# Supportare i controller in tutte le versioni di Android

Se stai supportando i controller di gioco nel tuo gioco, è tua responsabilità assicurarti che il tuo gioco risponda in modo coerente ai controller su dispositivi in esecuzione su diverse versioni di Android. Ciò consente al tuo gioco di raggiungere un pubblico più ampio e i tuoi giocatori possono godere di un'esperienza di gioco senza soluzione di continuità con i loro controller anche quando cambiano o aggiornano i loro dispositivi Android.

Questa lezione illustra come usare le API disponibili in Android 4.1 e versioni precedenti in modo compatibile con le versioni precedenti, consentendo al gioco di supportare le funzionalità seguenti sui dispositivi che eseguono Android 3.1 e versioni precedenti:

* Il gioco può rilevare se un nuovo controller di gioco viene aggiunto, modificato o rimosso.
* Il gioco può interrogare le funzionalità di un controller di gioco.
* Il gioco può riconoscere gli eventi di movimento in arrivo da un controller di gioco.

Gli esempi di questa lezione si basano sull'implementazione di riferimento fornita dall'esempio disponibile per il download precedente. Questo esempio mostra come implementare l'interfaccia per supportare diverse versioni di Android. Per compilare l'esempio, devi usare Android 4.1 (livello API 16) o versione successiva. Una volta compilata, l'app di esempio viene eseguita su qualsiasi dispositivo che esegue Android 3.1 (livello API 12) o superiore come destinazione di compilazione.

## Prepararsi alle API astratte per il supporto dei controller di gioco

Si supponga di voler essere in grado di determinare se lo stato di connessione di un controller di gioco è cambiato nei dispositivi in esecuzione su Android 3.1 (livello API 12). Tuttavia, le API sono disponibili solo in Android 4.1 (livello API 16) e versioni superiori, quindi è necessario fornire un'implementazione che supporti Android 4.1 e versioni superiori fornendo al contempo un meccanismo di fallback che supporti Android 3.1 fino ad Android 4.0.

Per aiutarti a determinare quali funzionalità richiedono un tale meccanismo di fallback per le versioni precedenti, la tabella 1 elenca le differenze nel supporto del controller di gioco tra Android 3.1 (livello API 12) e 4.1 (livello API 16).

**La tabella 1.** API per il supporto del controller di gioco in diverse versioni di Android.

|  |  |  |  |
| --- | --- | --- | --- |
| **Informazioni sul controller** | **Controller API** | **API livello 12** | **API livello 16** |
| Identificazione del dispositivo | getInputDeviceIds() |  | • |
| getInputDevice() |  | • |
| getVibrator() |  | • |
| SOURCE\_JOYSTICK | • | • |
| SOURCE\_GAMEPAD | • | • |
| Stato connessione | onInputDeviceAdded() |  | • |
| onInputDeviceChanged() |  | • |
| onInputDeviceRemoved() |  | • |
| Identificazione evento di input | Pressa D-pad ( , , , , KEYCODE\_DPAD\_UPKEYCODE\_DPAD\_DOWNKEYCODE\_DPAD\_LEFTKEYCODE\_DPAD\_RIGHTKEYCODE\_DPAD\_CENTER) | • | • |
| Pulsante Gamepad ( , , , , , , , , , BUTTON\_ABUTTON\_BBUTTON\_THUMBLBUTTON\_THUMBRBUTTON\_SELECTBUTTON\_STARTBUTTON\_R1BUTTON\_L1BUTTON\_R2BUTTON\_L2) | • | • |
| Movimento joystick e interruttore cappello ( , , , , AXIS\_XAXIS\_YAXIS\_ZAXIS\_RZAXIS\_HAT\_XAXIS\_HAT\_Y) | • | • |
| Pressione del grilletto analogico ( , AXIS\_LTRIGGERAXIS\_RTRIGGER) | • | • |

Puoi usare l'astrazione per creare il supporto del controller di gioco in grado di riconoscere la versione che funziona su più piattaforme. Questo approccio prevede i seguenti passaggi:

1. Definisci un'interfaccia Java intermedia che astrae l'implementazione delle funzionalità del controller di gioco richieste dal tuo gioco.
2. Crea un'implementazione proxy della tua interfaccia che usa API in Android 4.1 e versioni superiori.
3. Crea un'implementazione personalizzata della tua interfaccia che usa API disponibili tra Android 3.1 fino ad Android 4.0.
4. Crea la logica per passare da un'implementazione all'altro in fase di esecuzione e inizia a usare l'interfaccia nel gioco.

Per una panoramica di come l'astrazione può essere utilizzata per garantire che le applicazioni possano funzionare in modo compatibile con le versioni precedenti di Android, vedere Creazione di URI compatibili con le versioni precedenti.

## Aggiungere un'interfaccia per la compatibilità con le versioni precedenti

Per garantire la compatibilità con le versioni precedenti, è possibile creare un'interfaccia personalizzata e quindi aggiungere implementazioni specifiche della versione. Un vantaggio di questo approccio è che ti consente di rispecchiare le interfacce pubbliche su Android 4.1 (livello API 16) che supportano i controller di gioco.

KOTLIN

// The InputManagerCompat interface is a reference example.  
// The full code is provided in the ControllerSample.zip sample.  
interface InputManagerCompat {  
val inputDeviceIds: IntArray  
fun getInputDevice(id: Int): InputDevice  
  
fun registerInputDeviceListener(listener: InputManager.InputDeviceListener,  
handler: Handler?)  
  
fun unregisterInputDeviceListener(listener:InputManager.InputDeviceListener)  
  
fun onGenericMotionEvent(event: MotionEvent)  
  
fun onPause()  
fun onResume()  
  
interface InputDeviceListener {  
 fun onInputDeviceAdded(deviceId: Int)  
 fun onInputDeviceChanged(deviceId: Int)  
 fun onInputDeviceRemoved(deviceId: Int)  
 }  
}

L'interfaccia fornisce i metodi seguenti:InputManagerCompat

getInputDevice()

Specchi. Ottiene l'oggetto che rappresenta le funzionalità di un controller di gioco.getInputDevice()InputDevice

getInputDeviceIds()

Specchi. Restituisce una matrice di numeri interi, ognuno dei quali è un ID per un dispositivo di input diverso. Questo è utile se stai costruendo un gioco che supporta più giocatori e vuoi rilevare quanti controller sono connessi.getInputDeviceIds()

registerInputDeviceListener()

Specchi. Consente di registrarsi per essere informati quando un nuovo dispositivo viene aggiunto, modificato o rimosso.registerInputDeviceListener()

unregisterInputDeviceListener()

Specchi. Annulla la registrazione di un listener di dispositivi di input.unregisterInputDeviceListener()

onGenericMotionEvent()

Specchi. Consente al gioco di intercettare e gestire oggetti e valori dell'asse che rappresentano eventi come movimenti del joystick e presse analogiche.onGenericMotionEvent()MotionEvent

onPause()

Interrompe il polling per gli eventi del controller di gioco quando l'attività principale è in pausa o quando il gioco non ha più lo stato attivo.

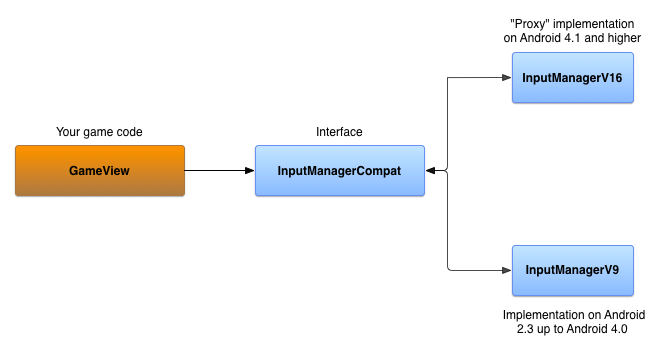
onResume()

Avvia il polling degli eventi del controller di gioco quando l'attività principale viene ripresa o quando il gioco viene avviato ed eseguito in primo piano.

InputDeviceListener

Rispecchia l'interfaccia. Consente al tuo gioco di sapere quando un controller di gioco è stato aggiunto, modificato o rimosso.InputManager.InputDeviceListener

Quindi, crea implementazioni per questo lavoro in diverse versioni della piattaforma. Se il gioco è in esecuzione su Android 4.1 o versione successiva e chiama un metodo, l'implementazione proxy chiama il metodo equivalente in . Tuttavia, se il tuo gioco è in esecuzione su Android 3.1 fino ad Android 4.0, l'implementazione personalizzata elabora le chiamate ai metodi utilizzando solo API introdotte non oltre Android 3.1. Indipendentemente dall'implementazione specifica della versione utilizzata in fase di esecuzione, l'implementazione restituisce i risultati della chiamata in modo trasparente al gioco. InputManagerCompat



**Figura 1.** Diagramma di classe delle implementazioni specifiche dell'interfaccia e della versione.

## Implementare l'interfaccia su Android 4.1 e versioni successiva

InputManagerCompatV16 è un'implementazione dell'interfaccia che proxya le chiamate al metodo a un oggetto effettivo e . L'oggetto è ottenuto dal sistema .InputManagerCompatInputManagerInputManager.InputDeviceListenerInputManagerContext

KOTLIN

// The InputManagerCompatV16 class is a reference implementation.  
// The full code is provided in the ControllerSample.zip sample.  
public class InputManagerV16(context: Context,  
private val inputManager: InputManager = context.getSystemService(Context.INPUT\_SERVICE) as InputManager,  
private val listeners: MutableMap<InputManager.InputDeviceListener, V16InputDeviceListener> = mutableMapOf()) : InputManagerCompat {  
override val inputDeviceIds: IntArray = inputManager.inputDeviceIds  
  
override fun getInputDevice(id: Int): InputDevice = inputManager.getInputDevice(id)  
  
override fun registerInputDeviceListener(  
listener: InputManager.InputDeviceListener,  
handler: Handler?) {  
V16InputDeviceListener(listener).also { v16listener ->  
inputManager.registerInputDeviceListener(v16listener, handler)  
listeners += listener to v16listener  
}  
}  
  
// Do the same for unregistering an input device listener  
...  
  
override fun onGenericMotionEvent(event: MotionEvent) {  
// unused in V16  
}  
  
override fun onPause() {  
// unused in V16  
}  
  
override fun onResume() {  
// unused in V16  
}  
  
}  
  
class V16InputDeviceListener(  
private val idl: InputManager.InputDeviceListener  
) : InputManager.InputDeviceListener {  
  
override fun onInputDeviceAdded(deviceId: Int) {  
idl.onInputDeviceAdded(deviceId)  
}  
// Do the same for device change and removal  
...  
}

## Implementa l'interfaccia su Android 3.1 fino ad Android 4.0

Per creare un'implementazione che supporti Android 3.1 fino ad Android 4.0, puoi usare i seguenti oggetti:

* Uno degli ID dispositivo per tenere traccia dei controller di gioco collegati al dispositivo. SparseArray
* Oggetto per elaborare gli eventi del dispositivo. Quando un'app viene avviata o ripresa, riceve un messaggio per avviare il polling per la disconnessione del controller di gioco. L'oggetto avvia un ciclo per controllare ogni controller di gioco connesso noto e vedere se viene restituito un ID dispositivo. Un valore restituito indica che il controller di gioco è disconnesso. Interrompe il polling quando l'app è in pausa. Handler
* Oggetto di oggetti. Utilizzerai i listener per aggiornare lo stato di connessione dei controller di gioco tracciati.

// The InputManagerCompatV9 class is a reference implementation.  
// The full code is provided in the ControllerSample.zip sample.  
class InputManagerV9(  
 val devices: SparseArray<Array<Long>> = SparseArray(),  
 private val listeners: MutableMap<InputManager.InputDeviceListener, Handler> = mutableMapOf()  
) : InputManagerCompat {  
 private val defaultHandler: Handler = PollingMessageHandler(this)  
…  
}

Implementare un oggetto che estenda ed eseguono l'override del metodo . Questo metodo verifica se un controller di gioco collegato è stato disconnesso e notifica ai listener registrati.PollingMessageHandlerHandlerhandleMessage()

KOTLIN

private class PollingMessageHandler(  
 inputManager: InputManagerV9,  
 private val mInputManager: WeakReference<InputManagerV9> = WeakReference(inputManager)  
) : Handler() {  
  
 override fun handleMessage(msg: Message) {  
 super.handleMessage(msg)  
 when (msg.what) {  
 MESSAGE\_TEST\_FOR\_DISCONNECT -> {  
 mInputManager.get()?.also { imv ->  
 val time = SystemClock.elapsedRealtime()  
 val size = imv.devices.size()  
 for (i in 0 until size) {  
 imv.devices.valueAt(i)?.also { lastContact ->  
 if (time - lastContact[0] > CHECK\_ELAPSED\_TIME) {  
 // check to see if the device has been  
 // disconnected  
 val id = imv.devices.keyAt(i)  
 if (null == InputDevice.getDevice(id)) {  
 // Notify the registered listeners  
 // that the game controller is disconnected  
 imv.devices.remove(id)  
 } else {  
 lastContact[0] = time  
 }  
 }  
 }  
 }  
 sendEmptyMessageDelayed(MESSAGE\_TEST\_FOR\_DISCONNECT, CHECK\_ELAPSED\_TIME)  
 }  
 }  
 }  
 }  
}

Per avviare e interrompere il polling per la disconnessione del controller di gioco, eseguire l'override dei metodi seguenti:

KOTLIN

private const val MESSAGE\_TEST\_FOR\_DISCONNECT = 101  
private const val CHECK\_ELAPSED\_TIME = 3000L  
  
class InputManagerV9(  
 val devices: SparseArray<Array<Long>> = SparseArray(),  
 private val listeners:  
 MutableMap<InputManager.InputDeviceListener, Handler> = mutableMapOf()  
) : InputManagerCompat {  
 ...  
 override fun onPause() {  
 defaultHandler.removeMessages(MESSAGE\_TEST\_FOR\_DISCONNECT)  
 }  
  
 override fun onResume() {  
 defaultHandler.sendEmptyMessageDelayed(MESSAGE\_TEST\_FOR\_DISCONNECT, CHECK\_ELAPSED\_TIME)  
 }  
 ...  
}

Per rilevare che è stato aggiunto un dispositivo di input, eseguire l'override del metodo . Quando il sistema segnala un evento di movimento, verificare se l'evento proviene da un ID dispositivo già monitorato o da un nuovo ID dispositivo. Se l'ID del dispositivo è nuovo, avvisare i listener registrati.onGenericMotionEvent()

KOTLIN

override fun onGenericMotionEvent(event: MotionEvent) {  
 // detect new devices  
 val id = event.deviceId  
 val timeArray: Array<Long> = mDevices.get(id) ?: run {  
 // Notify the registered listeners that a game controller is added  
 ...  
 arrayOf<Long>().also {  
 mDevices.put(id, it)  
 }  
 }  
 timeArray[0] = SystemClock.elapsedRealtime()  
}

La notifica dei listener viene implementata utilizzando l'oggetto per inviare un oggetto alla coda messaggi. Contiene un riferimento a un oggetto . Quando l'oggetto viene eseguito, il metodo di callback appropriato del listener viene chiamato per segnalare se il controller di gioco è stato aggiunto, modificato o rimosso. HandlerDeviceEventRunnableDeviceEventInputManagerCompat.InputDeviceListenerDeviceEvent

KOTLIN

class InputManagerV9(  
 val devices: SparseArray<Array<Long>> = SparseArray(),  
 private val listeners:  
 MutableMap<InputManager.InputDeviceListener, Handler> = mutableMapOf()  
) : InputManagerCompat {  
 ...  
 override fun registerInputDeviceListener(  
 listener: InputManager.InputDeviceListener,  
 handler: Handler?  
 ) {  
 listeners[listener] = handler ?: defaultHandler  
 }  
  
 override fun unregisterInputDeviceListener(listener: InputManager.InputDeviceListener) {  
 listeners.remove(listener)  
 }  
  
 private fun notifyListeners(why: Int, deviceId: Int) {  
 // the state of some device has changed  
 listeners.forEach { listener, handler ->  
 DeviceEvent.getDeviceEvent(why, deviceId, listener).also {  
 handler?.post(it)  
 }  
 }  
 }  
 ...  
}  
  
private val sObjectQueue: Queue<DeviceEvent> = ArrayDeque<DeviceEvent>()  
  
private class DeviceEvent(  
 private var mMessageType: Int,  
 private var mId: Int,  
 private var mListener: InputManager.InputDeviceListener  
) : Runnable {  
  
 companion object {  
 fun getDeviceEvent(messageType: Int, id: Int, listener: InputManager.InputDeviceListener) =  
 sObjectQueue.poll()?.apply {  
 mMessageType = messageType  
 mId = id  
 mListener = listener  
 } ?: DeviceEvent(messageType, id, listener)  
  
 }  
  
 override fun run() {  
 when(mMessageType) {  
 ON\_DEVICE\_ADDED -> mListener.onInputDeviceAdded(mId)  
 ON\_DEVICE\_CHANGED -> mListener.onInputDeviceChanged(mId)  
 ON\_DEVICE\_REMOVED -> mListener.onInputDeviceChanged(mId)  
 else -> {  
 // Handle unknown message type  
 }  
 }  
 }  
  
}

Ora hai due implementazioni di: una che funziona su dispositivi che eseguono Android 4.1 e versioni superiori e un'altra che funziona su dispositivi che eseguono Android 3.1 fino ad Android 4.0.InputManagerCompat

## Utilizzare l'implementazione specifica della versione

La logica di commutazione specifica della versione viene implementata in una classe che funge da factory.

KOTLIN

object Factory {  
 fun getInputManager(context: Context): InputManagerCompat =  
 if (Build.VERSION.SDK\_INT >= Build.VERSION\_CODES.JELLY\_BEAN) {  
 InputManagerV16(context)  
 } else {  
 InputManagerV9()  
 }  
}

Ora puoi semplicemente creare un'istanza di un oggetto e registrare un oggetto nel tuo principale . A causa della logica di cambio versione configurata, il gioco utilizza automaticamente l'implementazione appropriata per la versione di Android in esecuzione del dispositivo.InputManagerCompatInputManagerCompat.InputDeviceListenerView

KOTLIN

class GameView(context: Context) : View(context), InputManager.InputDeviceListener {  
 private val inputManager: InputManagerCompat = Factory.getInputManager(context).apply {  
 registerInputDeviceListener(this@GameView, null)  
 ...  
 }  
 ...  
}

Eseguire quindi l'override del metodo nella visualizzazione principale, come descritto in Handle a MotionEvent from a Game Controller. Il tuo gioco dovrebbe ora essere in grado di elaborare gli eventi del controller di gioco in modo coerente sui dispositivi che eseguono Android 3.1 (livello API 12) e versioni superiori. onGenericMotionEvent()

KOTLIN

override fun onGenericMotionEvent(event: MotionEvent): Boolean {  
 inputManager.onGenericMotionEvent(event)  
  
 // Handle analog input from the controller as normal  
 ...  
 return super.onGenericMotionEvent(event)  
}

È possibile trovare un'implementazione completa di questo codice di compatibilità nella classe fornita nell'esempio disponibile per il download precedente.GameViewControllerSample.zip

**Supportare più controller di gioco**

Scopri come rilevare e utilizzare più controller di gioco connessi contemporaneamente.

# Supportare più controller di gioco

Mentre la maggior parte dei giochi sono progettati per supportare un singolo utente per dispositivo Android, è anche possibile supportare più utenti con controller di gioco connessi contemporaneamente sullo stesso dispositivo Android.

Questa lezione illustra alcune tecniche di base per gestire l'input nel gioco multiplayer a dispositivo singolo da più controller connessi. Ciò include il mantenimento di una mappatura tra gli avatar del giocatore e ogni dispositivo controller e l'elaborazione appropriata degli eventi di input del controller.

## Mappare i lettori agli ID dispositivo controller

Quando un controller di gioco è collegato a un dispositivo Android, il sistema gli assegna un ID dispositivo intero. È possibile ottenere gli ID dispositivo per i controller di gioco connessi chiamando , come illustrato in Verificare che un controller di gioco sia connesso. Puoi quindi associare ogni ID dispositivo a un giocatore nel tuo gioco ed elaborare le azioni di gioco per ogni giocatore separatamente. InputDevice.getDeviceIds()

**Nota:** Nei dispositivi che eseguono Android 4.1 (livello API 16) e versioni superiori, è possibile ottenere il descrittore di un dispositivo di input utilizzando , che restituisce un valore stringa persistente univoco per il dispositivo di input. A differenza di un ID dispositivo, il valore del descrittore non cambierà anche se il dispositivo di input viene disconnesso, riconnesso o riconfigurato. **getDescriptor()**

Il frammento di codice riportato di seguito mostra come usare un oggetto per associare l'avatar di un giocatore a un controller specifico. In questo esempio, la variabile memorizza un insieme di oggetti. Un nuovo avatar giocatore viene creato nel gioco quando un nuovo controller viene collegato da un utente e rimosso quando il controller associato viene rimosso. SparseArraymShipsShip

I metodi di callback e fanno parte del livello di astrazione introdotto in Support Controllers Across Android Versions. Implementando questi callback listener, il gioco può identificare l'ID del dispositivo del controller di gioco quando un controller viene aggiunto o rimosso. Questo rilevamento è compatibile con Android 2.3 (livello API 9) e versioni precedenti. onInputDeviceAdded()onInputDeviceRemoved()

KOTLIN

private val ships = SparseArray<Ship>()  
  
override fun onInputDeviceAdded(deviceId: Int) {  
 getShipForID(deviceId)  
}  
  
override fun onInputDeviceRemoved(deviceId: Int) {  
 removeShipForID(deviceId)  
}  
  
private fun getShipForID(shipID: Int): Ship {  
 return ships.get(shipID) ?: Ship().also {  
 ships.append(shipID, it)  
 }  
}  
  
private fun removeShipForID(shipID: Int) {  
 ships.remove(shipID)  
}

## Elaborare l'input di più controller

Il gioco deve eseguire il seguente ciclo per elaborare l'input da più controller:

1. Rilevare se si è verificato un evento di input.
2. Identificare l'origine di input e il relativo ID dispositivo.
3. In base all'azione indicata dal codice della chiave dell'evento di input o dal valore dell'asse, aggiornare l'avatar del giocatore associato a tale ID dispositivo.
4. Eseguire il rendering e l'aggiornamento dell'interfaccia utente.

KeyEvent e agli eventi di input sono associati ID dispositivo. Il tuo gioco può approfittarne per determinare da quale controller proviene l'evento di input e aggiornare l'avatar del giocatore associato a quel controller. MotionEvent

Il seguente frammento di codice mostra come è possibile ottenere un riferimento avatar del giocatore corrispondente a un ID dispositivo controller di gioco e aggiornare il gioco in base alla pressione del pulsante dell'utente su quel controller.

KOTLIN

override fun onKeyDown(keyCode: Int, event: KeyEvent): Boolean {  
 if (event.source and InputDevice.SOURCE\_GAMEPAD == InputDevice.SOURCE\_GAMEPAD) {  
 event.deviceId.takeIf { it != -1 }?.also { deviceId ->  
 val currentShip: Ship = getShipForID(deviceId)  
 // Based on which key was pressed, update the player avatar  
 // (e.g. set the ship headings or fire lasers)  
 return true  
 }  
 }  
 return super.onKeyDown(keyCode, event)  
}

**Nota:** Come best practice, quando il controller di gioco di un utente si disconnette, è necessario sospendere il gioco e chiedere se l'utente desidera riconnettersi.

# Creare un metodo di input

Un IME (Input Method Editor) è un controllo utente che consente agli utenti di immettere testo. Android fornisce un framework di input-method estensibile che consente alle applicazioni di fornire agli utenti metodi di input alternativi, come tastiere su schermo o persino input vocale. Dopo aver installato le IMI desiderate, un utente può selezionare quale utilizzare dalle impostazioni di sistema e utilizzarlo nell'intero sistema; è possibile abilitare un solo IME alla volta.

Per aggiungere un IME al sistema Android, è necessario creare un'applicazione Android contenente una classe che estende . Inoltre, in genere si crea un'attività "impostazioni" che passa le opzioni al servizio IME. Puoi anche definire un'interfaccia utente delle impostazioni visualizzata come parte delle impostazioni di sistema.

La presente guida tratta quanto segue:

* Ciclo di vita dell'IME
* Dichiarazione di componenti IME nel manifesto dell'applicazione
* The IME API
* Progettazione di un'interfaccia utente IME
* Invio di testo da un IME a un'applicazione
* Utilizzo dei sottotipi IME

Se non hai mai lavorato con gli IME prima, dovresti prima leggere l'articolo introduttivo Metodi di input sullo schermo.

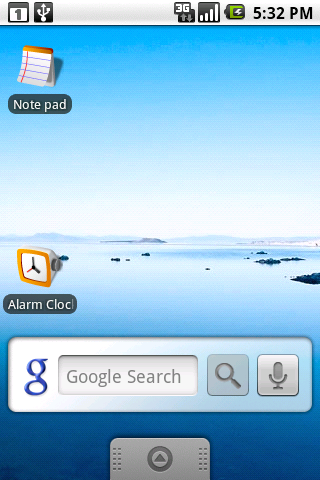
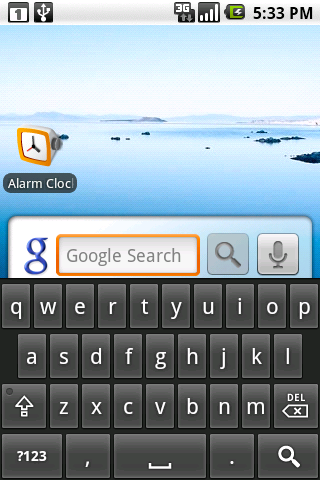
## **Metodi di input su schermo**

A partire da Android 1.5, la piattaforma Android offre un Framework del metodo di input (IMF) che consente di creare metodi di input sullo schermo come le tastiere software. Questo articolo fornisce una panoramica di cosa sono gli editor di metodi di input Android (IMI) e cosa deve fare un'applicazione per funzionare bene con loro. Il FMI è progettato per supportare nuove classi di dispositivi Android, come quelli senza tastiere hardware, quindi è importante che la tua applicazione funzioni bene con il FMI e offra una grande esperienza per gli utenti.

### Che cos'è un metodo di input?

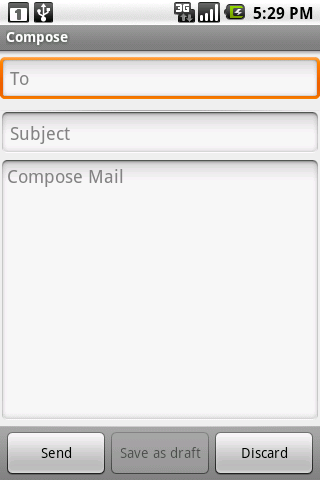
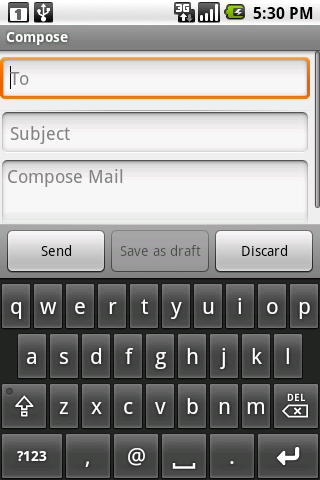
Android IMF è progettato per supportare una varietà di IMI, tra cui tastiera morbida, riconoscimento della scrittura a mano e traduttori hard keyboard. Il nostro focus, tuttavia, sarà sulle tastiere morbide, poiché questo è il tipo di metodo di input che attualmente fa parte della piattaforma.

Un utente accede in genere all'IME corrente toccando una visualizzazione di testo da modificare, come mostrato qui nella schermata iniziale:

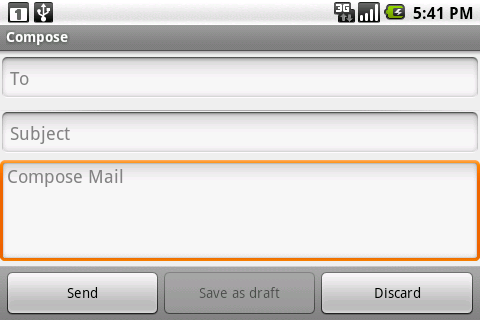
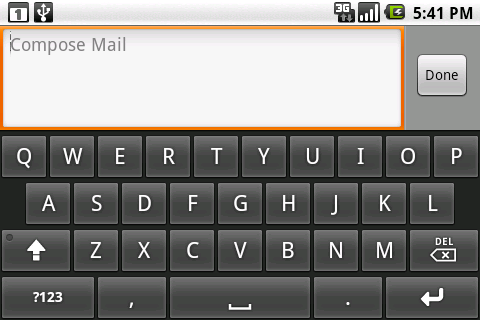
La tastiera morbida è posizionata nella parte inferiore dello schermo sopra la finestra dell'applicazione. Per organizzare lo spazio disponibile tra l'applicazione e l'IME, utilizziamo alcuni approcci; quello mostrato qui è chiamato pan and scane comporta semplicemente lo scorrimento della finestra dell'applicazione in modo che la visualizzazione attualmente focalizzata sia visibile. Questa è la modalità predefinita, poiché è la più sicura per le applicazioni esistenti.

Molto spesso il layout dello schermo preferito è un ridimensionamento , in cui la finestra dell'applicazione viene ridimensionata per essere completamente visibile. Un esempio viene mostrato qui, quando si compone un messaggio di posta elettronica:

Le dimensioni della finestra dell'applicazione vengono modificate in modo che nessuna di essa sia nascosta dall'IME, consentendo l'accesso completo sia all'applicazione che all'IME. Questo naturalmente funziona solo per le applicazioni che hanno un'area ridimensionabile che può essere ridotta per fare abbastanza spazio, ma lo spazio verticale in questa modalità è in realtà non inferiore a quello disponibile nell'orientamento orizzontale, quindi molto spesso un'applicazione può già ospitarlo.

L'ultima modalità principale è la modalità a schermo intero o di estrazione. Questo viene utilizzato quando l'IME è troppo grande per condividere ragionevolmente lo spazio con l'applicazione sottostante. Con gli IE standard, incontrerai questa situazione solo quando lo schermo ha un orientamento orizzontale, anche se altri IMI sono liberi di usarlo ogni volta che lo desiderano. In questo caso la finestra dell'applicazione viene lasciata così com'è e l'IME visualizza semplicemente lo schermo intero sopra di esso, come mostrato qui:

Poiché l'IME copre l'applicazione, ha una propria area di modifica, che mostra il testo effettivamente contenuto nell'applicazione. Ci sono anche alcune opportunità limitate che l'applicazione ha per personalizzare parti dell'IME (il pulsante "fatto" in alto e inserire l'etichetta chiave in basso) per migliorare l'esperienza dell'utente.

### Attributi XML di base per il controllo delle IMI

Ci sono una serie di cose che il sistema fa per cercare di aiutare le applicazioni esistenti a funzionare con gli IMI nel miglior modo possibile, come ad esempio:

* Usa la modalità di panoramica e scansione per impostazione predefinita, a meno che non possa ragionevolmente indovinare che la modalità di ridimensionamento funzionerà in base all'esistenza di elenchi, visualizzazioni di scorrimento e così via.
* Analizza i vari attributi TextView esistenti per indovinare il tipo di contenuto (numeri, testo normale e così via) per aiutare la tastiera soft a visualizzare un layout di tasti appropriato.
* Assegnare alcune azioni predefinite all'IME a schermo intero, ad esempio "campo successivo" e "fatto".

Ci sono anche alcune semplici cose che puoi fare nella tua applicazione che spesso miglioreranno notevolmente la sua esperienza utente. Tranne dove esplicitamente menzionato, questi funzioneranno in qualsiasi versione della piattaforma Android, anche quelli precedenti ad Android 1.5 (poiché ignoreranno semplicemente queste nuove opzioni).

#### Specifica del tipo di input di ogni controllo EditText

La cosa più importante da eseguire per un'applicazione è utilizzare il nuovo attributo in ogni . L'attributo fornisce informazioni molto più dettagliate sul contenuto del testo. Questo attributo sostituisce in realtà molti attributi esistenti (, , , , , e ). Se si specificano gli attributi precedenti e il nuovo attributo, il sistema utilizza e ignora gli altri. android:inputTypeEditTextandroid:passwordandroid:singleLineandroid:numericandroid:phoneNumberandroid:capitalizeandroid:autoTextandroid:editableandroid:inputTypeandroid:inputType

L'attributo ha tre pezzi:android:inputType

* La classe è l'interpretazione complessiva dei caratteri. Le classi attualmente supportate sono (testo normale), (numero decimale), (numero di telefono) e (data o ora).textnumberphonedatetime
* La variazione è un ulteriore perfezionamento della classe. Nell'attributo si specificano normalmente la classe e la variante insieme, con la classe come prefisso. Ad esempio, è un campo di testo in cui l'utente immetterà qualcosa che è un indirizzo di posta elettronica (foo@bar.com) in modo che il layout della chiave abbia un carattere '@' in facile accesso ed è un campo numerico con un segno. Se viene specificata solo la classe, si ottiene la variante predefinita/generica.textEmailAddressnumberSigned
* È possibile specificare flag aggiuntivi che forniscono ulteriore perfezionamento. Questi flag sono specifici di una classe. Ad esempio, alcuni flag per la classe sono , e .texttextCapSentencestextAutoCorrecttextMultiline

Ad esempio, ecco il nuovo EditText per la visualizzazione testo del messaggio dell'applicazione di messaggistica istantanea:

<EditText android:id="@+id/edtInput"

android:layout\_width="0dip"

android:layout\_height="wrap\_content"

android:layout\_weight="1"

android:inputType="textShortMessage|textAutoCorrect|textCapSentences|textMultiLine"

android:imeOptions="actionSend|flagNoEnterAction"

android:maxLines="4"

android:maxLength="2000"

android:hint="@string/compose\_hint"/>

Una descrizione completa di tutti i tipi di input è disponibile nella documentazione. È importante utilizzare i tipi di input corretti disponibili, in modo che la tastiera soft possa utilizzare il layout di tastiera ottimale per il testo che l'utente immetterà.

#### Abilitazione della modalità di ridimensionamento e di altre funzionalità della finestra

La seconda cosa più importante da fare per la tua app è specificare il comportamento complessivo della finestra in relazione al metodo di input. L'aspetto più visibile di questo è il controllo del ridimensionamento rispetto alla modalità di panoramica e scansione, ma ci sono anche altre cose che puoi fare per migliorare la tua esperienza utente.

In genere si controllerà questo comportamento tramite l'attributo in ogni definizione dell'. Come il tipo di input, ci sono un paio di diversi dati che possono essere specificati qui combinandoli insieme:android:windowSoftInputMode<activity>AndroidManifest.xml

* La modalità di regolazione della finestra viene specificata con o . Si consiglia vivamente di specificare sempre l'uno o l'altro.adjustResizeadjustPan
* È possibile controllare ulteriormente se l'IME verrà visualizzato automaticamente quando viene visualizzata l'attività e altre situazioni in cui l'utente si sposta su di essa. Il sistema non mostrerà automaticamente un IME per impostazione predefinita, ma in alcuni casi può essere conveniente per l'utente se un'applicazione abilita questo comportamento. È possibile richiederlo con . Ci sono anche una serie di altre opzioni di stato per un controllo più fine che puoi trovare nella documentazione.stateVisible

Un esempio tipico di questo campo può essere visualizzato nell'attività di modifica del contatto, che garantisce che venga ridimensionato e visualizza automaticamente l'IME per l'utente:

<activity name="EditContactActivity"

android:windowSoftInputMode="stateVisible|adjustResize">

...

</activity>

**Nota:** A partire da Android 1.5 (API Level 3), la piattaforma offre un nuovo metodo, {@link android.view.Window#setSoftInputMode(modalità int)}, che le finestre non attività possono utilizzare per controllarne il comportamento. Chiamare questo metodo nel tuo renderà la tua applicazione incompatibile con le versioni precedenti della piattaforma Android.

#### Controllo dei pulsanti di azione

La personalizzazione finale che remo a guardare sono i pulsanti "azione" nell'IME. Attualmente esistono due tipi di azioni:

* Il tasto INVIO su una tastiera soft è in genere associato a un'azione quando non si opera su un testo di modifica a riga mult. Ad esempio, sul G1 premendo il tasto hard enter si sposterà in genere al campo successivo o l'applicazione lo intercetterà per eseguire un'azione; con una tastiera morbida, questo overload del tasto INVIO rimane, poiché il pulsante invio invia solo un evento tasto di ingresso.
* In modalità a schermo intero, un IME può anche mettere un pulsante di azione aggiuntivo a destra del testo in fase di modifica, dando all'utente un rapido accesso a un'operazione comune dell'applicazione.

Queste opzioni vengono controllate con l'attributo su . Il valore che fornisi qui può essere qualsiasi combinazione di:android:imeOptionsTextView

* Una delle costanti di azione predefinita (, , , , ). Se nessuno di questi è specificato, il sistema dedurrà o a seconda che ci sia un campo messa a fuoco dopo questo; è possibile forzare esplicitamente nessuna azione con .actionGoactionSearchactionSendactionNextactionDoneactionNextactionDoneactionNone
* L'opzione indica all'IME che l'azione non deve essere disponibile sul tasto INVIO, anche se il testo stesso non è multi-riga. In questo modo si evita di avere azioni irrecuperabili come (inviare) che possono essere toccate accidentalmente dall'utente durante la digitazione.flagNoEnterAction
* Rimuove il pulsante di azione dall'area di testo, lasciando più spazio per il testo.flagNoAccessoryAction
* Rimuove completamente l'area di testo, consentendo all'applicazione di essere vista dietro di essa.flagNoExtractUi

La visualizzazione del messaggio dell'applicazione di messaggistica istantanea precedente fornisce anche un esempio di utilizzo interessante di , per specificare l'azione di invio, ma non lasciarla visualizzare nella chiave invio:imeOptions

android:imeOptions="actionSend|flagNoEnterAction"

### API per il controllo delle IMI

Per un controllo più avanzato sull'IME, è possibile utilizzare una varietà di nuove API. A meno che non venga presa particolare attenzione (ad esempio usando la riflessione), l'utilizzo di queste API causerà l'incompatibilità dell'applicazione con le versioni precedenti di Android e dovresti assicurarti di specificare nel manifesto. Per ulteriori informazioni, vedere la documentazione relativa all'elemento manifesto <uses-sdk>.android:minSdkVersion="3"

L'API primaria è la nuova classe, che è possibile recuperare con . Consente di interagire con lo stato del metodo di input globale, ad esempio nascondendo o mostrando esplicitamente l'area di input dell'IME corrente.android.view.inputmethod.InputMethodManagerContext.getSystemService()

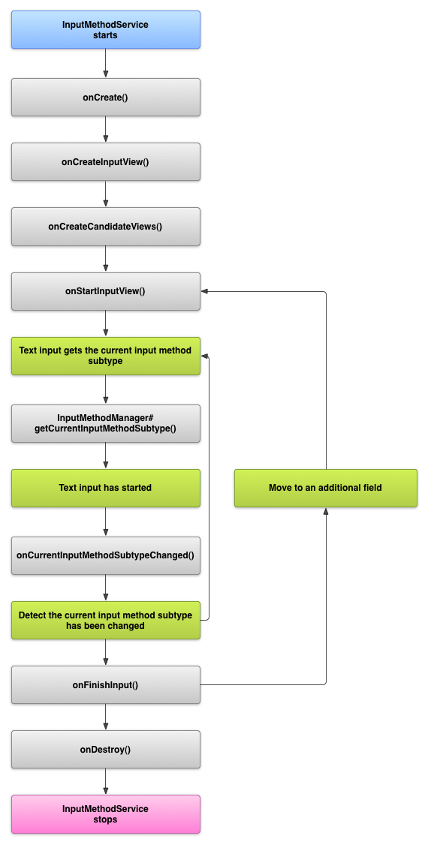
Esistono anche nuovi flag di finestra che controllano l'interazione del metodo di input, che è possibile controllare tramite il metodo esistente e il nuovo metodo. La classe ha aumentato i metodi corrispondenti per controllare queste opzioni nella relativa finestra. Una cosa in particolare di cui essere consapevoli è la nuova costante, che viene utilizzata per controllare se una finestra si trova sopra o dietro l'IME corrente.Window.addFlags()Window.setSoftInputMode()PopupWindowWindowManager.LayoutParams.FLAG\_ALT\_FOCUSABLE\_IM

La maggior parte dell'interazione tra un IME attivo e un'applicazione viene eseguita attraverso la classe . Si tratta dell'API implementa da un'applicazione, chiamata da un IME per eseguire le operazioni di modifica appropriate sull'applicazione. Normalmente non dovrai preoccuparti di questo, poiché fornisce la propria implementazione per se stesso.android.view.inputmethod.InputConnectionTextView

Ci sono anche una manciata di nuove API, la più importante delle quali è che crea una nuova per un IME (e compila una struttura con il tipo di input, le opzioni IME e altri dati); ancora una volta, la maggior parte degli sviluppatori non dovrà preoccuparsi di questo, dal momento che TextView se ne occupa per te.View onCreateInputConnection()InputConnectionandroid.view.inputmethod.EditorInfo

**Nota:** A partire da Android 11, la piattaforma consente agli IPE di visualizzare i suggerimenti di riempimento automatico in linea, invece di utilizzare un menu a discesa. Per ulteriori informazioni su come i servizi di riempimento automatico possono supportare questa funzionalità, vedere Integrazione del riempimento automatico con le tastiere.

## Ciclo di vita dell'IME

Il diagramma seguente descrive il ciclo di vita di un IME:

**Figura 1.** Ciclo di vita di un IME.

Nelle sezioni seguenti viene descritto come implementare l'interfaccia utente e il codice associati a un IME che segue questo ciclo di vita.

## Dichiarare componenti IME nel manifesto

Nel sistema Android, un IME è un'applicazione Android che contiene uno speciale servizio IME. Il file manifesto dell'applicazione deve dichiarare il servizio, richiedere le autorizzazioni necessarie, fornire un filtro intento che corrisponda all'azione e fornire metadati che definisce le caratteristiche dell'IME. Inoltre, per fornire un'interfaccia di impostazioni che consenta all'utente di modificare il comportamento dell'IME, è possibile definire un'attività "impostazioni" che può essere avviata dalle impostazioni di sistema. action.view.InputMethod

Il frammento seguente dichiara un servizio IME. Richiede l'autorizzazione per consentire al servizio di connettere l'IME al sistema, imposta un filtro intento che corrisponde all'azione e definisce i metadati per l'IME: BIND\_INPUT\_METHODandroid.view.InputMethod

<!-- Declares the input method service -->  
<service android:name="FastInputIME"  
 android:label="@string/fast\_input\_label"  
 android:permission="android.permission.BIND\_INPUT\_METHOD">  
 <intent-filter>  
 <action android:name="android.view.InputMethod" />  
 </intent-filter>  
 <meta-data android:name="android.view.im"  
 android:resource="@xml/method" />  
</service>

Questo frammento successivo dichiara l'attività delle impostazioni per l'IME. Ha un filtro intento per che indica che questa attività è il punto di ingresso principale per l'applicazione IME:ACTION\_MAIN

<!-- Optional: an activity for controlling the IME settings -->  
<activity android:name="FastInputIMESettings"  
 android:label="@string/fast\_input\_settings">  
 <intent-filter>  
 <action android:name="android.intent.action.MAIN"/>  
 </intent-filter>  
</activity>

Puoi anche fornire l'accesso alle impostazioni dell'IME direttamente dalla sua interfaccia utente.

## API del metodo di input

Le classi specifiche delle IMI si trovano nei pacchetti e . La classe è importante per la gestione dei caratteri della tastiera. android.inputmethodserviceandroid.view.inputmethodKeyEvent

La parte centrale di un IME è un componente di servizio, una classe che estende . Oltre a implementare il normale ciclo di vita del servizio, questa classe dispone di callback per fornire l'interfaccia utente dell'IME, gestire l'input dell'utente e fornire testo al campo attualmente attivo. Per impostazione predefinita, la classe fornisce la maggior parte dell'implementazione per la gestione dello stato e della visibilità dell'IME e la comunicazione con il campo di input corrente. InputMethodServiceInputMethodService

Importanti sono anche le seguenti classi:

BaseInputConnection

Definisce il canale di comunicazione da un back all'applicazione che riceve il relativo input. Lo si usa per leggere il testo intorno al cursore, eseguire il commit del testo nella casella di testo e inviare eventi chiave non elaborati all'applicazione. Le applicazioni devono estendere questa classe anziché implementare l'interfaccia di base . InputMethodInputConnection

KeyboardView

Estensione di che esegue il rendering di una tastiera e risponde agli eventi di input dell'utente. Il layout di tastiera è specificato da un'istanza di , che è possibile definire in un file XML. ViewKeyboard

## Progettare l'interfaccia utente del metodo di input

Esistono due elementi visivi principali per un IME: la visualizzazione **di input** e la **visualizzazione** candidati. Devi solo implementare gli elementi rilevanti per il metodo di input che stai progettando.

### Visualizzazione input

La visualizzazione di input è l'interfaccia utente in cui l'utente immette testo sotto forma di clic con tasti, grafia o gesti. Quando l'IME viene visualizzato per la prima volta, il sistema chiama il callback. Nell'implementazione di questo metodo si crea il layout che si desidera visualizzare nella finestra IME e si restituisce il layout al sistema. Questo frammento è un esempio di implementazione del metodo: onCreateInputView()onCreateInputView()

KOTLIN

override fun onCreateInputView(): View {  
 return layoutInflater.inflate(R.layout.input, null).apply {  
 if (this is MyKeyboardView) {  
 setOnKeyboardActionListener(this@MyInputMethod)  
 keyboard = latinKeyboard  
 }  
 }  
}

In questo esempio, è un'istanza di un'implementazione personalizzata di che esegue il rendering di un oggetto . MyKeyboardViewKeyboardViewKeyboard

### Visualizzazione Candidati

La visualizzazione candidati è l'interfaccia utente in cui l'IME visualizza potenziali correzioni di parole o suggerimenti da selezionare per l'utente. Nel ciclo di vita dell'IME, il sistema chiama quando è pronto per visualizzare la visualizzazione dei candidati. Nell'implementazione di questo metodo, restituire un layout che mostra i suggerimenti di parole o restituire null se non si desidera mostrare nulla. Una risposta null è il comportamento predefinito, quindi non è necessario implementarlo se non si forniscono suggerimenti.onCreateCandidatesView()

### Considerazioni sulla progettazione dell'interfaccia utente

Questa sezione descrive alcune considerazioni specifiche sulla progettazione dell'interfaccia utente per gli IMI.

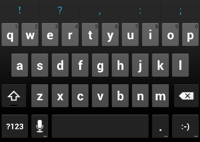
#### Gestire più dimensioni dello schermo

L'interfaccia utente per l'IME deve essere in grado di scalare in base alle diverse dimensioni dello schermo e deve anche gestire gli orientamenti orizzontale e verticale. In modalità IME non a schermo intero, lasciare spazio sufficiente all'applicazione per visualizzare il campo di testo e qualsiasi contesto associato, in modo che non più della metà dello schermo sia occupata dall'IME. In modalità IME a schermo intero questo non è un problema.

#### Gestire diversi tipi di input

I campi di testo android consentono di impostare un tipo di input specifico, ad esempio testo in formato libero, numeri, URL, indirizzi di posta elettronica e stringhe di ricerca. Quando si implementa un nuovo IME, è necessario rilevare il tipo di input di ogni campo e fornire l'interfaccia appropriata. Tuttavia, non è necessario configurare l'IME per verificare che l'utente abbia immesso testo valido per il tipo di input; questa è la responsabilità dell'applicazione che possiede il campo di testo.

Ad esempio, ecco screenshot delle interfacce fornite dall'IME latino con la piattaforma Android per gli input di testo e numeri di telefono:

**Figura 2.** Tipi di input IME latini.

Quando un campo di input riceve lo stato attivo e l'IME viene avviato, il sistema chiama , passando un oggetto contenente dettagli sul tipo di input e su altri attributi del campo di testo. In questo oggetto il campo contiene il tipo di input del campo di testo. onStartInputView()EditorInfoinputType

Il campo è un oggetto che contiene modelli di bit per varie impostazioni del tipo di input. Per testarlo per il tipo di input del campo di testo, mascherarlo con la costante , in questo modo: inputTypeintTYPE\_MASK\_CLASS

KOTLIN

inputType and InputType.TYPE\_MASK\_CLASS

Il modello di bit del tipo di input può avere uno dei diversi valori, tra cui:

TYPE\_CLASS\_NUMBER

Campo di testo per l'immissione di numeri. Come illustrato nella cattura di schermata precedente, l'IME latino visualizza un blocco numerico per campi di questo tipo.

TYPE\_CLASS\_DATETIME

Campo di testo per l'immissione di una data e di un'ora.

TYPE\_CLASS\_PHONE

Campo di testo per l'immissione di numeri di telefono.

TYPE\_CLASS\_TEXT

Campo di testo per l'immissione di tutti i caratteri supportati.

Queste costanti sono descritte più dettagliatamente nella documentazione di riferimento per . InputType

Il campo può contenere altri bit che indicano una variante del tipo di campo di testo, ad esempio: inputType

TYPE\_TEXT\_VARIATION\_PASSWORD

Una variante di per l'immissione di password. Il metodo di input visualizza i dingbat anziché il testo effettivo. TYPE\_CLASS\_TEXT

TYPE\_TEXT\_VARIATION\_URI

Una variante di per l'immissione di URL Web e altri URI (Uniform Resource Identifier). TYPE\_CLASS\_TEXT

TYPE\_TEXT\_FLAG\_AUTO\_COMPLETE

Una variante di per l'immissione di testo che l'applicazione "completa automaticamente" da un dizionario, una ricerca o un'altra funzionalità. TYPE\_CLASS\_TEXT

Ricordarsi di mascherare con la costante appropriata quando si testano queste varianti. Le costanti mask disponibili sono elencate nella documentazione di riferimento per . inputTypeInputType

**Attenzione:** Nel proprio IME, assicurarsi di gestire correttamente il testo quando lo si invia a un campo password. Nascondi la password nell'interfaccia utente sia nella visualizzazione di input che nella visualizzazione candidati. Ricorda anche che non dovresti archiviare le password su un dispositivo. Per ulteriori informazioni, vedere la guida Designing for Security.

## Inviare testo all'applicazione

Quando l'utente immette testo con l'IME, è possibile inviare testo all'applicazione inviando singoli eventi chiave o modificando il testo intorno al cursore nel campo di testo dell'applicazione. In entrambi i casi, si utilizza un'istanza di per consegnare il testo. Per ottenere questa istanza, chiamare . InputConnectionInputMethodService.getCurrentInputConnection()

### Modificare il testo intorno al cursore

Quando si gestisce la modifica del testo esistente in un campo di testo, alcuni dei metodi più utili in sono: BaseInputConnection

getTextBeforeCursor()

Restituisce un oggetto contenente il numero di caratteri richiesti prima della posizione corrente del cursore. CharSequence

getTextAfterCursor()

Restituisce un oggetto contenente il numero di caratteri richiesti dopo la posizione corrente del cursore. CharSequence

deleteSurroundingText()

Elimina il numero specificato di caratteri prima e dopo la posizione corrente del cursore.

commitText()

Eseguire il commit di un oggetto nel campo di testo e impostare una nuova posizione del cursore. CharSequence

Ad esempio, il frammento seguente mostra come sostituire i quattro caratteri a sinistra del cursore con il testo "Hello!":

KOTLIN

currentInputConnection.also { ic: InputConnection ->  
 ic.deleteSurroundingText(4, 0)  
 ic.commitText("Hello", 1)  
 ic.commitText("!", 1)  
}

### Composizione del testo prima del commit

Se l'IME esegue la previsione del testo o richiede più passaggi per comporre un glifo o una parola, è possibile visualizzare lo stato di avanzamento nel campo di testo fino a quando l'utente non esegue il commit della parola, quindi è possibile sostituire la composizione parziale con il testo completato. È possibile fornire un trattamento speciale al testo aggiungendo un "span" ad esso quando lo si passa a . setComposingText()

Il frammento seguente mostra come mostrare lo stato di avanzamento in un campo di testo:

KOTLIN

currentInputConnection.also { ic: InputConnection ->  
 ic.setComposingText("Composi", 1)  
 ic.setComposingText("Composin", 1)  
 ic.commitText("Composing ", 1)  
}

Le schermate seguenti mostrano come appare all'utente:

**Figura 3.** Composizione del testo prima del commit.

### Intercettare gli eventi chiave hardware

Anche se la finestra del metodo di input non ha lo stato attivo esplicito, riceve prima gli eventi chiave hardware e può scegliere di consumarli o inoltrarli all'applicazione. Ad esempio, potresti voler utilizzare i tasti direzionali per navigare all'interno dell'interfaccia utente per la selezione dei candidati durante la composizione. È inoltre possibile intercettare il tasto indietro per chiudere tutti i popup provenienti dalla finestra del metodo di input.

Per intercettare le chiavi hardware, eseguire l'override e . onKeyDown()onKeyUp()

Ricorda di chiamare il metodo per le chiavi che non vuoi gestire da solo. super()

## Creare un sottotipo IME

I sottotipi consentono all'IME di esporre più modalità di input e linguaggi supportati da un IME. Un sottotipo può rappresentare:

* Impostazioni locali, ad esempio en\_US o fr\_FR.
* Modalità di input, ad esempio voce, tastiera o grafia.
* Altri stili di input, moduli o proprietà specifici dell'IME, ad esempio layout di tastiera a 10 tasti o qwerty.

Fondamentalmente, la modalità può essere qualsiasi testo come "tastiera", "voce" e così via. Un sottotipo può anche esporre una combinazione di questi.

Le informazioni sul sottotipo vengono utilizzate per una finestra di dialogo dello switcher IME disponibile dalla barra delle notifiche e anche per le impostazioni IME. Le informazioni consentono inoltre al framework di far emergere direttamente un sottotipo specifico di un IME. Quando si crea un IME, utilizzare la funzione sottotipo, perché consente all'utente di identificare e passare da una lingua e da una modalità IME all'altra.

È possibile definire sottotipi in uno dei file di risorse XML del metodo di input utilizzando l'elemento . Il frammento seguente definisce un IME con due sottotipi: un sottotipo di tastiera per le impostazioni locali inglesi negli Stati Uniti e un altro sottotipo di tastiera per le impostazioni locali della lingua francese per la Francia: <subtype>

<input-method xmlns:android="http://schemas.android.com/apk/res/android"  
 android:settingsActivity="com.example.softkeyboard.Settings"  
 android:icon="@drawable/ime\_icon">  
 <subtype android:name="@string/display\_name\_english\_keyboard\_ime"  
 android:icon="@drawable/subtype\_icon\_english\_keyboard\_ime"  
 android:imeSubtypeLanguage="en\_US"  
 android:imeSubtypeMode="keyboard"  
 android:imeSubtypeExtraValue="somePrivateOption=true" />  
 <subtype android:name="@string/display\_name\_french\_keyboard\_ime"  
 android:icon="@drawable/subtype\_icon\_french\_keyboard\_ime"  
 android:imeSubtypeLanguage="fr\_FR"  
 android:imeSubtypeMode="keyboard"  
 android:imeSubtypeExtraValue="foobar=30,someInternalOption=false" />  
 <subtype android:name="@string/display\_name\_german\_keyboard\_ime" ... />  
</input-method>

Per assicurarsi che i sottotipi siano etichettati correttamente nell'interfaccia utente, utilizzare %s per ottenere un'etichetta di sottotipo uguale all'etichetta delle impostazioni locali del sottotipo. Questo è dimostrato nei prossimi due frammenti. Il primo frammento mostra parte del file XML del metodo di input:

<subtype  
 android:label="@string/label\_subtype\_generic"  
 android:imeSubtypeLocale="en\_US"  
 android:icon="@drawable/icon\_en\_us"  
 android:imeSubtypeMode="keyboard" />

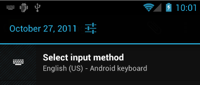
Il frammento successivo fa parte del file dell'IME. La risorsa stringa , utilizzata dalla definizione dell'interfaccia utente del metodo di input per impostare l'etichetta del sottotipo, è definita come: strings.xmllabel\_subtype\_generic

<string name="label\_subtype\_generic">%s</string>

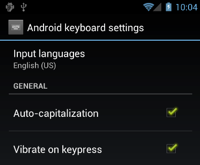
Questa impostazione fa sì che il nome visualizzato del sottotipo corrisponda all'impostazione delle impostazioni locali. Ad esempio, in qualsiasi lingua inglese, il nome visualizzato è "Inglese (Stati Uniti)".

### Scegliere i sottotipi IME dalla barra delle notifiche

Il sistema Android gestisce tutti i sottotipi esposti da tutti gli IMI. I sottotipi IME vengono trattati come modalità dell'IME a cui appartengono. Nella barra di notifica, un utente può selezionare un sottotipo disponibile per l'IME attualmente impostato, come illustrato nella schermata seguente:



**Figura 4.** Scelta di un sottotipo IME dalla barra di notifica.



**Figura 5.** Impostazione delle preferenze del sottotipo in Impostazioni di sistema.

### Scegliere sottotipi IME da Impostazioni di sistema

Un utente può controllare il modo in cui vengono utilizzati i sottotipi nel pannello impostazioni "Lingua e input" nell'area Impostazioni di sistema.



**Figura 6.** Scelta di una lingua per l'IME.

## Passare da un sottotipo IME all'altro

È possibile consentire agli utenti di passare facilmente da più sottotipi IME fornendo un tasto di commutazione, ad esempio l'icona della lingua a forma di globo, come parte della tastiera. Ciò migliora notevolmente l'usabilità della tastiera e può aiutare a evitare la frustrazione dell'utente. Per abilitare tale commutazione, eseguire la procedura seguente:

1. Dichiarare nei file di risorse XML del metodo di input. La dichiarazione dovrebbe essere simile al frammento seguente: supportsSwitchingToNextInputMethod = "true"

<input-method xmlns:android="http://schemas.android.com/apk/res/android"  
 android:settingsActivity="com.example.softkeyboard.Settings"  
 android:icon="@drawable/ime\_icon"  
 android:supportsSwitchingToNextInputMethod="true">

1. Chiamare il metodo .shouldOfferSwitchingToNextInputMethod()
2. Se il metodo restituisce true, visualizzare un tasto di commutazione.
3. Quando l'utente tocca il tasto di commutazione, chiamare , passando false al secondo parametro. Un valore false indica al sistema di trattare tutti i sottotipi allo stesso modo, indipendentemente dall'IME a cui appartengono. Se si specifica true, il sistema deve scorrere i sottotipi nell'IME corrente.switchToNextInputMethod()

**Attenzione:** Prima di Android 5.0 (livello API 21), non è a conoscenza dell'attributo. Se l'utente passa a un IME senza un tasto di commutazione, potrebbe rimanere bloccato in quell'IME, incapace di spegnerlo facilmente.**switchToNextInputMethod()supportsSwitchingToNextInputMethod**

## Considerazioni generali sull'IME

Ecco alcune altre cose da considerare quando stai implementando il tuo IME:

* Offri agli utenti un modo per impostare le opzioni direttamente dall'interfaccia utente dell'IME.
* Poiché nel dispositivo possono essere installati più IME, fornire all'utente un modo per passare a un IME diverso direttamente dall'interfaccia utente del metodo di input.
* Visualizzare rapidamente l'interfaccia utente dell'IME. Precaricare o caricare su richiesta risorse di grandi dimensioni in modo che gli utenti vedano l'IME non appena toccano un campo di testo. Memorizzare nella cache le risorse e le visualizzazioni per le chiamate successive del metodo di input.
* Al contrario, è necessario rilasciare allocazioni di memoria di grandi dimensioni subito dopo che la finestra del metodo di input è nascosta, in modo che le applicazioni possano disporre di memoria sufficiente per l'esecuzione. Prendere in considerazione l'utilizzo di un messaggio ritardato per rilasciare risorse se l'IME si trova in uno stato nascosto per alcuni secondi.
* Assicurarsi che gli utenti possano immettere il maggior numero possibile di caratteri per la lingua o le impostazioni locali associate all'IME. Ricorda che gli utenti possono utilizzare la punteggiatura nelle password o nei nomi utente, quindi il tuo IME deve fornire molti caratteri diversi per consentire agli utenti di inserire una password e ottenere l'accesso al dispositivo.

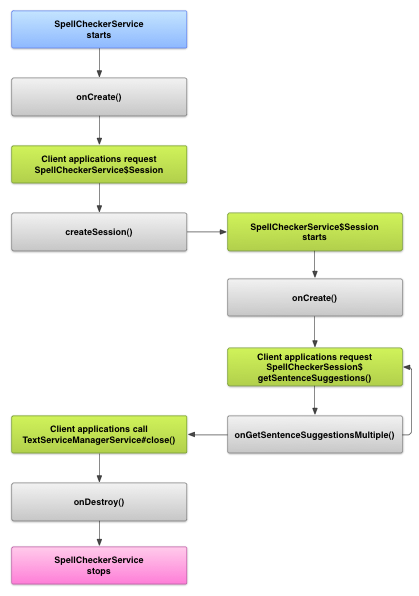
# Framework di controllo ortografico

La piattaforma Android offre un framework di controllo ortografico che ti consente di implementare e accedere al controllo ortografico nella tua applicazione. Il framework è una delle API del servizio di testo offerte dalla piattaforma Android.

Per usare il framework nella tua app, crei un tipo speciale di servizio Android che genera un oggetto sessione del **correttore ortografico.** In base al testo fornito, l'oggetto sessione restituisce suggerimenti ortografici generati dal correttore ortografico.

## Ciclo di vita del correttore ortografico

Il diagramma seguente mostra il ciclo di vita del servizio correttore ortografico:



**Figura 1.** Ciclo di vita del servizio controllo ortografico.

Per avviare il controllo ortografico, l'app avvia l'implementazione del servizio di controllo ortografico. I client dell'app, ad esempio attività o singoli elementi dell'interfaccia utente, richiedono una sessione di controllo ortografico al servizio, quindi usano la sessione per ottenere suggerimenti per il testo. Quando un client termina il funzionamento, chiude la sessione del correttore ortografico. Se necessario, l'app può arrestare il servizio di controllo ortografico in qualsiasi momento.

## Implementare un servizio di controllo ortografico

Per usare il framework del correttore ortografico nell'app, aggiungi un componente del servizio di controllo ortografico che include la definizione dell'oggetto sessione. Puoi anche aggiungere alla tua app un'attività facoltativa che controlla le impostazioni. È inoltre necessario aggiungere un file di metadati XML che descriva il servizio di controllo ortografico e aggiungere gli elementi appropriati al file manifesto.

### Classi di controllo ortografico

Definire l'oggetto servizio e sessione con le classi seguenti:

**Una sottoclasse di** SpellCheckerService

Implementa sia la classe che l'interfaccia del framework del correttore ortografico. All'interno della sottoclasse è necessario implementare il metodo seguente: SpellCheckerServiceService

createSession()

Metodo factory che restituisce un oggetto a un client che desidera eseguire il controllo ortografico. SpellCheckerService.Session

Vedi l'app di esempio Spell Checker Service per saperne di più sull'implementazione di questa classe.

**L'attuazione di** SpellCheckerService.Session

Oggetto fornito dal servizio di controllo ortografico ai client, per consente loro di passare testo al correttore ortografico e ricevere suggerimenti. All'interno di questa classe, è necessario implementare i metodi seguenti:

onCreate()

Chiamato dal sistema in risposta a . In questo metodo è possibile inizializzare l'oggetto in base alle impostazioni locali correnti e così via. createSession()SpellCheckerService.Session

onGetSentenceSuggestionsMultiple()

Fa il controllo ortografico effettivo. Questo metodo restituisce una matrice di suggerimenti contenenti per le frasi passate. SentenceSuggestionsInfo

Facoltativamente, è possibile implementare , che gestisce le richieste di annullamento del controllo ortografico, , che gestisce una richiesta di suggerimento di parole, o , che gestisce batch di richieste di suggerimento di parole. onCancel()onGetSuggestions()onGetSuggestionsMultiple()

Vedi l'app di esempio Spell Checker Client per saperne di più sull'implementazione di questa classe.

**Nota:** È necessario implementare tutti gli aspetti del controllo ortografico come asincroni e thread-safe. Un correttore ortografico può essere chiamato contemporaneamente da thread diversi in esecuzione su core diversi. E prendersi cura di questo automaticamente. **SpellCheckerServiceSpellCheckerService.Session**

### Manifesto e metadati del correttore ortografico

Oltre al codice, è necessario fornire il file manifesto appropriato e un file di metadati per il correttore ortografico.

Il file manifesto definisce l'applicazione, il servizio e l'attività per il controllo delle impostazioni, come illustrato nel frammento seguente:

<manifest xmlns:android="http://schemas.android.com/apk/res/android"  
 package="com.example.android.samplespellcheckerservice" >  
 <application  
 android:label="@string/app\_name" >  
 <service  
 android:label="@string/app\_name"  
 android:name=".SampleSpellCheckerService"  
 android:permission="android.permission.BIND\_TEXT\_SERVICE" >  
 <intent-filter >  
 <action android:name="android.service.textservice.SpellCheckerService" />  
 </intent-filter>  
  
 <meta-data  
 android:name="android.view.textservice.scs"  
 android:resource="@xml/spellchecker" />  
 </service>  
  
 <activity  
 android:label="@string/sample\_settings"  
 android:name="SpellCheckerSettingsActivity" >  
 <intent-filter >  
 <action android:name="android.intent.action.MAIN" />  
 </intent-filter>  
 </activity>  
 </application>  
</manifest>

Si noti che i componenti che desiderano utilizzare il servizio devono richiedere l'autorizzazione per assicurarsi che solo il sistema si associa al servizio. La definizione del servizio specifica anche il file di metadati, descritto nella sezione successiva. BIND\_TEXT\_SERVICEspellchecker.xml

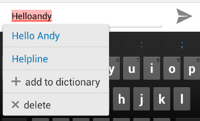
Il file di metadati contiene il codice XML seguente: spellchecker.xml

<spell-checker xmlns:android="http://schemas.android.com/apk/res/android"  
 android:label="@string/spellchecker\_name"  
 android:settingsActivity="com.example.SpellCheckerSettingsActivity">  
 <subtype  
 android:label="@string/subtype\_generic"  
 android:subtypeLocale="en”  
 />  
 <subtype  
 android:label="@string/subtype\_generic"  
 android:subtypeLocale="fr”  
 />  
</spell-checker>

I metadati specificano l'attività utilizzata dal correttore ortografico per il controllo delle impostazioni. Definisce inoltre i sottotipi per il correttore ortografico; in questo caso, i sottotipi definiscono le impostazioni locali che il correttore ortografico può gestire.

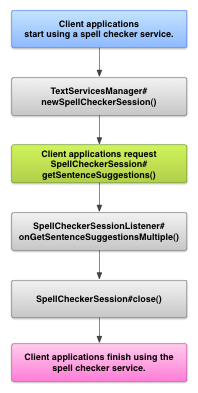
## Accedere al servizio di controllo ortografico da un client

Le applicazioni che usano le visualizzazioni beneficiano automaticamente del controllo ortografico, perché usa automaticamente un correttore ortografico. Le schermate seguenti mostrano questo: TextViewTextView

**Figura 2.** Controllo ortografico in TextView.

Tuttavia, potresti voler interagire direttamente con un servizio di controllo ortografico anche in altri casi. Il diagramma seguente mostra il flusso di controllo per l'interazione con un servizio di controllo ortografico:



**Figura 3.** Interagire con un servizio di controllo ortografico.

L'editor di metodi di input LatinIME in Android Open Source Project contiene un esempio di controllo ortografico.