User Interface Components - App Inventor for Android

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

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>, <br>, <cite>, <dfn>, <div>, <em>, <small>, <strong>, <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.

`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.