Mobile accessibility

- Accessibility on mobile devices
- Summary of screen reader testing on Android and iOS
  - Android TalkBack
    - Unlocking the phone
    - Global and local menus
    - Browsing web pages
  - iOS VoiceOver
    - Unlock phone
    - Using the Rotor
    - Browsing web pages
- Control mechanisms
- Responsive design
  - Specific mobile considerations
    - Not disabling zoom
    - Keeping menus accessible
- User input
- Summary
- See also

With web access on mobile devices being so popular and renowned platforms such as iOS and Android having full-fledged accessibility tools, it is important to consider the accessibility of your web content on these platforms. This article looks at mobile-specific accessibility considerations.

Prerequisites:	Familiarity with HTML, CSS, and accessibility best practices as taught in previous lessons in the module..

Learning outcomes:	• Familiarity with screen readers on iOS and Android. • Familiarity with accessibility issues behind some types of events. • Specific techniques for more usable user input mechanisms on mobile. • Know that mobile browsers provide specific usability advantages for specific `<input>` types such as `number` or `tel`.

Accessibility on mobile devices

The state of accessibility — and support for web standards in general — is good in modern mobile devices. Long gone are the days when mobile devices ran completely different web technologies to desktop browsers, forcing developers to use browser sniffing and serve them completely separate sites (although quite a few companies still detect usage of mobile devices and serve them a separate mobile domain).

These days, mobile devices can usually handle fully-featured websites, and the main platforms even have screen readers built in to enable visually impaired users to use them successfully.

Modern mobile browsers tend to have good support for WAI-ARIA, too.

To make a website accessible and usable on mobile, you just need to follow general good web design and accessibility best practices.

There are some exceptions that need special consideration for mobile; the main ones are:

- Control mechanisms — Make sure interface controls such as buttons are accessible on mobiles (i.e., mainly touchscreen), as well as desktops/laptops (mainly mouse/keyboard).
- User input — Make user input requirements as painless as possible on mobile (e.g., in forms, keep typing to a minimum).
- Responsive design — Make sure layouts work on mobile, conserve image download sizes, and think about the provision of images for high-resolution screens.

Summary of screen reader testing on Android and iOS

The most common mobile platforms have fully functional screen readers. These function in much the same way as desktop screen readers, except they are largely operated using touch gestures rather than key combinations.

Let's look at the main two: TalkBack on Android and VoiceOver on iOS.

Android TalkBack

The TalkBack screen reader is built into the Android operating system.

To turn it on, look up what phone model and Android version you have, and then look up where the TalkBack menu is. It tends to differ widely between Android versions and even between different phone models. Some phone manufacturers (e.g., Samsung) don't even have TalkBack in newer phones, and instead opted for their own screen reader.

When you've found the TalkBack menu, press the slider switch to turn TalkBack on. Follow any additional on-screen prompts that you are presented with.

When TalkBack is turned on, your Android device's basic controls will be a bit different. For example:

1. Single-tapping an app will select it, and the device will read out what the app is.
2. Swiping left and right will move between apps, or buttons/controls if you are in a control bar. The device will read out each option.
3. Double-tapping anywhere will open the app/select the option.
4. You can also "explore by touch" — hold your finger down on the screen and drag it around, and your device will read out the different apps/items you move across.

If you want to turn TalkBack off:

1. Navigate back to the TalkBack menu screen (using the different gestures that are currently enabled.)
2. Navigate to the slider switch and activate it to turn it off.

> **NOTE:** You can get to your home screen at any time by swiping up and left in a smooth motion. If you have more than one home screen, you can move between them by swiping two fingers left and right.

For a more complete list of TalkBack gestures, see Use TalkBack gestures.

Unlocking the phone

When TalkBack is turned on, unlocking the phone is a bit different.

You can do a two-finger swipe up from the bottom of the lock screen. If you've set a passcode or pattern for unlocking your device, you will then be taken to the relevant entry screen to enter it.

You can also explore by touch to find the *Unlock* button at the bottom middle of the screen, and then double-tap.

Global and local menus

TalkBack allows you to access global and local context menus, wherever you have navigated on the device. The former provides global options relating to the device as a whole, and the latter provides options relating just to the current app/screen you are in.

To get to these menus:

1. Access the global menu by quickly swiping down, and then right.
2. Access the local menu by quickly swiping up, and then right.
3. Swipe left and right to cycle between the different options.
4. Once you've selected the option you want, double-tap to choose that option.

For details on all the options available under the global and local context menus, see Use global and local context menus.

Browsing web pages

You can use the local context menu while in a web browser to find options to navigate web pages using just the headings, form controls, or links, or navigate line by line, etc.

For example, with TalkBack turned on:

1. Open your web browser.
2. Activate the URL bar.
3. Enter a web page that has a bunch of headings on it, such as the front page of bbc.co.uk. To enter the text of the URL:
   - Select the URL bar by swiping left/right till you get to it, and then double-tapping.
   - Hold your finger down on the virtual keyboard until you get the character you want, and then release your finger to type it. Repeat for each character.
   - Once you've finished, find the Enter key and press it.
4. Swipe left and right to move between different items on the page.
5. Swipe up and right with a smooth motion to enter the local content menu.
6. Swipe right until you find the "Headings and Landmarks" option.
7. Double-tap to select it. Now you'll be able to swipe left and right to move between headings and ARIA landmarks.
8. To go back to the default mode, enter the local context menu again by swiping up and right, select "Default", and then double-tap to activate.

> **NOTE:** See Get started on Android with TalkBack for more complete documentation.

iOS VoiceOver

A mobile version of VoiceOver is built into the iOS operating system.

To turn it on, go to Your *Settings* app and select *Accessibility* > *VoiceOver*. Press the *VoiceOver* slider to enable it (you'll also see several other options related to VoiceOver on this page).

> **NOTE:** Some older iOS devices have the VoiceOver menu at *Settings app* > *General* > *Accessibility* > *VoiceOver*.

Once VoiceOver is enabled, iOS's basic control gestures will be a bit different:

1. A single tap will cause the item you tap on to be selected; your device will speak the item you've tapped on.
2. You can also navigate the items on the screen by swiping left and right to move between them, or by sliding your finger around on the screen to move between different items (when you find the item you want, you can remove your finger to select it).
3. To activate the selected item (e.g., open a selected app), double-tap anywhere on the screen.
4. Swipe with three fingers to scroll through a page.
5. Tap with two fingers to perform a context-relevant action — for example, taking a photo while in the camera app.

To turn it off again, navigate back to *Settings* > *General* > *Accessibility* > *VoiceOver* using the above gestures, and toggle the *VoiceOver* slider back to off.

Unlock phone

To unlock the phone, you need to press the home button (or swipe) as normal. If you have a passcode set, you can select each number by swiping/sliding (as explained above) and then double-tapping to enter each number when you've found the right one.

Using the Rotor

When VoiceOver is turned on, you have a navigation feature called the Rotor available to you, which allows you to quickly choose from a number of common useful options. To use it:

1. Twist two fingers around on the screen like you are turning a dial. Each option will be read aloud as you twist further around. You can go back and forth to cycle through the options.
2. Once you've found the option you want:
   - Release your fingers to select it.
   - If it is an option you can iterate the value of (such as Volume or Speaking Rate), you can do a swipe up or down to increase or decrease the value of the selected item.

The options available under the Rotor are context-sensitive — they will differ depending on what app or view you are in (see below for an example).

Browsing web pages

Let's have a go at web browsing with VoiceOver:

1. Open your web browser.
2. Activate the URL bar.
3. Enter a web page that has a bunch of headings on it, such as the front page of bbc.co.uk. To enter the text of the URL:
   - Select the URL bar by swiping left/right until you get to it, and then double-tapping.
   - For each character, hold your finger down on the virtual keyboard until you get the character you want, and then release your finger to select it. Double-tap to type it.
   - Once you've finished, find the Enter key and press it.
4. Swipe left and right to move between items on the page. You can double-tap an item to select it (e.g., follow a link).
5. By default, the selected Rotor option will be Speaking Rate; you can currently swipe up and down to increase or decrease the speaking rate.
6. Now turn two fingers around the screen like a dial to show the rotor and move between its options. Here are a few examples of the options available:
   - *Speaking Rate*: Change the speaking rate.
   - *Containers*: Move between different semantic containers on the page.
   - *Headings*: Move between headings on the page.
   - *Links*: Move between links on the page.
   - *Form Controls*: Move between form controls on the page.
   - *Language*: Move between different translations, if they are available.
7. Select *Headings*. Now you'll be able to swipe up and down to move between headings on the page.

> **NOTE:** For a more complete reference covering the VoiceOver gestures available and other hints on accessibility testing on iOS, see Apple's VoiceOver documentation.

Control mechanisms

In our CSS and JavaScript accessibility article, we looked at the idea of events that are specific to a certain type of control mechanism (see Mouse-specific events). To recap, these cause accessibility issues because other control mechanisms can't activate the associated functionality.

As an example, the click event is good in terms of accessibility — an associated event handler can be invoked by clicking the element the handler is set on, tabbing to it and pressing Enter/Return, or tapping it on a touchscreen device. Try the following basic button example to see what we mean:

```
<button>Press me!</button>
```

```
html {
  height: 100%;
}

body {
  height: inherit;
  font-family: sans-serif;
  display: flex;
  align-items: center;
}

h1 {
  text-align: center;
}

button {
  width: 70%;
  margin: 0 auto;
  display: block;
  font-size: 150%;
  line-height: 1.5;
}
```

```
const btn = document.querySelector("button");

btn.addEventListener("click", () => {
  alert("Ouch, that hurt!");
```

```
});
```

Mouse-specific events, however, such as mousedown and mouseup, create problems — their event handlers cannot be invoked using non-mouse controls.

The next example uses code like the following to allow you to drag a box around the screen with your mouse:

```
div.addEventListener("mousedown", () => {
  initialBoxX = div.offsetLeft;
  initialBoxY = div.offsetTop;
  movePanel();
});

document.addEventListener("mouseup", stopMove);
```

```
<div></div>
```

```
html {
  font-family: sans-serif;
  overflow: hidden;
}

body {
  background: #ffffee;
  margin: 0;
}

div {
  background-color: #1fe200;
  background-image: linear-gradient(
    to bottom right,
    rgb(0 0 0 / 0),
    rgb(0 0 0 / 0.4)
  );
  width: 200px;
  height: 150px;
  border: 1px solid green;
  position: absolute;
}
```

```
document.body.width = window.innerWidth;
document.body.height = window.innerHeight;

let mouseX, mouseY;

document.addEventListener("mousemove", (e) => {
  mouseX = e.clientX;
  mouseY = e.clientY;
});

const div = document.querySelector("div");

let initialMouseX = null;

let initialMouseY = null;

let initialBoxX, initialBoxY, rAF;

div.addEventListener("mousedown", () => {
  initialBoxX = div.offsetLeft;
  initialBoxY = div.offsetTop;
  movePanel();
});

document.addEventListener("mouseup", stopMove);

function movePanel() {
  if (initialMouseX === null) {
    initialMouseX = mouseX;
    initialMouseY = mouseY;
  } else {
    let mouseMoveX = mouseX - initialMouseX;
    let mouseMoveY = mouseY - initialMouseY;

    let offsetX = initialBoxX + mouseMoveX;
    let offsetY = initialBoxY + mouseMoveY;
    console.log(`${offsetX} ${offsetY}`);

    div.style.left = `${offsetX}px`;
    div.style.top = `${offsetY}px`;
```

```
  }

  rAF = requestAnimationFrame(movePanel);
}

function stopMove() {
  cancelAnimationFrame(rAF);

  console.log("mousemove stopped");

  initialMouseX = null;
  initialMouseY = null;
}
```

However, if you try to drag it with your finger on a touchscreen device, it won't work. To enable other forms of control, you need to use different, yet equivalent events — for example, touch events work on touchscreen devices:

```
div.addEventListener("touchstart", (e) => {
  initialBoxX = div.offsetLeft;
  initialBoxY = div.offsetTop;
  positionHandler(e);
  movePanel();
});

document.addEventListener("touchend", stopMove);
```

```
document.body.width = window.innerWidth;
document.body.height = window.innerHeight;

let posX, posY;

document.addEventListener("mousemove", positionHandler);
document.addEventListener("touchmove", positionHandler);

function positionHandler(e) {
  if (e.clientX && e.clientY) {
    posX = e.clientX;
    posY = e.clientY;
  } else if (e.targetTouches) {
```

```
    posX = e.targetTouches[0].clientX;
    posY = e.targetTouches[0].clientY;
    e.preventDefault();
  }
}

const div = document.querySelector("div");

let initialPosX = null;

let initialPosY = null;

let rAF;

div.addEventListener("mousedown", () => {
  initialBoxX = div.offsetLeft;
  initialBoxY = div.offsetTop;
  movePanel();
});

div.addEventListener("touchstart", (e) => {
  initialBoxX = div.offsetLeft;
  initialBoxY = div.offsetTop;
  positionHandler(e);
  movePanel();
});

document.addEventListener("mouseup", stopMove);
document.addEventListener("touchend", stopMove);

function movePanel() {
  if (initialPosX === null) {
    initialPosX = posX;
    initialPosY = posY;
  } else {
    let posMoveX = posX - initialPosX;
    let posMoveY = posY - initialPosY;

    let offsetX = initialBoxX + posMoveX;
    let offsetY = initialBoxY + posMoveY;
```

```
    div.style.left = `${offsetX}px`;
    div.style.top = `${offsetY}px`;
  }

  rAF = requestAnimationFrame(movePanel);
}

function stopMove() {
  cancelAnimationFrame(rAF);

  initialPosX = null;
  initialPosY = null;
}
```

The updated version will work with both mouse and touch drag:

> **NOTE:** You can also see fully functional examples showing how to implement different control mechanisms at Implementing game control mechanisms.

Responsive design

Responsive design is the practice of making your layouts and other features of your apps dynamically change depending on factors such as screen size and resolution, so they are usable and accessible to users of different device types.

In particular, the most common problems that need to be addressed for mobile are:

- Suitability of layouts for mobile devices. A multi-column layout won't work as well on a narrow screen, for example, and the text size may need to be increased so it is legible. Such issues can be solved by creating a responsive layout using technologies such as media queries, viewport, and flexbox.
- Conserving image sizes downloaded. In general, small-screen devices won't need images that are as large as their desktop counterparts, and they are more likely to be on slow network connections. Therefore, it is wise to serve smaller images to narrow screen devices as appropriate. You can handle this using responsive image techniques.
- Thinking about high resolutions. Many mobile devices have high-resolution screens, and therefore need higher-resolution images so that the display can continue to look crisp and

sharp. Again, you can serve images as appropriate using responsive image techniques. In addition, many image requirements can be fulfilled using the SVG vector images format, which is well-supported across browsers today. SVG has a small file size and will stay sharp regardless of whatever size is being displayed (see Including vector graphics in HTML for more details).

> **NOTE:** We won't provide a full discussion of responsive design techniques here, as they are covered in other places around MDN (see the above links).

Specific mobile considerations

There are other important issues to consider when making sites more accessible on mobile. We have listed a couple here, but we will add more when we think of them.

Not disabling zoom

Using viewport, it is possible to disable zoom. Always ensure resizing is enabled, and set the width to the device's width in the head:

```
<meta name="viewport" content="width=device-width; user-scalable=yes" />
```

You should never set `user-scalable=no` if at all possible — many people rely on zoom to be able to see the content of your website, so taking this functionality away is a really bad idea. There are certain situations where zooming might break the UI; in such cases, if you feel that you need to disable zoom, you should provide some other kind of equivalent, such as a control for increasing the text size in a way that doesn't break your UI.

Keeping menus accessible

Because the screen is so much narrower on mobile devices, it is very common to use media queries and other technologies to make the navigation menu shrink down to a tiny icon at the top of the display — which can be pressed to reveal the menu only if it's needed — when the site is viewed on mobile. This is commonly represented by a "three horizontal lines" icon, and the design pattern is consequently known as a "hamburger menu".

When implementing such a menu, you need to make sure that the control to reveal it is accessible by appropriate control mechanisms (normally touch for mobile), as discussed in Control mechanisms above, and that the rest of the page is moved out of the way or hidden in some way while the menu is being accessed, to avoid confusion with navigating it.

Click here for a good hamburger menu example.

User input

On mobile devices, inputting data tends to be more annoying for users than the equivalent experience on desktop computers. It is more convenient to type text into form inputs using a desktop or laptop keyboard than a touchscreen virtual keyboard or a tiny mobile physical keyboard.

For this reason, it is worth trying to minimize the amount of typing needed. As an example, instead of getting users to fill out their job title each time using a regular text input, you could instead offer a select menu containing the most common options (which also helps with consistency in data entry) and offer an "Other" option that displays a text field to type any outliers into. You can see an example of this idea in action in the following example:

```
<div>
  <label for="job">Job type:</label>
  <select id="job" name="job">
    <option value="">-- select job --</option>
    <option value="butcher">Butcher</option>
    <option value="baker">Baker</option>
    <option value="candle">Candlestick maker</option>
    <option value="other">Other</option>
  </select>
</div>
<div>
  <label for="other-job">Other job:</label>
  <input type="text" name="other-job" id="other-job" />
</div>
```

```
html {
  font-family: sans-serif;
}

div {
  margin-bottom: 10px;
}
```

```
const select = document.querySelector("select");
const other = document.querySelector("input");

other.parentElement.style.display = "none";

select.onchange = function () {
  if (select.value === "other") {
    other.parentElement.style.display = "block";
  } else {
    other.parentElement.style.display = "none";
  }
};
```

It is also worth considering the use of HTML form input types on mobile platforms as they handle them well — both Android and iOS.

For example:

- Types `number`, `tel`, and `email` display suitable virtual keyboards for entering numbers/telephone numbers.
- Types `time` and `date` display suitable pickers for selecting times and dates.

To try these out, see the live examples available at The HTML5 input types.

If you want to provide a different solution for desktops, you could always serve different markup to your mobile devices using feature detection. Check out our feature detection article for more information.

Summary

In this article, we have provided you with some details about common mobile accessibility-specific issues and how to overcome them. We also took you through the usage of the most common screen readers to aid you in accessibility testing.

See also

- Guidelines For Mobile Web Development — A list of articles in *Smashing Magazine* covering different techniques for mobile web design.

- Make your site work on touch devices — Useful article about using touch events to get interactions working on mobile devices.

□□□ #3
□ □□□ □□□ 2026-10-31 11:27:53 UTC
□ cocobook-sync □□□ 2026-11-04 17:32:30 UTC