Building Accessibility Services

Topics

Key classes

- `AccessibilityService`
- `AccessibilityServiceInfo`
- `AccessibilityEvent`
- `AccessibilityRecord`
- `AccessibilityNodeInfo`

See also

- Training: Implementing Accessibility

An accessibility service is an application that provides user interface enhancements to assist users with disabilities, or who may temporarily be unable to fully interact with a device. For example, users who are driving, taking care of a young child or attending a very loud party might need additional or alternative interface feedback.

Android provides standard accessibility services, including TalkBack, and developers can create and distribute their own services. This document explains the basics of building an accessibility service.

The ability for you to build and deploy accessibility services was introduced with Android 1.6 (API Level 4) and received significant improvements with Android 4.0 (API Level 14). The Android Support Library was also updated with the release of Android 4.0 to provide support for these enhanced accessibility features back to Android 1.6. Developers aiming for widely compatible accessibility services are encouraged to use the Support Library and develop for the more advanced accessibility features introduced in Android 4.0.

Manifest Declarations and Permissions

Applications that provide accessibility services must include specific declarations in their application manifests to be treated as an accessibility service by the Android system. This section explains the required and optional settings for accessibility services.

Accessibility service declaration

In order to be treated as an accessibility service, you must include a `service` element (rather than the `activity` element) within the `application` element in your manifest. In addition, within the `service` element, you must also include an accessibility service intent filter. For compatiblity with Android 4.1 and higher, the manifest must also protect the service by adding the `BIND_ACCESSIBILITY_SERVICE`

permission to ensure that only the system can bind to it. Here's an example:

```
  <application>
    <service android:name=".MyAccessibilityService"
        android:permission="android.permission.BIND_ACCESSIBILITY_SERVICE"
        android:label="@string/accessibility_service_label">
      <intent-filter>
        <action android:name="android.accessibilityservice.AccessibilityService" />
      </intent-filter>
    </service>
  </application>
```

These declarations are required for all accessibility services deployed on Android 1.6 (API Level 4) or higher.

Accessibility service configuration

Accessibility services must also provide a configuration which specifies the types of accessibility events that the service handles and additional information about the service. The configuration of an accessibility service is contained in the `AccessibilityServiceInfo` class. Your service can build and set a configuration using an instance of this class and `setServiceInfo()` at runtime. However, not all configuration options are available using this method.

Beginning with Android 4.0, you can include a `<meta-data>` element in your manifest with a reference to a configuration file, which allows you to set the full range of options for your accessibility service, as shown in the following example:

```
<service android:name=".MyAccessibilityService">
  ...
  <meta-data
    android:name="android.accessibilityservice"
    android:resource="@xml/accessibility_service_config" />
</service>
```

This meta-data element refers to an XML file that you create in your application's resource directory (`<project_dir>/res/xml/accessibility_service_config.xml`). The following code shows example contents for the service configuration file:

```
<accessibility-service xmlns:android="http://schemas.android.com/apk/res/android"
    android:description="@string/accessibility_service_description"
    android:packageNames="com.example.android.apis"
    android:accessibilityEventTypes="typeAllMask"
    android:accessibilityFlags="flagDefault"
    android:accessibilityFeedbackType="feedbackSpoken"
    android:notificationTimeout="100"
    android:canRetrieveWindowContent="true"
    android:settingsActivity="com.example.android.accessibility.ServiceSettingsActivity"
/>
```

For more information about the XML attributes which can be used in the accessibility service configuration file, follow these links to the reference documentation:

- `android:description`
- `android:packageNames`
- `android:accessibilityEventTypes`
- `android:accessibilityFlags`
- `android:accessibilityFeedbackType`
- `android:notificationTimeout`
- `android:canRetrieveWindowContent`
- `android:settingsActivity`

For more information about which configuration settings can be dynamically set at runtime, see the `AccessibilityServiceInfo` reference documentation.

Registering for Accessibility Events

One of the most important functions of the accessibility service configuration parameters is to allow you to specify what types of accessibility events your service can handle. Being able to specify this information enables accessibility services to cooperate with each other, and allows you as a developer the flexibility to handle only specific events types from specific applications. The event filtering can include the following criteria:

- **Package Names** - Specify the package names of applications whose accessibility events you want your service to handle. If this parameter is omitted, your accessibility service is considered available to service accessibility events for any application. This parameter can be set in the accessibility service configuration files with the `android:packageNames` attribute as a comma-separated list, or set using the `AccessibilityServiceInfo.packageNames` member.
- **Event Types** - Specify the types of accessibility events you want your service to handle. This parameter can be set in the accessibility service configuration files with the `android:accessibilityEventTypes` attribute as a list separated by the `|` character (for example `accessibilityEventTypes="typeViewClicked|typeViewFocused"`), or set using the `AccessibilityServiceInfo.eventTypes` member.

When setting up your accessibility service, carefully consider what events your service is able to handle and only register for those events. Since users can activate more than one accessibility services at a time, your service must not consume events that it is not able to handle. Remember that other services may handle those events in order to improve a user's experience.

> **Note:** The Android framework dispatches accessibility events to more than one accessibility service if the services provide different feedback types. However, if two or more services provide the same feedback type, then only the first registered service receives the event.

AccessibilityService Methods

An accessibility service must extend the `AccessibilityService` class and override the following methods from that class. These methods are presented in the order in which they are called by the Android system, from when the service is started (`onServiceConnected()`), while it is running (`onAccessibilityEvent()`, `onInterrupt()`) to when it is shut down (`onUnbind()`).

- `onServiceConnected()` - (optional) This system calls this method when it successfully connects to your accessibility service. Use this method to do any one-time setup steps for your service, including connecting to user feedback system services, such as the audio manager or device vibrator. If you want to set the configuration of your service at runtime or make one-time adjustments, this is a convenient location from which to call `setServiceInfo()`.
- `onAccessibilityEvent()` - (required) This method is called back by the system when it detects an `AccessibilityEvent` that matches the event filtering parameters specified by your accessibility service. For example, when the user clicks a button or focuses on a user interface control in an application for which your accessibility service is providing feedback. When this happens, the system calls this method, passing the associated `AccessibilityEvent`, which the service can then interpret and use to provide feedback to the user. This method may be called many times over the lifecycle of your service.
- `onInterrupt()` - (required) This method is called when the system wants to interrupt the feedback your service is providing, usually in response to a user action such as moving focus to a different control. This method may be called many times over the lifecycle of your service.
- `onUnbind()` - (optional) This method is called when the system is about to shutdown the accessibility service. Use this method to do any one-time shutdown procedures, including de-allocating user feedback system services, such as the audio manager or device vibrator.

These callback methods provide the basic structure for your accessibility service. It is up to you to decide on how to process data provided by the Android system in the form of `AccessibilityEvent` objects and provide feedback to the user. For more information about getting information from an accessibility event, see the Implementing Accessibility training.

Getting Event Details

The Android system provides information to accessibility services about the user interface interaction through `AccessibilityEvent` objects. Prior to Android 4.0, the information available in an accessibility event, while providing a significant amount of detail about a user interface control selected by the user, offered limited contextual information. In many cases, this missing context information might be critical to understanding the meaning of the selected control.

An example of an interface where context is critical is a calendar or day planner. If the user selects a 4:00 PM time slot in a Monday to Friday day list and the accessibility service announces “4 PM”, but does not announce the weekday name, the day of the month, or the month name, the resulting feedback is confusing. In this case, the context of a user interface control is critical to a user who wants to schedule a meeting.

Android 4.0 significantly extends the amount of information that an accessibility service can obtain about an user interface interaction by composing accessibility events based on the view hierarchy. A view hierarchy is the set of user interface components that contain the component (its parents) and the user interface elements that may be contained by that component (its children). In this way, the Android system can provide much richer detail about accessibility events, allowing accessibility services to provide more useful feedback to users.

An accessibility service gets information about an user interface event through an `AccessibilityEvent` passed by the system to the service's `onAccessibilityEvent()` callback method. This object provides details about the event, including the type of object being acted upon, its descriptive text and other details. Starting in Android 4.0 (and supported in previous releases through the `AccessibilityEventCompat` object in the Support Library), you can obtain additional information about the event using these calls:

- `AccessibilityEvent.getRecordCount()` and `getRecord(int)` - These methods allow you to retrieve the set of `AccessibilityRecord` objects which contributed to the `AccessibilityEvent` passed to you by the system. This level of detail provides more context for the event that triggered your accessibility service.
- `AccessibilityEvent.getSource()` - This method returns an `AccessibilityNodeInfo` object. This object allows you to request view layout hierarchy (parents and children) of the component that originated the accessibility event. This feature allows an accessibility service to investigate the full context of an event, including the content and state of any enclosing views or child views.

> **Important:** The ability to investigate the view hierarchy from an `AccessibilityEvent` potentially exposes private user information to your accessibility service. For this reason, your service must request this level of access through the accessibility service configuration XML file, by including the `canRetrieveWindowContent` attribute and setting it to `true`. If you do not include this setting in your service configuration xml file, calls to `getSource()` fail.

> **Note:** In Android 4.1 (API Level 16) and higher, the `getSource()` method, as well as `AccessibilityNodeInfo.getChild()` and `getParent()`, return only view objects that are considered important for accessibility (views that draw content or respond to user actions). If your service requires all views, it can request them by setting the `flags` member of the service's `AccessibilityServiceInfo` instance to `FLAG_INCLUDE_NOT_IMPORTANT_VIEWS`.

Taking Action for Users

Starting with Android 4.0 (API Level 14), accessibility services can act on behalf of users, including changing the input focus and selecting (activating) user interface elements. In Android 4.1 (API Level 16) the range of actions has been expanded to include scrolling lists and interacting with text fields. Accessibility services can also take global actions, such as navigating to the Home screen, pressing the Back button, opening the notifications screen and recent applications list. Android 4.1 also includes a new type of focus, *Accessibilty Focus*, which makes all visible elements selectable by an accessibility service.

These new capabilities make it possible for developers of accessibility services to create alternative navigation modes such as gesture navigation, and give users with disabilities improved control of their Android devices.

Listening for gestures

Accessibility services can listen for specific gestures and respond by taking action on behalf of a user. This feature, added in Android 4.1 (API Level 16), and requires that your accessibility service request activation of the Explore by Touch feature. Your service can request this activation by setting the `flags` member of the service's `AccessibilityServiceInfo` instance to `FLAG_REQUEST_TOUCH_EXPLORATION_MODE`, as shown in the following example.

```
public class MyAccessibilityService extends AccessibilityService {
    @Override
    public void onCreate() {
        getServiceInfo().flags = AccessibilityServiceInfo.FLAG_REQUEST_TOUCH_EXPLORATION_MODE;
    }
    ...
}
```

Once your service has requested activation of Explore by Touch, the user must allow the feature to be turned on, if it is not already active. When this feature is active, your service receives notification of accessibility gestures through your service's `onGesture()` callback method and can respond by taking actions for the user.

Using accessibility actions

Accessibility services can take action on behalf of users to make interacting with applications simpler and more productive. The ability of accessibility services to perform actions was added in Android 4.0 (API Level 14) and significantly expanded with Android 4.1 (API Level 16).

In order to take actions on behalf of users, your accessibility service must register to receive events from a few or many applications and request permission to view the content of applications by setting the `android:canRetrieveWindowContent` to `true` in the service configuration file. When events are received by your service, it can then retrieve the `AccessibilityNodeInfo` object from the event using `getSource()`. With the `AccessibilityNodeInfo` object, your service can then explore the view hierarchy to determine what action to take and then act for the user using `performAction()`.

```
public class MyAccessibilityService extends AccessibilityService {

    @Override
    public void onAccessibilityEvent(AccessibilityEvent event) {
        // get the source node of the event
        AccessibilityNodeInfo nodeInfo = event.getSource();

        // Use the event and node information to determine
        // what action to take

        // take action on behalf of the user
        nodeInfo.performAction(AccessibilityNodeInfo.ACTION_SCROLL_FORWARD);

        // recycle the nodeInfo object
        nodeInfo.recycle();
    }
    ...
}
```

The `performAction()` method allows your service to take action within an application. If your service needs to perform a global action such as navigating to the Home screen, pressing the Back button, opening the notifications screen or recent applications list, then use the `performGlobalAction()` method.

Using focus types

Android 4.1 (API Level 16) introduces a new type of user interface focus called *Accessibility Focus*. Accessibility services can used this type of focus to select any visible user interface element and act on it. This focus type is different from the more well known *Input Focus*, which determines what on-screen user interface element receives input when a user types characters, presses **Enter** on a keyboard or pushes the center button of a D-pad control.

Accessibility Focus is completely separate and independent from Input Focus. In fact, it is possible for one element in a user interface to have Input Focus while another element has Accessibility Focus. The purpose of Accessibility Focus is to provide accessibility services with a method of interacting with any visible element on a screen, regardless of whether or not the element is input-focusable from a system perspective. You can see accessibility focus in action by testing accessibility gestures. For more information about testing this feature, see Testing gesture navigation.

> **Note:** Accessibility services that use Accessibility Focus are responsible for synchronizing the current Input Focus when an element is capable of this type of focus. Services that do not synchronize Input Focus with Accessibility Focus run the risk of causing problems in

> applications that expect input focus to be in a specific location when certain actions are taken.

An accessibility service can determine what user interface element has Input Focus or Accessibility Focus using the `AccessibilityNodeInfo.findFocus()` method. You can also search for elements that can be selected with Input Focus using the `focusSearch()` method. Finally, your accessibility service can set Accessibility Focus using the `performAction(AccessibilityNodeInfo.ACTION_SET_ACCESSIBILITY_FOCUS)` method.

Example Code

The API Demo project contains two samples which can be used as a starting point for generating accessibility services (`<sdk>/samples/<platform>/ApiDemos/src/com/example/android/apis/accessibility`):

- ClockBackService - This service is based on the original implementation of `AccessibilityService` and can be used as a base for developing basic accessibility services that are compatible with Android 1.6 (API Level 4) and higher.
- TaskBackService - This service is based on the enhanced accessibility APIs introduced in Android 4.0 (API Level 14). However, you can use the Android Support Libary to substitute classes introduced in later API levels (e.g., `AccessibilityRecord`, `AccessibilityNodeInfo`) with equivalent support package classes (e.g., `AccessibilityRecordCompat`, `AccessibilityNodeInfoCompat`) to make this example work with API versions back to Android 1.6 (API Level 4).