Accessibility Developer Checklist

In this document

See also

Making an application accessible is about a deep commitment to usability, getting the details right and delighting your users. This document provides a checklist of accessibility requirements, recommendations and considerations to help you make sure your application is accessible. Following this checklist does not guarantee your application is accessible, but it's a good place to start.

Creating an accessible application is not just the responsibility of developers. Involve your design and testing folks as well, and make them are aware of the guidelines for these other stages of development:

- Android Design: Accessibility
- Accessibility Testing Checklist

In most cases, creating an accessible Android application does not require extensive code restructuring. Rather, it means working through the subtle details of how users interact with your application, so you can provide them with feedback they can sense and understand. This checklist helps you focus on the key development issues to get the details of accessibility right.

Accessibility Requirements

The following steps must be completed in order to ensure a minimum level of application accessibility.

1. **Describe user interface controls:** Provide content descriptions for user interface components that do not have visible text, particularly `ImageButton`, `ImageView` and `CheckBox` components. Use the `android:contentDescription` XML layout attribute or the `setContentDescription(CharSequence)` method to provide this information for accessibility services. (Exception: decorative graphics)
2. **Enable focus-based navigation:** Make sure users can navigate your screen layouts using hardware-based or software directional controls (D-pads, trackballs, keyboards and navigation gestures). In a few cases, you may need to make user interface components focusable or change the focus order to be more logical for user actions.
3. **Custom view controls:** If you build custom interface controls for your application, implement accessibility interfaces for your custom views and provide content descriptions. For custom controls that are intended to be compatible with versions of Android back to 1.6, use the Support Library to implement the latest accessibility features.
4. **No audio-only feedback:** Audio feedback must always have a secondary feedback mechanism to support users who are deaf or hard of hearing. For example, a sound alert for the arrival of a message must be accompanied by a system `Notification`, haptic feedback (if available) or other visual alert.
5. **Test:** Test accessibility by navigating your application using directional controls, and using eyes-free navigation with TalkBack enabled. For more accessibility testing information, see the Accessibility Testing Checklist.

Accessibility Recommendations

The following steps are recommended for ensuring the accessibility of your application. If you do not take these actions, it may impact the overall accessibility and quality of your application.

1. **Android Design Accessibility Guidelines:** Before building your layouts, review and follow the accessibility guidelines provided in the Design guidelines.
2. **Framework-provided controls:** Use Android's built-in user interface controls whenever possible, as these components provide accessibility support by default.
3. **Temporary or self-hiding controls and notifications:** Avoid having user interface controls that fade out or disappear after a certain amount of time. If this behavior is important to your application, provide an alternative interface for these functions.

Special Cases and Considerations

The following list describes specific situations where action should be taken to ensure an accessible app. Review this list to see if any of these special cases and considerations apply to your application, and take the appropriate action.

1. **Text field hints:** For `EditText` fields, provide an android:hint attribute *instead* of a content description, to help users understand what content is expected when the text field is empty and allow the contents of the field to be spoken when it is filled.
2. **Custom controls with high visual context:** If your application contains a custom control with a high degree of visual context (such as a calendar control), default accessibility services processing may not provide adequate descriptions for users, and you should consider providing a virtual view hierarchy for your control using `AccessibilityNodeProvider`.
3. **Custom controls and click handling:** If a custom control in your application performs specific handling of user touch interaction, such as listening with `onTouchEvent(MotionEvent)` for `MotionEvent.ACTION_DOWN` and `MotionEvent.ACTION_UP` and treating it as a click event, you must trigger an `AccessibilityEvent` equivalent to a click and provide a way for accessibility services to perform this action for users. For more information, see Handling custom touch events.
4. **Controls that change function:** If you have buttons or other controls that change function during the normal activity of a user in your application (for example, a button that changes from **Play** to **Pause**), make sure you also change the `android:contentDescription` of the button appropriately.
5. **Prompts for related controls:** Make sure sets of controls which provide a single function, such as the `DatePicker`, provide useful audio feedback when an user interacts with the individual controls.
6. **Video playback and captioning:** If your application provides video playback, it must support captioning and subtitles to assist users who are deaf or hard of hearing. Your video playback controls must also clearly indicate if captioning is available for a video and provide a clear way of enabling captions.
7. **Supplemental accessibility audio feedback:** Use only the Android accessibility framework to provide accessibility audio feedback for your app. Accessibility services such as TalkBack should be the only way your application provides accessibility audio prompts to users. Provide the prompting information with a `android:contentDescription` XML layout attribute or dynamically add it using accessibility framework APIs. For example, if your application takes action that you want to announce to a user, such as automatically turning the page of a book, use the `announceForAccessibility(CharSequence)` method to have accessibility services speak this information to the user.
8. **Custom controls with complex visual interactions:** For custom controls that provide complex or non-standard visual interactions, provide a virtual view hierarchy for your control using `AccessibilityNodeProvider` that allows accessibility services to provide a simplified interaction model for the user. If this approach is not feasible, consider providing an alternate view that is accessible.
9. **Sets of small controls:** If you have controls that are smaller than the minimum recommended touch size in your application screens, consider grouping these controls together using a `ViewGroup` and providing a `android:contentDescription` for the group.
10. **Decorative images and graphics:** Elements in application screens that are purely decorative and do not provide any content or enable a user action should not have accessibility content descriptions.