to the Android log.

`LogWarning(text message)`

Writes a warning message to the Android log. See the Google Android documentation for how to access the log.

`ShowAlert(text notice)`

Display a temporary notification

`ShowChooseDialog(text message, text title, text button1Text, text button2Text, boolean cancelable)`

Shows a dialog box with two buttons, from which the user can choose. If cancelable is true there will be an additional CANCEL button. Pressing a button will raise the AfterChoosing event. The "choice" parameter to AfterChoosing will be the text on the button that was pressed, or "Cancel" if the CANCEL button was pressed.

`ShowMessageDialog(text message, text title, text buttonText)`

Display an alert dialog with a single button that dismisses the alert.

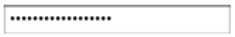
`ShowProgressDialog(text message, text title)`

Shows a dialog box with an optional title and message (use empty strings if they are not wanted). This dialog box contains a spinning artifact to indicate that the program is working. It cannot be canceled by the user but must be dismissed by the App Inventor Program by using the DismissProgressDialog block.

`ShowTextDialog(text message, text title, boolean cancelable)`

Shows a dialog box where the user can enter text, after which the AfterTextInput event will be raised. If cancelable is true there will be an additional CANCEL button. Entering text will raise the AfterTextInput event. The "response" parameter to AfterTextInput will be the text that was entered, or "Cancel" if the CANCEL button was pressed.

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.

METHODS

`RequestFocus()`

Sets the PasswordTextBox active.

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).

`TextAlignment`

Left, center, or right.

`TextColor`

Color for text.

`Hint`

Password hint.

EVENTS

`GotFocus()`

Box became the focused component.

`LostFocus()`

Box is no longer the focused component.

SCREEN

Top-level component containing all other components in the program

PROPERTIES

`AboutScreen`

Information about the screen. It appears when "About this Application" is selected from the system menu. Use it to tell users about your app. In multiple screen apps, each screen has its own AboutScreen info.

`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`

`AppName` (designer only)

This is the display name of the installed application in the phone. If the AppName is blank, it will be set to the name of the project when the project is built.

`BackgroundImage`

The screen background image.

`CloseScreenAnimation`

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

`HEIGHT`

Screen height (y-size).

`Icon` (designer only)

`OpenScreenAnimation`

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

`ScreenOrientation`

The requested screen orientation, specified as a text value. Commonly used values are landscape, portrait, sensor, user and unspecified. See the Android developer documentation for ActivityInfo.Screen_Orientation for the complete list of possible settings.

`Scrollable`

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.

`SHOWSTATUSBAR`

The status bar is the topmost bar on the screen. This property reports whether the status bar is visible.

`TITLEVISIBLE`

The title bar is the top gray bar on the screen. This property reports whether the title bar is visible.

`Title`

The caption for the form, which apears in the title bar

`VersionCode` (designer only)

An integer value which must be incremented each time a new Android Application Package File (APK) is created for the Google Play Store.

`VersionName` (designer only)

A string which can be changed to allow Google Play Store users to distinguish between different versions of the App.

`WIDTH`

Screen width (x-size).

EVENTS

`BackPressed()`

Device back button pressed.

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

Event raised when an error occurs. Only some errors will raise this condition. For those errors, the system will show a notification by default. You can use this event handler to prescribe an error behavior different than the default.

`Initialize()`

Screen starting

`OtherScreenClosed(text otherScreenName, any result)`

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

`ScreenOrientationChanged()`

Screen orientation changed

METHODS

none

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.

`ThumbEnabled`

Sets whether or not to display the slider thumb.

`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

SPINNER

A spinner component that displays a pop-up with a list of elements. These elements can be set in the Designer or Blocks Editor by setting the`ElementsFromString`property to a string-separated concatenation (for example, CHOICE 1, CHOICE 2, CHOICE 3) or by setting the `Elements` property to a List in the Blocks editor. Spinners are created with the first item already selected. So selecting it does not generate an After Picking event. Consequently it's useful to make the first Spinner item be a non-choice like "Select from below...".

PROPERTIES

`Elements`

returns a list of text elements to be picked from.

`ElementsFromString`

sets the Spinner list to the elements passed in the comma-separated string

`Height`

`Prompt`

Text with the current title for the Spinner window

`Selection`

Returns the current selected item in the spinner

`SelectionIndex`

The index of the currently selected item, starting at 1. If no item is selected, the value will be 0.

`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

`AfterSelecting(text selection)`

Event called after the user selects an item from the dropdown list.

METHODS

`DisplayDropdown()`

displays the dropdown list for selection, same action as when the user clicks on the spinner.

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.

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.

`RequestFocus()`

Sets the textbox active.

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 mutiline 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.

TIMEPICKER

A button that, when clicked on, launches a popup dialog to allow the user to select a time.

PROPERTIES

`BackgroundColor`

Returns the button's background color

`Enabled`

`FontBold` (designer only)

`FontItalic` (designer only)

`FontSize` (designer only)

`FontTypeface` (designer only)

`Height`

`HOUR`

The hour of the last time set using the time picker. The hour is in a 24 hour format. If the last time set was 11:53 pm, this property will return 23.

`Image`

Specifies the path of the button's image. If there is both an Image and a BackgroundColor, only the Image will be visible.

`MINUTE`

The minute of the last time set using the time picker

`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 as an image as background.

`Text`

`TextAlignment` (designer only)

`TextColor`

`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

`AfterTimeSet()`

This event is run when a user has set the time in the popup dialog.

`GotFocus()`

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

`LostFocus()`

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

METHODS

none

WEBVIEWER

Component for viewing Web pages. The Home URL can be specified in the Designer or in the Blocks Editor. The view can be set to follow links when they are tapped, and users can fill in Web forms. Warning: This is not a full browser. For example, pressing the phone's hardware Back key will exit the app, rather than move back in the browser history.

You can use the WebViewer.WebViewString property to communicate between your app and Javascript code running in the Webviewer page. In the app, you get and set WebViewString. In the WebViewer, you include Javascript that references the window.AppInventor object, using the methoods and SETWEBVIEWSTRING(TEXT).
For example, if the WebViewer opens to a page that contains the Javascript command
DOCUMENT.WRITE("THE ANSWER IS" + WINDOW.APPINVENTOR.GETWEBVIEWSTRING());
and if you set WebView.WebVewString to "hello", then the web page will show
THE ANSWER IS HELLO.
And if the Web page contains Javascript that executes the command
WINDOWAPPINVENTOR.SETWEBVIEWSTRING("HELLO FROM JAVASCRIPT"),
then the value of the WebViewString property will be
HELLO FROM JAVASCRIPT.

PROPERTIES

`CURRENTPAGETITLE`

Title of the page currently viewed

`CURRENTURL`

URL of the page currently viewed. This could be different from the Home URL if new pages were visited by following links.

`FollowLinks`

Determines whether to follow links when they are tapped in the WebViewer. If you follow links, you can use GoBack and GoForward to navigate the browser history.

`Height`

`HomeUrl`

URL of the page the WebViewer should initially open to. Setting this will load the page.

`IgnoreSslError`

Determine whether or not to ignore SSL errors. Set to true to ignore errors. Use this to accept self signed certificates from websites.

`PromptforPermission`

If True, then prompt the user of the WebView to give permission to access the geolocation API. If False, then assume permission is granted.

`UsesLocation` (designer only)

Whether or not to give the application permission to use the Javascript geolocation API. This property is available only in the designer.

`Visible`

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

`WebViewString`

Gets the WebView's String, which is viewable through Javascript in the WebView as the window.AppInventor object

`Width`

EVENTS

none

METHODS

`boolean CanGoBack()`

Returns true if the WebViewer can go back in the history list.

`boolean CanGoForward()`

Returns true if the WebViewer can go forward in the history list.

`ClearCaches()`

Clear the WebViewer caches

`ClearLocations()`

Clear stored location permissions.

`GoBack()`

Go back to the previous page in the history list. Does nothing if there is no previous page.

`GoForward()`

Go forward to the next page in the history list. Does nothing if there is no next page.

`GoHome()`

Loads the home URL page. This happens automatically when the home URL is changed.

`GoToUrl(text url)`

Load the page at the given URL.

LAYOUT COMPONENTS - APP INVENTOR FOR ANDROID

TABLE OF CONTENTS

HORIZONTALARRANGEMENT

Use a horizontal arrangement component to display a group of components laid out from left to right.

This component is a formatting element in which you place components that should be displayed from left to right. If you want to have components displayed one over another, use `VerticalArrangement` instead.

In a HorizontalArrangement, components are arranged along the horizontal axis, vertically center-aligned.

If a HorizontalArrangement's Height property is set to Automatic, the actual height of the arrangement is determined by the tallest component in the arrangement whose Height property is not set to Fill Parent. If a HorizontalArrangment's Height property is set to Automatic and it contains only components whose Height properties are set to Fill Parent, the actual height of the arrangement is calculated using the automatic heights of the components. If a HorizontalArrangement's Height property is set to Automatic and it is empty, the height will be 100.

If a HorizontalArrangement's Width property is set to Automatic, the actual width of the arrangement is determined by the sum of the widths of the components. **If a HorizontalArrangement's Width property is set to Automatic, any components whose Width properties are set to Fill Parent will behave as if they were set to Automatic.**

If a HorizontalArrangement's Width property is set to Fill Parent or specified in pixels, any components whose Width properties are set to Fill Parent will equally take up the width not occupied by other components.

PROPERTIES

`Visible`

: If true, component and its contents are visible.

`Height`

: Horizontal arrangement height (y-size).

`Width`

: Horizontal arrangement width (x-size).

TABLEARRANGEMENT

Use a table arrangement component to display a group of components in a tabular fashion.

This component is a formatting element in which you place components that should be displayed in tabular form.

In a TableArrangement, components are arranged in a grid of rows and columns, with not more than one component visible in each cell. **If multiple components occupy the same cell, only the last one will be visible.**

Within each row, components are vertically center-aligned.

The width of a column is determined by the widest component in that column. When calculating column width, the automatic width is used for components whose Width property is set to Fill Parent. **However, each component will always fill the full width of the column that it occupies.**

The height of a row is determined by the tallest component in that row whose Height property is not set to Fill Parent. If a row contains only components whose Height properties are set to Fill Parent, the height of the row is calculated using the automatic heights of the components.

PROPERTIES

`Visible`

: If true, component and its contents are visible.

`Rows (number-of-rows)`

: The number of rows in the table.

`Columns (number-of-columns)`

: The number of columns in the table.

`Height`

: Table arrangement height (y-size).

`Width`

: Table arrangement width (x-size).

VERTICALARRANGEMENT

Use a vertical arrangement component to display a group of components laid out from top to bottom, left-aligned.

This component is a formatting element in which you place components that should be displayed one below another. The first child component is stored on top, the second beneath it, and so on. If you want to have components displayed next to one another, use `HorizontalArrangement` instead.

In a VerticalArrangement, components are arranged along the vertical axis, left-aligned.

If a VerticalArrangement's Width property is set to Automatic, the actual width of the arrangement is determined by the widest component in the arrangement whose Width property is not set to Fill Parent. If a VerticalArrangement's Width property is set to Automatic and it contains only components whose Width properties are set to Fill Parent, the actual width of the arrangement is calculated using the automatic widths of the components. If a VerticalArrangement's Width property is set to Automatic and it is empty, the width will be 100.

If a VerticalArrangement's Height property is set to Automatic, the actual height of the arrangement is determined by the sum of the heights of the components. **If a VerticalArrangement's Height property is set to Automatic, any components whose Height properties are set to Fill Parent will behave as if they were set to Automatic.**

If a VerticalArrangement's Height property is set to Fill Parent or specified in pixels, any components whose Height properties are set to Fill Parent will equally take up the height not occupied by other components.

PROPERTIES

`Visible`

If true, component and its contents are visible.

`Height`

Vertical arrangement height (y-size).

`Width`

Vertical arrangement width (x-size).

MEDIA COMPONENTS - APP INVENTOR FOR ANDROID

TABLE OF CONTENTS

CAMCORDER

A component to record a video using the device's camcorder.After the video is recorded, the name of the file on the phone containing the clip is available as an argument to the AfterRecording event. The file name can be used, for example, to set the source property of a VideoPlayer component.

PROPERTIES

none

EVENTS

`AfterRecording(text clip)`

Indicates that a video was recorded with the camera and provides the path to the stored picture.

METHODS

`RecordVideo()`

Records a video, then raises the AfterRecoding event.

CAMERA

Use a camera component to take a picture on the phone.

Camera is a non-visible component that takes a picture using the device's camera. After the picture is taken, the path to the file on the phone containing the picture is available as an argument to the AfterPicture event. The path can be used, for example, as the Picture property of an Image component.

PROPERTIES

`UseFront`

Specifies whether the front-facing camera should be used (when available). If the device does not have a front-facing camera, this option will be ignored and the camera will open normally.

METHODS

`TakePicture()`

Opens the phone's camera to allow a picture to be taken.

EVENTS

`AfterPicture(Text image)`

Called after the picture is taken. The text argument `image` is the path that can be used to locate the image on the phone.

IMAGEPICKER

A special-purpose button. When the user taps an image picker, the device's image gallery appears, and the user can choose an image. After an image is picked, it is saved, and the `Selected` property will be the name of the file where the image is stored. In order to not fill up storage, a maximum of 10 images will be stored. Picking more images will delete previous images, in order from oldest to newest.

PROPERTIES

`BackgroundColor`

Returns the button's background color

`Enabled`

`FontBold` (designer only)

`FontItalic` (designer only)

`FontSize` (designer only)

`FontTypeface` (designer only)

`Height`

`Image`

Specifies the path of the button's image. If there is both an Image and a BackgroundColor, only the Image will be visible.

`SELECTION`

Path to the file containing the image that was selected.

`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 as an image as background.

`Text`

`TextAlignment` (designer only)

`TextColor`

`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

`AfterPicking()`

Simple event to be raised after the picker activity returns its result and the properties have been filled in.

`BeforePicking()`

Simple event to raise when the component is clicked but before the picker activity is started.

`GotFocus()`

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

`LostFocus()`

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

METHODS

`Open()`

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

PLAYER

Multimedia component that plays audio and controls phone vibration. The name of a multimedia field is specified in the `Source` property, which can be set in the Designer or in the Blocks Editor. The length of time for a vibration is specified in the Blocks Editor in milliseconds (thousandths of a second).
For supported audio formats, see Android Supported Media Formats.
This component is best for long sound files, such as songs, while the `Sound` component is more efficient for short files, such as sound effects.

PROPERTIES

`ISPLAYING`

Reports whether the media is playing

`Loop`

If true, the player will loop when it plays. Setting Loop while the player is playing will affect the current playing.

`PlayOnlyInForeground`

If true, the player will pause playing when leaving the current screen; if false (default option), the player continues playing whenever the current screen is displaying or not.

`Source`

`Volume`

Sets the volume to a number between 0 and 100

EVENTS

`Completed()`

Indicates that the media has reached the end

`OtherPlayerStarted()`

This event is signaled when another player has started (and the current player is playing or paused, but not stopped).

METHODS

`Pause()`

Suspends playing the media if it is playing.

`Start()`

Plays the media. If it was previously paused, the playing is resumed. If it was previously stopped, it starts from the beginning.

`Stop()`

Stops playing the media and seeks to the beginning of the song.

`Vibrate(number milliseconds)`

Vibrates for specified number of milliseconds.

SOUND

A multimedia component that plays sound files and optionally vibrates for the number of milliseconds (thousandths of a second) specified in the Blocks Editor. The name of the sound file to play can be specified either in the Designer or in the Blocks Editor.

For supported sound file formats, see Android Supported Media Formats.
This `Sound` component is best for short sound files, such as sound effects, while the `Player` component is more efficient for longer sounds, such as songs.

PROPERTIES

`MinimumInterval`

The minimum interval between sounds. If you play a sound, all further Play() calls will be ignored until the interval has elapsed.

`Source`

The name of the sound file. Only certain formats are supported. See http://developer.android.com/guide/appendix/media-formats.html.

EVENTS

none

METHODS

`Pause()`

Pauses playing the sound if it is being played.

`Play()`

Plays the sound.

`Resume()`

Resumes playing the sound after a pause.

`Stop()`

Stops playing the sound if it is being played.

`Vibrate(number millisecs)`

Vibrates for the specified number of milliseconds.

SOUNDRECORDER

Multimedia component that records audio.

PROPERTIES

none

EVENTS

`AfterSoundRecorded(text sound)`

Provides the location of the newly created sound.

`StartedRecording()`

Indicates that the recorder has started, and can be stopped.

`StoppedRecording()`

Indicates that the recorder has stopped, and can be started again.

METHODS

`Start ()`

Starts recording.

`Stop ()`

Stops recording.

SPEECHRECOGNIZER

Use a speech recognizer component to listen to the user speaking and convert the spoken sound into text using Android's speech recognition feature.

PROPERTIES

`Result`

The last text produced by the recognizer.

METHODS

`GetText()`

Asks the user to speak, and converts the speech to text. Signals the `AfterGettingText` event when the result is available.

EVENTS

`AfterGetting(Text result)`

Signaled after the recognizer has produced text. The argument is the text result that was produced.

`BeforeGettingText()`

Signaled just before the recognizer is called.

TEXTTOSPEECH

The TestToSpeech component speaks a given text aloud. You can set the pitch and the rate of speech.

You can also set a language by supplying a language code. This changes the pronounciation of words, not the actual language spoken. For example, setting the language to French and speaking English text will sound like someone speaking English (en) with a French accent.

You can also specify a country by supplying a country code. This can affect the pronounciation. For example, British English (GBR) will sound different from US English (USA). Not every country code will affect every language.

The languages and countries available depend on the particular device, and can be listed with the AvailableLanguages and AvailableCountries properties.

PROPERTIES

`AVAILABLECOUNTRIES`

List of the country codes available on this device for use with TextToSpeech. Check the Android developer documentation under supported languages to find the meanings of these abbreviations.

`AVAILABLELANGUAGES`

List of the languages available on this device for use with TextToSpeech. Check the Android developer documentation under supported languages to find the meanings of these abbreviations.

`Country`

Country code to use for speech generation. This can affect the pronounciation. For example, British English (GBR) will sound different from US English (USA). Not every country code will affect every language.

`Language`

Sets the language for TextToSpeech. This changes the way that words are pronounced, not the actual language that is spoken. For example setting the language to and speaking English text with sound like someone speaking English with a Frernch accent.

`Pitch`

Sets the Pitch for TextToSpeech. The values should be between 0 and 2 where lower values lower the tone of synthesized voice and greater values raise it.

`RESULT`

`SpeechRate`

Sets the SpeechRate for TextToSpeech. The values should be between 0 and 2 where lower values slow down the pitch and greater values accelerate it.

EVENTS

`AfterSpeaking(boolean result)`

Event to raise after the message is spoken.

`BeforeSpeaking()`

Event to raise when Speak is invoked, before the message is spoken.

METHODS

`Speak(text message)`

Speaks the given message.

VIDEOPLAYER

A multimedia component capable of playing videos. When the application is run, the VideoPlayer will be displayed as a rectangle on-screen. If the user touches the rectangle, controls will appear to play/pause, skip ahead, and skip backward within the video. The application can also control behavior by calling the `Start`, `Pause`, and `SeekTo` methods. Video files should be in 3GPP (.3gp) or MPEG-4 (.mp4) formats. For more details about legal formats, see Android Supported Media Formats.

App Inventor for Android only permits video files under 1 MB and limits the total size of an application to 5 MB, not all of which is available for media (video, audio, and sound) files. If your media files are too large, you may get errors when packaging or installing your application, in which case you should reduce the number of media files or their sizes. Most video editing software, such as Windows Movie Maker and Apple iMovie, can help you decrease the size of videos by shortening them or re-encoding the video into a more compact format.

You can also set the media source to a URL that points to a streaming video, but the URL must point to the video file itself, not to a program that plays the video.

PROPERTIES

`FullScreen`

`Height`

`Source`

The "path" to the video. Usually, this will be the name of the video file, which should be added in the Designer.

`Visible`

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

`Volume`

Sets the volume to a number between 0 and 100. Values less than 0 will be treated as 0, and values greater than 100 will be treated as 100.

`Width`

EVENTS

`Completed()`

Indicates that the video has reached the end

METHODS

`number GetDuration()`

Returns duration of the video in milliseconds.

`Pause()`

Pauses playback of the video. Playback can be resumed at the same location by calling the `Start` method.

`SeekTo(number ms)`

Seeks to the requested time (specified in milliseconds) in the video. Note that if the video is paused, the frame shown will not be updated by the seek.

`Start()`

Starts playback of the video.

YANDEXTRANSLATE

Use this component to translate words and sentences between different languages. This component needs Internet access, as it will request translations to the Yandex.Translate service. Specify the source and target language in the form source-target using two letter language codes. So"en-es" will translate from English to Spanish while "es-ru" will translate from Spanish to Russian. If you leave out the source language, the service will attempt to detect the source language. So providing just "es" will attempt to detect the source language and translate it to Spanish.

This component is powered by the Yandex translation service. See http://api.yandex.com/translate/ for more information, including the list of available languages and the meanings of the language codes and status codes.

Note: Translation happens asynchronously in the background. When the translation is complete, the "GotTranslation" event is triggered.

PROPERTIES

none

EVENTS

`GotTranslation(text responseCode, text translation)`

Event triggered when the Yandex.Translate service returns the translated text. This event also provides a response code for error handling. If the responseCode is not 200, then something went wrong with the call, and the translation will not be available.

METHODS

`RequestTranslation(text languageToTranslateTo, text textToTranslate)`

By providing a target language to translate to (for instance, 'es' for Spanish, 'en' for English, or 'ru' for Russian), and a word or sentence to translate, this method will request a translation to the Yandex.Translate service. Once the text is translated by the external service, the event GotTranslation will be executed. Note: Yandex.Translate will attempt to detect the source language. You can also specify prepending it to the language translation. I.e., es-ru will specify Spanish to Russian translation.

DRAWING AND ANIMATION COMPONENTS - APP INVENTOR FOR ANDROID

TABLE OF CONTENTS

BALL

A round 'sprite' that can be placed on a `Canvas`, where it can react to touches and drags, interact with other sprites (`ImageSprite`s and other `Ball`s) and the edge of the Canvas, and move according to its property values.
For example, to have a `Ball` move 4 pixels toward the top of a `Canvas` every 500 milliseconds (half second), you would set the `Speed` property to 4 [pixels], the`Interval` property to 500 [milliseconds], the `Heading` property to 90 [degrees], and the `Enabled` property to `True`. These and its other properties can be changed at any time.
The difference between a Ball and an `ImageSprite` is that the latter can get its appearance from an image file, while a Ball's appearance can only be changed by varying its `PaintColor` and `Radius` properties.

PROPERTIES

`Enabled`

Controls whether the sprite moves when its speed is non-zero.

`Heading`

Returns the sprite's heading in degrees above the positive x-axis. Zero degrees is toward the right of the screen; 90 degrees is toward the top of the screen.

`Interval`

The interval in milliseconds at which the sprite's position is updated. For example, if the interval is 50 and the speed is 10, then the sprite will move 10 pixels every 50 milliseconds.

`PaintColor`

`Radius`

`Speed`

The speed at which the sprite moves. The sprite moves this many pixels every interval.

`Visible`

True if the sprite is visible.

`X`

The horizontal coordinate of the left edge of the sprite, increasing as the sprite moves to the right.

`Y`

The vertical coordinate of the top of the sprite, increasing as the sprite moves down.

`Z`

How the sprite should be layered relative to other sprits, with higher-numbered layers in front of lower-numbered layers.

EVENTS

`CollidedWith(component other)`

Handler for CollidedWith events, called when two sprites collide. Note that checking for collisions with a rotated ImageSprite currently checks against the sprite's unrotated position. Therefore, collision checking will be inaccurate for tall narrow or short wide sprites that are rotated.

`Dragged(number startX, number startY, number prevX, number prevY, number currentX, number currentY)`

Handler for Dragged events. On all calls, the starting coordinates are where the screen was first touched, and the "current" coordinates describe the endpoint of the current line segment. On the first call within a given drag, the "previous" coordinates are the same as the starting coordinates; subsequently, they are the "current" coordinates from the prior call. Note that the Sprite won't actually move anywhere in response to the Dragged event unless MoveTo is specifically called.

`EdgeReached(number edge)`

Event handler called when the sprite reaches an edge of the screen. If Bounce is then called with that edge, the sprite will appear to bounce off of the edge it reached. Edge here is represented as an integer that indicates one of eight directions north(1), northeast(2), east(3), southeast(4), south (-1), southwest(-2), west(-3), and northwest(-4).

`Flung(number x, number y, number speed, number heading, number xvel, number yvel)`

When a fling gesture (quick swipe) is made on the sprite: 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.

`NoLongerCollidingWith(component other)`

Event indicating that a pair of sprites are no longer colliding.

`TouchDown(number x, number y)`

When the user begins touching the sprite (places finger on sprite 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 sprite (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)`

When the user touches the sprite and then immediately lifts finger: provides the (x,y) position of the touch, relative to the upper left of the canvas

METHODS

`Bounce(number edge)`

Makes this sprite bounce, as if off a wall. For normal bouncing, the edge argument should be the one returned by EdgeReached.

`boolean CollidingWith(component other)`

Indicates whether a collision has been registered between this sprite and the passed sprite.

`MoveIntoBounds()`

Moves the sprite back in bounds if part of it extends out of bounds, having no effect otherwise. If the sprite is too wide to fit on the canvas, this aligns the left side of the sprite with the left side of the canvas. If the sprite is too tall to fit on the canvas, this aligns the top side of the sprite with the top side of the canvas.

`MoveTo(number x, number y)`

Moves the sprite so that its left top corner is at the specfied x and y coordinates.

`PointInDirection(number x, number y)`

Turns the sprite to point towards the point with coordinates as (x, y).

`PointTowards(component target)`

Turns the sprite to point towards a designated target sprite. The new heading will be parallel to the line joining the centerpoints of the two sprites.

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`

`LineWidth`

The width of lines drawn on the canvas.

`PaintColor`

The color in which lines are drawn

`TextAlignment`

Determines the alignment of the text drawn by DrawText() or DrawAngle() with respect to the point specified by that command.

`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

`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.

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

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

IMAGESPRITE

A 'sprite' that can be placed on a `Canvas`, where it can react to touches and drags, interact with other sprites (`Ball`s and other `ImageSprite`s) and the edge of the Canvas, and move according to its property values. Its appearance is that of the image specified in its `Picture` property (unless its `Visible` property is `False`.
To have an `ImageSprite` move 10 pixels to the left every 1000 milliseconds (one second), for example, you would set the `Speed` property to 10 [pixels], the `Interval`property to 1000 [milliseconds], the `Heading` property to 180 [degrees], and the `Enabled` property to `True`. A sprite whose `Rotates` property is `True` will rotate its image as the sprite's `Heading` changes. Checking for collisions with a rotated sprite currently checks the sprite's unrotated position so that collision checking will be inaccurate for tall narrow or short wide sprites that are rotated. Any of the sprite properties can be changed at any time under program control.

PROPERTIES

`Enabled`

Controls whether the sprite moves when its speed is non-zero.

`Heading`

Returns the sprite's heading in degrees above the positive x-axis. Zero degrees is toward the right of the screen; 90 degrees is toward the top of the screen.

`Height`

`Interval`

The interval in milliseconds at which the sprite's position is updated. For example, if the interval is 50 and the speed is 10, then the sprite will move 10 pixels every 50 milliseconds.

`Picture`

The picture that determines the sprite's appearence

`Rotates`

If true, the sprite image rotates to match the sprite's heading. If false, the sprite image does not rotate when the sprite changes heading. The sprite rotates around its centerpoint.

`Speed`

The speed at which the sprite moves. The sprite moves this many pixels every interval.

`Visible`

True if the sprite is visible.

`Width`

`X`

The horizontal coordinate of the left edge of the sprite, increasing as the sprite moves to the right.

`Y`

The vertical coordinate of the top of the sprite, increasing as the sprite moves down.

`Z`

How the sprite should be layered relative to other sprits, with higher-numbered layers in front of lower-numbered layers.

EVENTS

`CollidedWith(component other)`

Handler for CollidedWith events, called when two sprites collide. Note that checking for collisions with a rotated ImageSprite currently checks against the sprite's unrotated position. Therefore, collision checking will be inaccurate for tall narrow or short wide sprites that are rotated.

`Dragged(number startX, number startY, number prevX, number prevY, number currentX, number currentY)`

Handler for Dragged events. On all calls, the starting coordinates are where the screen was first touched, and the "current" coordinates describe the endpoint of the current line segment. On the first call within a given drag, the "previous" coordinates are the same as the starting coordinates; subsequently, they are the "current" coordinates from the prior call. Note that the Sprite won't actually move anywhere in response to the Dragged event unless MoveTo is specifically called.

`EdgeReached(number edge)`

Event handler called when the sprite reaches an edge of the screen. If Bounce is then called with that edge, the sprite will appear to bounce off of the edge it reached. Edge here is represented as an integer that indicates one of eight directions north(1), northeast(2), east(3), southeast(4), south (-1), southwest(-2), west(-3), and northwest(-4).

`Flung(number x, number y, number speed, number heading, number xvel, number yvel)`

When a fling gesture (quick swipe) is made on the sprite: 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.

`NoLongerCollidingWith(component other)`

Event indicating that a pair of sprites are no longer colliding.

`TouchDown(number x, number y)`

When the user begins touching the sprite (places finger on sprite 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 sprite (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)`

When the user touches the sprite and then immediately lifts finger: provides the (x,y) position of the touch, relative to the upper left of the canvas

METHODS

`Bounce(number edge)`

Makes this sprite bounce, as if off a wall. For normal bouncing, the edge argument should be the one returned by EdgeReached.

`boolean CollidingWith(component other)`

Indicates whether a collision has been registered between this sprite and the passed sprite.

`MoveIntoBounds()`

Moves the sprite back in bounds if part of it extends out of bounds, having no effect otherwise. If the sprite is too wide to fit on the canvas, this aligns the left side of the sprite with the left side of the canvas. If the sprite is too tall to fit on the canvas, this aligns the top side of the sprite with the top side of the canvas.

`MoveTo(number x, number y)`

Moves the sprite so that its left top corner is at the specfied x and y coordinates.

`PointInDirection(number x, number y)`

Turns the sprite to point towards the point with coordinates as (x, y).

`PointTowards(component target)`

Turns the sprite to point towards a designated target sprite. The new heading will be parallel to the line joining the centerpoints of the two sprites.

SENSOR COMPONENTS - APP INVENTOR FOR ANDROID

TABLE OF CONTENTS

ACCELEROMETERSENSOR

Non-visible component that can detect shaking and measure acceleration approximately in three dimensions using SI units (m/s^2). The components are:

- **xAccel**: 0 when the phone is at rest on a flat surface, positive when the phone is tilted to the right (i.e., its left side is raised), and negative when the phone is tilted to the left (i.e., its right size is raised).
- **yAccel**: 0 when the phone is at rest on a flat surface, positive when its bottom is raised, and negative when its top is raised.
- **zAccel**: Equal to -9.8 (earth's gravity in meters per second per second when the device is at rest parallel to the ground with the display facing up, 0 when perpindicular to the ground, and +9.8 when facing down. The value can also be affected by accelerating it with or against gravity.

PROPERTIES

`AVAILABLE`

`Enabled`

`MinimumInterval`

The minimum interval between phone shakes

`Sensitivity`

A number that encodes how sensitive the accelerometer is. The choices are: 1 = weak, 2 = moderate, 3 = strong.

`XACCEL`

`YACCEL`

`ZACCEL`

EVENTS

`AccelerationChanged(number xAccel, number yAccel, number zAccel)`

Indicates the acceleration changed in the X, Y, and/or Z dimensions.

`Shaking()`

Indicates the device started being shaken or continues to be shaken.

METHODS

none

BARCODESCANNER

Component for using the Barcode Scanner to read a barcode

PROPERTIES

`RESULT`

Text result of the previous scan.

`USEEXTERNALSCANNER`

If true App Inventor will look for and use an external scanning program such as "Bar Code Scanner."

EVENTS

`AfterScan(text result)`

Indicates that the scanner has read a (text) result and provides the result

METHODS

`DoScan()`

Begins a barcode scan, using the camera. When the scan is complete, the AfterScan event will be raised.

CLOCK

Non-visible component that provides the instant in time using the internal clock on the phone.It can fire a timer at regularly set intervals and perform time calculations, manipulations, and conversions.

Methods to convert an instant to text are also available. Acceptable patterns are empty string, MM/DD/YYYY HH:mm:ss a, or MMM d, yyyy HH:mm. The empty string will provide the default format, which is "MMM d, yyyy HH:mm:ss a" for FormatDateTime, "MMM d, yyyy" for FormatDate. To see all possible format, please see here.

PROPERTIES

`TimerAlwaysFires`

Will fire even when application is not showing on the screen if true

`TimerEnabled`

Fires timer if true

`TimerInterval`

Interval between timer events in ms

EVENTS

`Timer()`

Timer has gone off.

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 elapsed between instants

`text FormatDate(InstantInTime instant, text pattern)`

Text representing the date of an instant in the specified pattern

`text FormatDateTime(InstantInTime instant, text pattern)`

Text representing the date and time of an instant in the specified pattern

`text FormatTime(InstantInTime instant)`

Text representing the 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 in time 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 represented as a number from 1 to 12)

`text MonthName(InstantInTime instant)`

The name of the month

`InstantInTime Now()`

The current instant in time read from phone's clock

`number Second(InstantInTime instant)`

The second of the minute

`number SystemTime()`

The phone's internal time

`number Weekday(InstantInTime instant)`

The day of the week represented as 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

LOCATIONSENSOR

Non-visible component providing location information, including longitude, latitude, altitude (if supported by the device), and address. This can also perform "geocoding", converting a given address (not necessarily the current one) to a latitude (with the `LatitudeFromAddress` method) and a longitude (with the`LongitudeFromAddress` method).
In order to function, the component must have its `Enabled` property set to True, and the device must have location sensing enabled through wireless networks or GPS satellites (if outdoors).

Location information might not be immediately available when an app starts. You'll have to wait a short time for a location provider to be found and used, or wait for the OnLocationChanged event

PROPERTIES

`ACCURACY`

`ALTITUDE`

`AVAILABLEPROVIDERS`

`CURRENTADDRESS`

`DistanceInterval`

Determines the minimum distance interval, in meters, that the sensor will try to use for sending out location updates. For example, if this is set to 5, then the sensor will fire a LocationChanged event only after 5 meters have been traversed. However, the sensor does not guarantee that an update will be received at exactly the distance interval. It may take more than 5 meters to fire an event, for instance.

`Enabled`

`HASACCURACY`

`HASALTITUDE`

`HASLONGITUDELATITUDE`

`LATITUDE`

`LONGITUDE`

`ProviderLocked`

`ProviderName`

`TimeInterval`

Determines the minimum time interval, in milliseconds, that the sensor will try to use for sending out location updates. However, location updates will only be received when the location of the phone actually changes, and use of the specified time interval is not guaranteed. For example, if 1000 is used as the time interval, location updates will never be fired sooner than 1000ms, but they may be fired anytime after.

EVENTS

`LocationChanged(number latitude, number longitude, number altitude)`

Indicates that a new location has been detected.

`StatusChanged(text provider, text status)`

Indicates that the status of the location provider service has changed, such as when a provider is lost or a new provider starts being used.

METHODS

`number LatitudeFromAddress(text locationName)`

Derives latitude of given address

`number LongitudeFromAddress(text locationName)`

Derives longitude of given address

NEARFIELD

Non-visible component to provide NFC capabilities. For now this component supports the reading and writing of text tags only (if supported by the device)

In order to read and write text tags, the component must have its `ReadMode` property set to True or False respectively.

PROPERTIES

`LASTMESSAGE`

`ReadMode`

`TextToWrite`

`WRITETYPE`

EVENTS

`TagRead(text message)`

Indicates that a new tag has been detected. Currently this is only a plain text tag, as specified in the manifest. See Compiler.java.

`TagWritten()`

Event for TagWritten

METHODS

none

ORIENTATIONSENSOR

Use an orientation sensor component to determine the phone's spatial orientation.

An orientation sensor is a non-visible component that reports the following three values, in degrees:

- **Roll** : 0 degree when the device is level, increasing to 90 degrees as the device is tilted up onto its left side, and decreasing to −90 degrees when the device is tilted up onto its right side.
- **Pitch** : 0 degree when the device is level, increasing to 90 degrees as the device is tilted so its top is pointing down, then decreasing to 0 degree as it gets turned over. Similarly, as the device is tilted so its bottom points down, pitch decreases to −90 degrees, then increases to 0 degree as it gets turned all the way over.
- **Azimuth** : 0 degree when the top of the device is pointing north, 90 degrees when it is pointing east, 180 degrees when it is pointing south, 270 degrees when it is pointing west, etc.

These measurements assume that the device itself is not moving.

PROPERTIES

`Available`

Indicates whether the orientation sensor is present on the Android device.

`Enabled`

If set, the orientation sensor is enabled.

`Azimuth`

Returns the azimuth angle of the device.

`Pitch`

Returns the pitch angle of the device.

`Roll`

Returns the roll angle of the device.

`Magnitude`

Returns a number between 0 and 1 that indicates how much the device is tilted. It gives the magnitude of the force that would be felt by a ball rolling on the surface of the device.

`Angle`

Returns an angle that tells the direction in which the device is tiled. That is, it tells the direction of the force that would be felt by a ball rolling on the surface of the device.

EVENTS

`OrientationChanged(number azimuth, number pitch, number roll)`

Called when the orientation has changed.

PROXIMITYSENSOR

A sensor component that can measure the proximity of an object (in cm) relative to the view screen of a device. This sensor is typically used to determine whether a handset is being held up to a persons ear; i.e. lets you determine how far away an object is from a device. Many devices return the absolute distance, in cm, but some return only near and far values. In this case, the sensor usually reports its maximum range value in the far state and a lesser value in the near state. It reports the following value:

- **Distance**: The distance from the object to the device

PROPERTIES

`Available`

Reports whether or not the device has a proximity sensor

`Enabled`

If enabled, then device will listen for changes in proximity

`KeepRunningWhenOnPause`

If set to true, it will keep sensing for proximity changes even when the app is not visible

`Distance`

Returns the distance from the object to the device

`MaximumRange`

Reports the Maximum Range of the device's ProximitySensor.

EVENTS

`ProximityChanged(number distance)`

Called when distance (in cm) of the object to the device changes.

SOCIAL COMPONENTS - APP INVENTOR FOR ANDROID

TABLE OF CONTENTS

CONTACTPICKER

A button that, when clicked on, displays a list of the contacts to choose among. After the user has made a selection, the following properties will be set to information about the chosen contact:

- `ContactName`: the contact's name
- `EmailAddress`: the contact's primary email address
- `EmailAddressList`: a list of the contact's email addresses
- `PhoneNumber`: the contact's primary phone number (on Later Android Verisons)
- `PhoneNumberList`: a list of the contact's phone numbers (on Later Android Versions)
- `Picture`: the name of the file containing the contact's image, which can be used as a `Picture` property value for the `Image` or `ImageSprite` component.

Other properties affect the appearance of the button (`TextAlignment`, `BackgroundColor`, etc.) and whether it can be clicked on (`Enabled`).

The ContactPicker component might not work on all phones. For example, on Android systems before system 3.0, it cannot pick phone numbers, and the list of email addresses will contain only one email.

PROPERTIES

`BackgroundColor`

Returns the button's background color

`CONTACTNAME`

`EMAILADDRESS`

`Enabled`

`FontBold` (designer only)

`FontItalic` (designer only)

`FontSize` (designer only)

`FontTypeface` (designer only)

`Height`

`Image`

Specifies the path of the button's image. If there is both an Image and a BackgroundColor, only the Image will be visible.

`PICTURE`

`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 as an image as background.

`Text`

`TextAlignment` (designer only)

`TextColor`

`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

`AfterPicking()`

Simple event to be raised after the picker activity returns its result and the properties have been filled in.

`BeforePicking()`

Simple event to raise when the component is clicked but before the picker activity is started.

`GotFocus()`

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

`LostFocus()`

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

METHODS

`Open()`

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

EMAILPICKER

An EmailPicker is a kind of text box. If the user begins entering the name or email address of a contact, the phone will show a dropdown menu of choices that complete the entry. If there are many contacts, the dropdown can take several seconds to appear, and can show intermediate results while the matches are being computed.

The initial contents of the text box and the contents< after user entry is in the `Text` property. If the `Text` property is initially empty, the contents of the `Hint` property will be faintly shown in the text box as a hint to the user.
Other properties affect the appearance of the text box (`TextAlignment`, `BackgroundColor`, etc.) and whether it can be used (`Enabled`).

Text boxes like this are usually used with `Button` components, with the user clicking on the button when text entry is complete.

METHODS

`RequestFocus()`

Sets the EmailPicker active.

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.

`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`

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

`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

none

PHONECALL

A non-visible component that makes a phone call to the number specified in the `PhoneNumber` property, which can be set either in the Designer or Blocks Editor. The component has a `MakePhoneCall` method, enabling the program to launch a phone call.
Often, this component is used with the `ContactPicker` component, which lets the user select a contact from the ones stored on the phone and sets the `PhoneNumber`property to the contact's phone number.
To directly specify the phone number (e.g., 650-555-1212), set the `PhoneNumber` property to a Text with the specified digits (e.g., "6505551212"). Dashes, dots, and parentheses may be included (e.g., "(650)-555-1212") but will be ignored; spaces may not be included.

PROPERTIES

`PhoneNumber`

EVENTS

`IncomingCallAnswered(text phoneNumber)`

Event indicating that an incoming phone call is answered. phoneNumber is the incoming call phone number.

`PhoneCallEnded(number status, text phoneNumber)`

Event indicating that a phone call has ended. If status is 1, incoming call is missed or rejected; if status is 2, incoming call is answered before hanging up; if status is 3, outgoing call is hung up. phoneNumber is the ended call phone number.

`PhoneCallStarted(number status, text phoneNumber)`

Event indicating that a phonecall has started. If status is 1, incoming call is ringing; if status is 2, outgoing call is dialled. phoneNumber is the incoming/outgoing phone number.

METHODS

`MakePhoneCall()`

Makes a phone call using the number in the PhoneNumber property.

PHONENUMBERPICKER

A button that, when clicked on, displays a list of the contacts' phone numbers to choose among. After the user has made a selection, the following properties will be set to information about the chosen contact:

- `ContactName`: the contact's name
- `PhoneNumber`: the contact's phone number
- `EmailAddress`: the contact's email address
- `Picture`: the name of the file containing the contact's image, which can be used as a `Picture` property value for the `Image` or `ImageSprite` component.

Other properties affect the appearance of the button (`TextAlignment`, `BackgroundColor`, etc.) and whether it can be clicked on (`Enabled`).

The PhoneNumberPicker component may not work on all Android devices. For example, on Android systems before system 3.0, the returned lists of phone numbers and email addresses will be empty.

PROPERTIES

`BackgroundColor`

Returns the button's background color

`CONTACTNAME`

`EMAILADDRESS`

`Enabled`

`FontBold` (designer only)

`FontItalic` (designer only)

`FontSize` (designer only)

`FontTypeface` (designer only)

`Height`

`Image`

Specifies the path of the button's image. If there is both an Image and a BackgroundColor, only the Image will be visible.

`PHONENUMBER`

`PICTURE`

`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 as an image as background.

`Text`

`TextAlignment` (designer only)

`TextColor`

`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

`AfterPicking()`

Simple event to be raised after the picker activity returns its result and the properties have been filled in.

`BeforePicking()`

Simple event to raise when the component is clicked but before the picker activity is started.

`GotFocus()`

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

`LostFocus()`

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

METHODS

`Open()`

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

SHARING

Sharing is a non-visible component that enables sharing files and/or messages between your app and other apps installed on a device. The component will display a list of the installed apps that can handle the information provided, and will allow the user to choose one to share the content with, for instance a mail app, a social network app, a texting app, and so on.
The file path can be taken directly from other components such as the Camera or the ImagePicker, but can also be specified directly to read from storage. Be aware that different devices treat storage differently, so a few things to try if, for instance, you have a file called arrow.gif in the folder `Appinventor/assets`, would be:

- `"file:///sdcard/Appinventor/assets/arrow.gif"`

or

- `"/storage/Appinventor/assets/arrow.gif"`

PROPERTIES

none

EVENTS

none

METHODS

`ShareFile(text file)`

Shares a file through any capable application installed on the phone by displaying a list of the available apps and allowing the user to choose one from the list. The selected app will open with the file inserted on it.

`ShareFileWithMessage(text file, text message)`

Shares both a file and a message through any capable application installed on the phone by displaying a list of available apps and allowing the user to choose one from the list. The selected app will open with the file and message inserted on it.

`ShareMessage(text message)`

Shares a message through any capable application installed on the phone by displaying a list of the available apps and allowing the user to choose one from the list. The selected app will open with the message inserted on it.

TEXTING

A component that will, when the `SendMessage` method is called, send the text message specified in the `Message` property to the phone number specified in the`PhoneNumber` property.
If the `ReceivingEnabled` property is set to 1 messages will **not** be received. If `ReceivingEnabled` is set to 2 messages will be received only when the application is running. Finally if `ReceivingEnabled` is set to 3, messages will be received when the application is running **and** when the application is not running they will be queued and a notification displayed to the user.
When a message arrives, the `MessageReceived` event is raised and provides the sending number and message.

An app that includes this component will receive messages even when it is in the background (i.e. when it's not visible on the screen) and, moreso, even if the app is not running, so long as it's installed on the phone. If the phone receives a text message when the app is not in the foreground, the phone will show a notification in the notification bar. Selecting the notification will bring up the app. As an app developer, you'll probably want to give your users the ability to control ReceivingEnabled so that they can make the phone ignore text messages.

If the GoogleVoiceEnabled property is true, messages can be sent over Wifi using Google Voice. This option requires that the user have a Google Voice account and that the mobile Voice app is installed on the phone. The Google Voice option works only on phones that support Android 2.0 (Eclair) or higher.

To specify the phone number (e.g., 650-555-1212), set the `PhoneNumber` property to a Text string with the specified digits (e.g., 6505551212). Dashes, dots, and parentheses may be included (e.g., (650)-555-1212) but will be ignored; spaces may not be included.
Another way for an app to specify a phone number would be to include a `PhoneNumberPicker` component, which lets the users select a phone numbers from the ones stored in the the phone's contacts.

PROPERTIES

`GoogleVoiceEnabled`

If true, then SendMessage will attempt to send messages over Wifi using Google Voice. This requires that the Google Voice app must be installed and set up on the phone or tablet, with a Google Voice account. If GoogleVoiceEnabled is false, the device must have phone and texting service in order to send or receive messages with this component.

`Message`

The message that will be sent when the SendMessage method is called.

`PhoneNumber`

The number that the message will be sent to when the SendMessage method is called. The number is a text string with the specified digits (e.g., 6505551212). Dashes, dots, and parentheses may be included (e.g., (650)-555-1212) but will be ignored; spaces should not be included.

`ReceivingEnabled`

If set to 1 (OFF) no messages will be received. If set to 2 (FOREGROUND) or3 (ALWAYS) the component will respond to messages if it is running. If theapp is not running then the message will be discarded if set to 2(FOREGROUND). If set to 3 (ALWAYS) and the app is not running the phone willshow a notification. Selecting the notification will bring up the appand signal the MessageReceived event. Messages received when the appis dormant will be queued, and so several MessageReceived events mightappear when the app awakens. As an app developer, it would be a good idea to give your users control over this property, so they can maketheir phones ignore text messages when your app is installed.

EVENTS

`MessageReceived(text number, text messageText)`

Event that's raised when a text message is received by the phone.

METHODS

`SendMessage()`

Send a text message

TWITTER

A non-visible component that enables communication with Twitter. Once a user has logged into their Twitter account (and the authorization has been confirmed successful by the `IsAuthorized` event), many more operations are available:

- Searching Twitter for tweets or labels (`SearchTwitter`)
- Sending a Tweet (`Tweet`)
- Sending a Tweet with an Image (`TweetWithImage`)
- Directing a message to a specific user (`DirectMessage`)
- Receiving the most recent messages directed to the logged-in user (`RequestDirectMessages`)
- Following a specific user (`Follow`)
- Ceasing to follow a specific user (`StopFollowing`)
- Getting a list of users following the logged-in user (`RequestFollowers`)
- Getting the most recent messages of users followed by the logged-in user (`RequestFriendTimeline`)
- Getting the most recent mentions of the logged-in user (`RequestMentions`)

You must obtain a Consumer Key and Consumer Secret for Twitter authorization specific to your app from http://twitter.com/oauth_clients/new

PROPERTIES

`ConsumerKey`

The the consumer key to be used when authorizing with Twitter via OAuth.

`ConsumerSecret`

The consumer secret to be used when authorizing with Twitter via OAuth

`DIRECTMESSAGES`

This property contains a list of the most recent messages mentioning the logged-in user. Initially, the list is empty. To set it, the program must:

1. Call the `Authorize` method.
2. Wait for the `Authorized` event.
3. Call the `RequestDirectMessages` method.
4. Wait for the `DirectMessagesReceived` event.

The value of this property will then be set to the list of direct messages retrieved (and maintain that value until any subsequent call to `RequestDirectMessages`).

`FOLLOWERS`

This property contains a list of the followers of the logged-in user. Initially, the list is empty. To set it, the program must:

1. Call the `Authorize` method.
2. Wait for the `IsAuthorized` event.
3. Call the `RequestFollowers` method.
4. Wait for the `FollowersReceived` event.

The value of this property will then be set to the list of followers (and maintain its value until any subsequent call to `RequestFollowers`).

`FRIENDTIMELINE`

This property contains the 20 most recent messages of users being followed. Initially, the list is empty. To set it, the program must:

1. Call the `Authorize` method.
2. Wait for the `IsAuthorized` event.
3. Specify users to follow with one or more calls to the `Follow` method.
4. Call the `RequestFriendTimeline` method.
5. Wait for the `FriendTimelineReceived` event.

The value of this property will then be set to the list of messages (and maintain its value until any subsequent call to `RequestFriendTimeline`.

`MENTIONS`

This property contains a list of mentions of the logged-in user. Initially, the list is empty. To set it, the program must:

1. Call the `Authorize` method.
2. Wait for the `IsAuthorized` event.
3. Call the `RequestMentions` method.
4. Wait for the `MentionsReceived` event.

The value of this property will then be set to the list of mentions (and will maintain its value until any subsequent calls to `RequestMentions`).

`SEARCHRESULTS`

This property, which is initially empty, is set to a list of search results after the program:

1. Calls the `SearchTwitter` method.
2. Waits for the `SearchSuccessful` event.

The value of the property will then be the same as the parameter to `SearchSuccessful`. Note that it is not necessary to call the `Authorize` method before calling`SearchTwitter`.

`USERNAME`

The user name of the authorized user. Empty if there is no authorized user.

EVENTS

`DirectMessagesReceived(list messages)`

This event is raised when the recent messages requested through `RequestDirectMessages` have been retrieved. A list of the messages can then be found in the`messages` parameter or the `Messages` property.

`FollowersReceived(list followers2)`

This event is raised when all of the followers of the logged-in user requested through `RequestFollowers` have been retrieved. A list of the followers can then be found in the `followers` parameter or the `Followers` property.

`FriendTimelineReceived(list timeline)`

This event is raised when the messages requested through `RequestFriendTimeline` have been retrieved. The `timeline` parameter and the `Timeline` property will contain a list of lists, where each sub-list contains a status update of the form (username message)

`IsAuthorized()`

This event is raised after the program calls `Authorize` if the authorization was successful. It is also called after a call to `CheckAuthorized` if we already have a valid access token. After this event has been raised, any other method for this component can be called.

`MentionsReceived(list mentions)`

This event is raised when the mentions of the logged-in user requested through `RequestMentions` have been retrieved. A list of the mentions can then be found in the `mentions` parameter or the `Mentions` property.

`SearchSuccessful(list searchResults)`

This event is raised when the results of the search requested through `SearchSuccessful` have been retrieved. A list of the results can then be found in the `results`parameter or the `Results` property.

METHODS

`Authorize()`

Redirects user to login to Twitter via the Web browser using the OAuth protocol if we don't already have authorization.

`CheckAuthorized()`

Checks whether we already have access, and if so, causes IsAuthorized event handler to be called.

`DeAuthorize()`

Removes Twitter authorization from this running app instance

`DirectMessage(text user, text message)`

This sends a direct (private) message to the specified user. The message will be trimmed if it exceeds 160characters.

Requirements: This should only be called after the `IsAuthorized` event has been raised, indicating that the user has successfully logged in to Twitter.

`Follow(text user)`

Starts following a user.

`RequestDirectMessages()`

Requests the 20 most recent direct messages sent to the logged-in user. When the messages have been retrieved, the system will raise the`DirectMessagesReceived` event and set the `DirectMessages` property to the list of messages.

Requirements: This should only be called after the `IsAuthorized` event has been raised, indicating that the user has successfully logged in to Twitter.

`RequestFollowers()`

Gets who is following you.

`RequestFriendTimeline()`

Gets the most recent 20 messages in the user's timeline.

`RequestMentions()`

Requests the 20 most recent mentions of the logged-in user. When the mentions have been retrieved, the system will raise the `MentionsReceived` event and set the `Mentions` property to the list of mentions.

Requirements: This should only be called after the `IsAuthorized` event has been raised, indicating that the user has successfully logged in to Twitter.

`SearchTwitter(text query)`

This searches Twitter for the given String query.

Requirements: This should only be called after the `IsAuthorized` event has been raised, indicating that the user has successfully logged in to Twitter.

`StopFollowing(text user)`

Stops following a user.

`Tweet(text status)`

This sends a tweet as the logged-in user with the specified Text, which will be trimmed if it exceeds 160 characters.

Requirements: This should only be called after the `IsAuthorized` event has been raised, indicating that the user has successfully logged in to Twitter.

`TweetWithImage(text status, text imagePath)`

This sends a tweet as the logged-in user with the specified Text and a path to the image to be uploaded, which will be trimmed if it exceeds 160 characters. If an image is not found or invalid, the update will not be sent.

Requirements: This should only be called after the `IsAuthorized` event has been raised, indicating that the user has successfully logged in to Twitter.

STORAGE - APP INVENTOR FOR ANDROID

TABLE OF CONTENTS

FILE

Non-visible component for storing and retrieving files. Use this component to write or read files on your device. The default behavior is to write files to the private data directory associated with your App. The Companion writes files to /sdcard/AppInventor/data for easy debugging. If the file path starts with a slash (/), then the file is created relative to /sdcard. For example, writing a file to /myFile.txt will write the file in /sdcard/myFile.txt.

PROPERTIES

none

EVENTS

`GotText(text text)`

Event indicating that the contents from the file have been read.

METHODS

`AppendToFile(text text, text fileName)`

Appends text to the end of a file. Creates the file if it does not already exist. See the help text under SaveFile for information about where files are written.

`Delete(text fileName)`

Deletes a file from storage. Prefix the filename with / to delete a specific file in the SD card (for example, /myFile.txt will delete the file /sdcard/myFile.txt). If the filename does not begin with a /, then the file located in the program's private storage will be deleted. Starting the file with // is an error because asset files cannot be deleted.

`ReadFrom(text fileName)`

Reads text from a file in storage. Prefix the filename with / to read from a specific file on the SD card (for example, /myFile.txt will read the file /sdcard/myFile.txt). To read assets packaged with an application (also works for the Companion) start the filename with // (two slashes). If a filename does not start with a slash, it will be read from the application's private storage (for packaged apps) and from /sdcard/AppInventor/data for the Companion.

`SaveFile(text text, text fileName)`

Saves text to a file. If the filename begins with a slash (/) the file is written to the sdcard (for example, writing to /myFile.txt will write the file to /sdcard/myFile.txt). If the filename does not start with a slash, it will be written in the program's private data directory where it will not be accessible to other programs on the phone. There is a special exception for the AI Companion where these files are written to /sdcard/AppInventor/data to facilitate

debugging. Note that this block will overwrite a file if it already exists. If you want to add content to a file use the append block.

FUSIONTABLESCONTROL

A non-visible component that communicates with Google Fusion Tables. Fusion Tables let you store, share, query and visualize data tables; this component lets you query, create, and modify these tables.

This component uses the Fusion Tables API V1.0.

Applications using Fusion Tables must authenticate with Google's servers. There are two ways this can be done. The first way only uses an API Key which you (the developer) obtain. With this approach end-users must also login to access a Fusion Table.

The second approach is to use Service Authentication. With this approach you create credentials and a special "Service Account Email Address" which allows end-users to use your Fusion Tables without logging in; your service account authenticates all access.

USING THE FUSIONTABLESCONTROL COMPONENT

CREATING FUSION TABLES

You will probably want to create your own Fusion Tables to experiment with as you are developing your apps. This is as easy as creating a Google document, if you are familiar with that process. Here are the steps:

1. On the web, login to your Gmail account or any other Google service (e.g., Drive, YouTube). Navigate to Google Drive.
2. Click the *New* button and navigate to *More*. If you do not see a Google Fusion Tables option, select *Connect more apps* and scroll through the page of Google services to find the *Fusion Tables* service and connect it to your Google Drive.
3. You may want to view some of the examples and work through a tutorial (e.g. Pizza Party Tutorial) to learn the basics.
4. Click the *See my tables* button (top right of the page). This will bring you to your own page.
5. You should see a list of your own tables or tables that have been shared with you (possibly none).
6. Use the *Create* button to create a new table. Give it some column names and save it.
7. Click on the *Share* button (top right) to modify the table's permissions.

CREATING A FUSIONTABLES APP

When you drag the *FusiontablesControl* component onto the Designer, don't forget to set its *ApiKey* property, which is initially blank. You should copy this from yourGoogle Developers Console and paste it into the property field.
To get an **API key**, follow these instructions.

1. Go to your Google Developers Console and login if necessary.
2. Under *APIs & auth* select the *APIs* item from the menu on the left.
3. Choose the *Fusion Tables API* from the list provided and turn it on.
4. On the left bar, select the *Credentials* item.
5. Under *Public API access* click *Create new Key*, choose *Android key* and click *Create* to generate an API key.

Your API key(s) will appear in the pane next to "Public API access". You must provide that key as the value for the *ApiKey* property in your Fusion Tables app.

Once you have an API key, set the value of the *Query* property to a valid Fusiontables SQL query and call *SendQuery* to execute the query. App Inventor will send the query to the Fusion Tables server and the *GotResult* block will fire when a result is returned from the server. Query results will be returned in CSV format, and can be converted to list format using the "list from csv table" or "list from csv row" blocks.
Note that you do not need to worry about UTF-encoding the query. But you do need to make sure the query follows the syntax described in the reference manual, which means that things like capitalization for names of columns matters, and that single quotes must be used around column names if there are spaces in them.
To set up **Service Authentication** for your Fusion Table, follow these additional steps:

1. In the Google APIs Console under *APIs & auth* select the *APIs* item from the menu on the left.
2. Click the *Create new Client ID* button. Select the *Service account* option, and click *Create Client ID*.
3. A file called the *KeyFile* (ends in extension .p12) will automatically download onto your computer. Save it in a place you will remember. Once the creation is complete, you will get a table with your Service Account information.
4. In the designer window of App Inventor, select the FusionTablesControl. In the properties pane, add the *ServiceAccountEmail* (from the table on the console), upload the KeyFile, and check the *UseServiceAuthentication* box.
5. Share the Fusion Table with your ServiceAccountEmail, and give it editing permissions, just like you would share any other Google Doc with an email address.

PROPERTIES

`ApiKey`

Your Google API Key. See above for details on obtaining an API key.

`KeyFile`

Specifies the path of the private key file. This key file is used to get access to the FusionTables API through Service Authentication.

`Query`

The query to send to the Fusion Tables API.

For legal query formats and examples, see the Fusion Tables API v1.0 reference manual.

Note that you do not need to worry about UTF-encoding the query. You must make sure the query follows the syntax described in the reference manual. Note that capitalization for names of columns is necessary and that single quotes must be used around column names if there are spaces in them.

`ServiceAccountEmail`

The Service Account Email Address used for Service Authentication.

`UseServiceAuthentication`

Indicates whether a service account should be used for authentication.

EVENTS

`GotResult(text result)`

Indicates that the Fusion Tables query has finished processing and returned with a result. The result of the query will generally be returned in CSV format, and can be converted to list format using the "list from csv table" or "list from csv row" blocks.

METHODS

`DoQuery()`

DEPRECATED. This block is deprecated as of the end of 2012. Use SendQuery instead.

`ForgetLogin()`

Forget the end-user's login credentials. Has no effect on Service Authentication.

`GetRows(text tableId, text columns)`

Gets all the rows from a specified Fusion Table. The tableId field (required) is the id of the Fusion Table. The columns field is a comma-separated list of the columns to retrieve.

`GetRowsWithConditions(text tableId, text columns, text conditions)`

Gets all the rows from a Fusion Table that meet certain conditions. The tableId field (required) is the id of the Fusion Table. The columns field is a comma-separated list of the columns to retrieve. The conditions field specifies what rows to retrieve from the table (for example, the rows in which a particular column value is not null).

`InsertRow(text tableId, text columns, text values)`

Inserts a row into the specified Fusion Table. The tableId field is the id of the Fusion Table. The columns field is a comma-separated list of the columns into which to insert values. The values field specifies what values to insert into each column.

`SendQuery()`

Send the query to the Fusion Tables server.

TINYDB

TinyDB is a non-visible component that stores data for an app.

Apps created with App Inventor are initialized each time they run. This means that 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. In contrast, TinyDB is a PERSISTENT data store for the app. The data stored in a TinyDB will be available each time the app is run. An example might be a game that saves the high score and retrieves it each time the game is played. Data items are strings stored under TAGS. To store a data item, you specify the tag it should be stored under. Subsequently, you can retrieve the data that was stored under a given tag.
Each app has its own data store. There is only one data store per app. Even if you have multiple TinyDB components, they will use the same data store. To get the effect of separate stores, use different keys. You cannot use the TinyDB to pass data between two different apps on the phone, although you CAN use the TinyDB to share data between the different screens of a multi-screen app.

When you are developing apps using the AI Companion, all the apps using that Companion will share the same TinyDB. That sharing will disappear once the apps are packaged and installed on the phone. During development you should be careful to clear the Companion app's data each time you start working on a new app.

PROPERTIES

none

EVENTS

none

METHODS

`ClearAll()`

Clear the entire data store in the TinyDB.

`ClearTag(text tag)`

Clear the entry with the given tag.

`any GetTags()`

Return a list of all the tags in the TinyDB.

`any GetValue(text tag, any valueIfTagNotThere)`

Retrieve the value stored under the given tag. If there's no such tag, then return valueIfTagNotThere.

`StoreValue(text tag, any valueToStore)`

Store the value under the given tag. The storage persists on the phone when the app is restarted.

TINYWEBDB

Non-visible component that communicates with a Web service to store and retrieve information.

See Creating a Custom TinyWebDB Service.

PROPERTIES

`ServiceURL`

The URL to the database with which the component should communicate.

EVENTS

`GotValue(text tagFromWebDB, any valueFromWebDB)`

Indicates that a GetValue server request has succeeded.

`ValueStored()`

Event indicating that a StoreValue server request has succeeded.

`WebServiceError(text message)`

Indicates that the communication with the Web service signaled an error.

METHODS

`GetValue(text tag)`

Sends a request to the Web service to get the value stored under the given tag. The Web service must decide what to return if there is no value stored under the tag. This component accepts whatever is returned.

`StoreValue(text tag, any valueToStore)`

Sends a request to the Web service to store the given value under the given tag.

CONNECTIVITY COMPONENTS - APP INVENTOR FOR ANDROID

TABLE OF CONTENTS

ACTIVITYSTARTER

A component that can launch an activity using the `StartActivity` method.

Activities that can be launched include:

- Starting another App Inventor for Android app. To do so, first find out the CLASS of the other application by downloading the source code and using a file explorer or unzip utility to find a file named "youngandroidproject/project.properties". The first line of the file will start with "main=" and be followed by the class name; for example, `main=com.gmail.Bitdiddle.Ben.HelloPurr.Screen1`. (The first components indicate that it was created by Ben.Bitdiddle@gmail.com.) To make your `ActivityStarter` launch this application, set the following properties:
  - `ActivityPackage` to the class name, dropping the last component (for example, `com.gmail.Bitdiddle.Ben.HelloPurr`)
  - `ActivityClass` to the entire class name (for example, `com.gmail.Bitdiddle.Ben.HelloPurr.Screen1`)
- Starting the camera application by setting the following properties:
  - `Action: android.intent.action.MAIN`
  - `ActivityPackage: com.android.camera`
  - `ActivityClass: com.android.camera.Camera`
- Performing web search. Assuming the term you want to search for is "vampire" (feel free to substitute your own choice), set the properties to:
  - `Action: android.intent.action.WEB_SEARCH`
  - `ExtraKey: query`
  - `ExtraValue: vampire`
  - `ActivityPackage: com.google.android.providers.enhancedgooglesearch`
  - `ActivityClass: com.google.android.providers.enhancedgooglesearch.Launcher`
- Opening a browser to a specified web page. Assuming the page you want to go to is "www.facebook.com" (feel free to substitute your own choice), set the properties to:
  - `Action: android.intent.action.VIEW`
  - `DataUri: http://www.facebook.com`

PROPERTIES

`Action`

`ActivityClass`

`ActivityPackage`

`DataType`

`DataUri`

`ExtraKey`

`ExtraValue`

`RESULT`

`ResultName`

`RESULTTYPE`

`RESULTURI`

EVENTS

`AfterActivity(text result)`

Event raised after this ActivityStarter returns.

`ActivityCanceled()`

Event raised if this ActivityStarter returns because the activity was canceled.

METHODS

`text ResolveActivity()`

Returns the name of the activity that corresponds to this ActivityStarter, or an empty string if no corresponding activity can be found.

`StartActivity()`

Start the activity corresponding to this ActivityStarter.

BLUETOOTHCLIENT

Bluetooth client component

PROPERTIES

`ADDRESSESANDNAMES`

The addresses and names of paired Bluetooth devices

`AVAILABLE`

Whether Bluetooth is available on the device

`CharacterEncoding`

`DelimiterByte`

`ENABLED`

Whether Bluetooth is enabled

`HighByteFirst`

`ISCONNECTED`

`Secure`

Whether to invoke SSP (Simple Secure Pairing), which is supported on devices with Bluetooth v2.1 or higher. When working with embedded Bluetooth devices, this property may need to be set to False. For Android 2.0-2.2, this property setting will be ignored.

EVENTS

none

METHODS

`number BytesAvailableToReceive()`

Returns an estimate of the number of bytes that can be received without blocking

`boolean Connect(text address)`

Connect to the Bluetooth device with the specified address and the Serial Port Profile (SPP). Returns true if the connection was successful.

`boolean ConnectWithUUID(text address, text uuid)`

Connect to the Bluetooth device with the specified address and UUID. Returns true if the connection was successful.

`Disconnect()`

Disconnect from the connected Bluetooth device.

`boolean IsDevicePaired(text address)`

Checks whether the Bluetooth device with the specified address is paired.

`number ReceiveSigned1ByteNumber()`

Receive a signed 1-byte number from the connected Bluetooth device.

`number ReceiveSigned2ByteNumber()`

Receive a signed 2-byte number from the connected Bluetooth device.

`number ReceiveSigned4ByteNumber()`

Receive a signed 4-byte number from the connected Bluetooth device.

`list ReceiveSignedBytes(number numberOfBytes)`

Receive multiple signed byte values from the connected Bluetooth device. If numberOfBytes is less than 0, read until a delimiter byte value is received.

`text ReceiveText(number numberOfBytes)`

Receive text from the connected Bluetooth device. If numberOfBytes is less than 0, read until a delimiter byte value is received.

`number ReceiveUnsigned1ByteNumber()`

Receive an unsigned 1-byte number from the connected Bluetooth device.

`number ReceiveUnsigned2ByteNumber()`

Receive a unsigned 2-byte number from the connected Bluetooth device.

`number ReceiveUnsigned4ByteNumber()`

Receive a unsigned 4-byte number from the connected Bluetooth device.

`list ReceiveUnsignedBytes(number numberOfBytes)`

Receive multiple unsigned byte values from the connected Bluetooth device. If numberOfBytes is less than 0, read until a delimiter byte value is received.

`Send1ByteNumber(text number)`

Send a 1-byte number to the connected Bluetooth device.

`Send2ByteNumber(text number)`

Send a 2-byte number to the connected Bluetooth device.

`Send4ByteNumber(text number)`

Send a 4-byte number to the connected Bluetooth device.

`SendBytes(list list)`

Send a list of byte values to the connected Bluetooth device.

`SendText(text text)`

Send text to the connected Bluetooth device.

BLUETOOTHSERVER

Bluetooth server component

PROPERTIES

`Available: boolean`

Tell whether Bluetooth is available on the Android device.

`CharacterEncoding: text`

The character encoding to use when sending and receiving text.

`DelimiterByte: number`

The delimiter byte to use when passing a negative number for the numberOfBytes parameter when calling ReceiveText, ReceiveSignedBytes, or ReceiveUnsignedBytes.

`Enabled: boolean`

Tell whether Bluetooth is enabled.

`HighByteFirst: boolean`

Whether 2 and 4 byte numbers should be sent and received with the high (or most significant) byte first. Check the documentation for the device with which your app will be communicating for the appropriate setting. This is also known as big-endian.

`IsAccepting: boolean`

Tell whether this BluetoothServer component is accepting an incoming connection.

`IsConnected: boolean`

Tell whether a Bluetooth connection has been made.

EVENTS

`ConnectionAccepted()`

Indicates that a bluetooth connection has been accepted.

METHODS

`AcceptConnection(text serviceName)`

Accept an incoming connection with the Serial Port Profile (SPP).

`AcceptConnectionWithUUID(text serviceName, text uuid)`

Accept an incoming connection with a specific UUID.

`number BytesAvailableToReceive()`

Returns an estimate of the number of bytes that can be received without blocking

`Disconnect()`

Disconnect from the connected Bluetooth device.

`number ReceiveSigned1ByteNumber()`

Receive a signed 1-byte number from the connected Bluetooth device.

`number ReceiveSigned2ByteNumber()`

Receive a signed 2-byte number from the connected Bluetooth device.

`number ReceiveSigned4ByteNumber()`

Receive a signed 4-byte number from the connected Bluetooth device.

`list ReceiveSignedBytes(number numberOfBytes)`

Receive multiple signed byte values from the connected Bluetooth device. If numberOfBytes is less than 0, read until a delimiter byte value is received.

`text ReceiveText(number numberOfBytes)`

Receive text from the connected Bluetooth device. If numberOfBytes is less than 0, read until a delimiter byte value is received.

`number ReceiveUnsigned1ByteNumber()`

Receive an unsigned 1-byte number from the connected Bluetooth device.

`number ReceiveUnsigned2ByteNumber()`

Receive a unsigned 2-byte number from the connected Bluetooth device.

`number ReceiveUnsigned4ByteNumber()`

Receive a unsigned 4-byte number from the connected Bluetooth device.

`list ReceiveUnsignedBytes(number numberOfBytes)`

Receive multiple unsigned byte values from the connected Bluetooth device. If numberOfBytes is less than 0, read until a delimiter byte value is received.

`Send1ByteNumber(text number)`

Send a 1-byte number to the connected Bluetooth device.

`Send2ByteNumber(text number)`

Send a 2-byte number to the connected Bluetooth device.

`Send4ByteNumber(text number)`

Send a 4-byte number to the connected Bluetooth device.

`SendBytes(list list)`

Send a list of byte values to the connected Bluetooth device.

`SendText(text text)`

Send text to the connected Bluetooth device.

`StopAccepting()`

Stop accepting an incoming connection.

WEB

Non-visible component that provides functions for HTTP GET, POST, PUT, and DELETE requests.

PROPERTIES

`AllowCookies`

Whether the cookies from a response should be saved and used in subsequent requests. Cookies are only supported on Android version 2.3 or greater.

`RequestHeaders`

The request headers, as a list of two-element sublists. The first element of each sublist represents the request header field name. The second element of each sublist represents the request header field values, either a single value or a list containing multiple values.

`ResponseFileName`

The name of the file where the response should be saved. If SaveResponse is true and ResponseFileName is empty, then a new file name will be generated.

`SaveResponse`

Whether the response should be saved in a file.

`Url`

The URL for the web request.

EVENTS

`GotFile(text url, number responseCode, text responseType, text fileName)`

Event indicating that a request has finished.

`GotText(text url, number responseCode, text responseType, text responseContent)`

Event indicating that a request has finished.

METHODS

`text BuildRequestData(list list)`

Converts a list of two-element sublists, representing name and value pairs, to a string formatted as application/x-www-form-urlencoded media type, suitable to pass to PostText.

`ClearCookies()`

Clears all cookies for this Web component.

`Delete()`

Performs an HTTP DELETE request using the Url property and retrieves the response.
If the SaveResponse property is true, the response will be saved in a file and the GotFile event will be triggered. The ResponseFileName property can be used to specify the name of the file.
If the SaveResponse property is false, the GotText event will be triggered.

`Get()`

Performs an HTTP GET request using the Url property and retrieves the response.
If the SaveResponse property is true, the response will be saved in a file and the GotFile event will be triggered. The ResponseFileName property can be used to specify the name of the file.
If the SaveResponse property is false, the GotText event will be triggered.

`text HtmlTextDecode(text htmlText)`

Decodes the given HTML text value. HTML character entities such as &, <, >, ', and " are changed to &, <, >, ', and ". Entities such as &#xhhhh, and &#nnnn are changed to the appropriate characters.

`any JsonTextDecode(text jsonText)`

Decodes the given JSON encoded value to produce a corresponding AppInventor value. A JSON list [x, y, z] decodes to a list (x y z), A JSON object with name A and value B, (denoted as A:B enclosed in curly braces) decodes to a list ((A B)), that is, a list containing the two-element list (A B).

`PostFile(text path)`

Performs an HTTP POST request using the Url property and data from the specified file.
If the SaveResponse property is true, the response will be saved in a file and the GotFile event will be triggered. The ResponseFileName property can be used to specify the name of the file.
If the SaveResponse property is false, the GotText event will be triggered.

`PostText(text text)`

Performs an HTTP POST request using the Url property and the specified text.
The characters of the text are encoded using UTF-8 encoding.
If the SaveResponse property is true, the response will be saved in a file and the GotFile event will be triggered. The responseFileName property can be used to specify the name of the file.
If the SaveResponse property is false, the GotText event will be triggered.

`PostTextWithEncoding(text text, text encoding)`

Performs an HTTP POST request using the Url property and the specified text.
The characters of the text are encoded using the given encoding.
If the SaveResponse property is true, the response will be saved in a file and the GotFile event will be triggered. The ResponseFileName property can be used to specify the name of the file.
If the SaveResponse property is false, the GotText event will be triggered.

`PutFile(text path)`

Performs an HTTP PUT request using the Url property and data from the specified file.
If the SaveResponse property is true, the response will be saved in a file and the GotFile event will be triggered. The ResponseFileName property can be used to specify the name of the file.
If the SaveResponse property is false, the GotText event will be triggered.

`PutText(text text)`

Performs an HTTP PUT request using the Url property and the specified text.
The characters of the text are encoded using UTF-8 encoding.
If the SaveResponse property is true, the response will be saved in a file and the GotFile event will be triggered. The responseFileName property can be used to specify the name of the file.
If the SaveResponse property is false, the GotText event will be triggered.

`PutTextWithEncoding(text text, text encoding)`

Performs an HTTP PUT request using the Url property and the specified text.
The characters of the text are encoded using the given encoding.
If the SaveResponse property is true, the response will be saved in a file and the GotFile event will be triggered. The ResponseFileName property can be used to specify the name of the file.
If the SaveResponse property is false, the GotText event will be triggered.

`text UriEncode(text text)`

Encodes the given text value so that it can be used in a URL.

`any XMLTextDecode(text XmlText)`

Decodes the given XML string to produce a list structure. See the App Inventor documentation on "Other topics, notes, and details" for information.