public class

Summary: Nested Classes | Constants | Inherited Constants | Fields | Methods | Protected Methods | Inherited Methods | [Expand All]

Added in API level 1

KeyCharacterMap

extends Object
implements Parcelable

java.lang.Object
↳ android.view.KeyCharacterMap

Class Overview

Describes the keys provided by a keyboard device and their associated labels.

Summary

Nested Classes	
class KeyCharacterMap.KeyData	*This class was deprecated in API level 11. instead use `getDisplayLabel(int)`, `getNumber(int)` and `get(int, int)`.*
class KeyCharacterMap.UnavailableException	Thrown by `load(int)` when a key character map could not be loaded.

Constants	
int ALPHA	A keyboard with all the letters, and maybe some numbers.
int BUILT_IN_KEYBOARD	*This constant was deprecated in API level 11. This constant should no longer be used because there is no guarantee that a device has a built-in keyboard that can be used for typing text. There might not be a*

	built-in keyboard, the built-in keyboard might be a `NUMERIC` or `SPECIAL_FUNCTION` keyboard, or there might be multiple keyboards installed including external keyboards. When interpreting key presses received from the framework, applications should use the device id specified in the `KeyEvent` received. When synthesizing key presses for delivery elsewhere or when translating key presses from unknown keyboards, applications should use the special `VIRTUAL_KEYBOARD` device id.
int COMBINING_ACCENT	
int COMBINING_ACCENT_MASK	Mask the return value from `get(int, int)` with this value to get a printable representation of the accent character of a "dead key."
int FULL	A full PC-style keyboard.
char HEX_INPUT	This private-use character is used to trigger Unicode character input by hex digits.
int MODIFIER_BEHAVIOR_CHORDED	Modifier keys may be chorded with character keys.

int MODIFIER_BEHAVIOR_CHORDED_OR_TOGGLED	Modifier keys may be chorded with character keys or they may toggle into latched or locked states when pressed independently.
int NUMERIC	A numeric (12-key) keyboard.
char PICKER_DIALOG_INPUT	This private-use character is used to bring up a character picker for miscellaneous symbols.
int PREDICTIVE	A keyboard with all the letters, but with more than one letter per key.
int SPECIAL_FUNCTION	A keyboard that is only used to control special functions rather than for typing.
int VIRTUAL_KEYBOARD	The id of a generic virtual keyboard with a full layout that can be used to synthesize key events.

Inherited Constants [Expand]

▸ From interface android.os.Parcelable

Fields

public static final Creator<KeyCharacterMap> CREATOR

Public Methods

	describeContents ()
int	Describe the kinds of special objects contained in this Parcelable's marshalled representation.
	deviceHasKey (int keyCode)
static boolean	Queries the framework about whether any physical keys exist on the any keyboard attached to the device that are capable of producing the given key code.

	deviceHasKeys (int[] keyCodes)
static boolean[]	Queries the framework about whether any physical keys exist on the any keyboard attached to the device that are capable of producing the given array of key codes.
	get (int keyCode, int metaState)
int	Gets the Unicode character generated by the specified key and meta key state combination.
	getDeadChar (int accent, int c)
static int	Get the character that is produced by combining the dead key producing accent with the key producing character c.
	getDisplayLabel (int keyCode)
char	Gets the primary character for this key.
	getEvents (char[] chars)
KeyEvent[]	Get an array of KeyEvent objects that if put into the input stream could plausibly generate the provided sequence of characters.
	getKeyData (int keyCode, KeyCharacterMap.KeyData results)
boolean	*This method was deprecated in API level 11. instead use `getDisplayLabel(int)`, `getNumber(int)` or `get(int, int)`.*
	getKeyboardType ()
int	Gets the keyboard type.
	getMatch (int keyCode, char[] chars)
char	Gets the first character in the character array that can be generated by the specified key code.
	getMatch (int keyCode, char[] chars, int metaState)
char	Gets the first character in the character array that can be generated by the specified key code.
	getModifierBehavior ()
int	Gets a constant that describes the behavior of this keyboard's modifier keys such as KEYCODE_SHIFT_LEFT.
	getNumber (int keyCode)
char	Gets the number or symbol associated with the key.
	isPrintingKey (int keyCode)
boolean	Returns true if the specified key produces a glyph.
	load (int deviceId)
static KeyCharacterMap	Loads the key character maps for the keyboard with the specified device id.
	writeToParcel (Parcel out, int flags)
void	Flatten this object in to a Parcel.

Protected Methods

void	finalize() Invoked when the garbage collector has detected that this instance is no longer reachable.

Inherited Methods [Expand]

- From class java.lang.Object
- From interface android.os.Parcelable

Constants

public static final int **ALPHA**

Added in API level 1

A keyboard with all the letters, and maybe some numbers.

An alphabetic keyboard supports text entry directly but may have a condensed layout with a small form factor. In contrast to a `full keyboard (/reference/android/view/KeyCharacterMap.html#FULL)`, some symbols may only be accessible using special on-screen character pickers. In addition, to improve typing speed and accuracy, the framework provides special affordances for alphabetic keyboards such as auto-capitalization and toggled / locked shift and alt keys.

This type of keyboard is generally designed for thumb typing.

Constant Value: 3 (0x00000003)

public static final int **BUILT_IN_KEYBOARD**

Added in API level 1

This constant was deprecated in API level 11.
This constant should no longer be used because there is no guarantee that a device has a built-in keyboard that can be used for typing text. There might not be a built-in keyboard, the built-in keyboard might be a `NUMERIC (/reference/android/view/KeyCharacterMap.html#NUMERIC)` or `SPECIAL_FUNCTION (/reference/android/view/KeyCharacterMap.html#SPECIAL_FUNCTION)` keyboard, or there might be multiple keyboards installed including external keyboards. When interpreting key presses received from the framework, applications should use the device id specified in the `KeyEvent (/reference/android/view/KeyEvent.html)` received. When synthesizing key presses for delivery elsewhere or when translating key presses from unknown keyboards, applications should use the special `VIRTUAL_KEYBOARD (/reference/android/view/KeyCharacterMap.html#VIRTUAL_KEYBOARD)` device id.

The id of the device's primary built in keyboard is always 0.

Constant Value: 0 (0x00000000)

public static final int **COMBINING_ACCENT** Added in API level 1

Constant Value: -2147483648 (0x80000000)

public static final int **COMBINING_ACCENT_MASK** Added in API level 1

Mask the return value from `get(int, int)` (/reference/android/view/KeyCharacterMap.html#get(int, int)) with this value to get a printable representation of the accent character of a "dead key."

Constant Value: 2147483647 (0x7fffffff)

public static final int **FULL** Added in API level 11

A full PC-style keyboard.

A full keyboard behaves like a PC keyboard. All symbols are accessed directly by pressing keys on the keyboard without on-screen support or affordances such as auto-capitalization.

This type of keyboard is generally designed for full two hand typing.

Constant Value: 4 (0x00000004)

public static final char **HEX_INPUT** Added in API level 1

This private-use character is used to trigger Unicode character input by hex digits.

Constant Value: 61184 (0x0000ef00)

public static final int **MODIFIER_BEHAVIOR_CHORDED** Added in API level 11

Modifier keys may be chorded with character keys.

See Also

`ERROR(#getModifierBehavior()} for more details. /{#link #getModifierBehavior()} for more details.)`

Constant Value: 0 (0x00000000)

public static final int

MODIFIER_BEHAVIOR_CHORDED_OR_TOGGLED Added in API level 11

Modifier keys may be chorded with character keys or they may toggle into latched or locked states when pressed independently.

See Also

ERROR(#getModifierBehavior()} for more details. /{#link #getModifierBehavior()} for more details.)

Constant Value: 1 (0x00000001)

public static final int NUMERIC Added in API level 1

A numeric (12-key) keyboard.

A numeric keyboard supports text entry using a multi-tap approach. It may be necessary to tap a key multiple times to generate the desired letter or symbol.

This type of keyboard is generally designed for thumb typing.

Constant Value: 1 (0x00000001)

public static final char PICKER_DIALOG_INPUT Added in API level 1

This private-use character is used to bring up a character picker for miscellaneous symbols.

Constant Value: 61185 (0x0000ef01)

public static final int PREDICTIVE Added in API level 1

A keyboard with all the letters, but with more than one letter per key.

This type of keyboard is generally designed for thumb typing.

Constant Value: 2 (0x00000002)

public static final int SPECIAL_FUNCTION Added in API level 11

A keyboard that is only used to control special functions rather than for typing.

A special function keyboard consists only of non-printing keys such as HOME and POWER that are not actually used for typing.

Constant Value: 5 (0x00000005)

public static final int **VIRTUAL_KEYBOARD** Added in API level 11

The id of a generic virtual keyboard with a full layout that can be used to synthesize key events. Typically used with `getEvents(char[])` (/reference/android/view/KeyCharacterMap.html#getEvents(char[])).

Constant Value: -1 (0xffffffff)

Fields

public static final Creator<KeyCharacterMap> **CREATOR** Added in API level 16

Public Methods

public int **describeContents** () Added in API level 16

Describe the kinds of special objects contained in this Parcelable's marshalled representation.

Returns

a bitmask indicating the set of special object types marshalled by the Parcelable.

public static boolean **deviceHasKey** (int keyCode) Added in API level 3

Queries the framework about whether any physical keys exist on the any keyboard attached to the device that are capable of producing the given key code.

Parameters

keyCode The key code to query.

Returns

True if at least one attached keyboard supports the specified key code.

public static boolean[] **deviceHasKeys** (int[] keyCodes) Added in API level 3

Queries the framework about whether any physical keys exist on the any keyboard attached to the device that are capable of producing the given array of key codes.

Parameters

keyCodes The array of key codes to query.

Returns

A new array of the same size as the key codes array whose elements are set to true if at least one attached keyboard supports the corresponding key code at the same index in the key codes array.

public int **get** (int keyCode, int metaState)

Added in API level 1

Gets the Unicode character generated by the specified key and meta key state combination.

Returns the Unicode character that the specified key would produce when the specified meta bits (see `MetaKeyKeyListener` (/reference/android/text/method/MetaKeyKeyListener.html)) were active.

Returns 0 if the key is not one that is used to type Unicode characters.

If the return value has bit `COMBINING_ACCENT` (/reference/android/view/KeyCharacterMap.html#COMBINING_ACCENT) set, the key is a "dead key" that should be combined with another to actually produce a character -- see `getDeadChar(int, int)` (/reference/android/view/KeyCharacterMap.html#getDeadChar(int, int)) -- after masking with `COMBINING_ACCENT_MASK` (/reference/android/view/KeyCharacterMap.html#COMBINING_ACCENT_MASK).

Parameters

keyCode The key code.

metaState The meta key modifier state.

Returns

The associated character or combining accent, or 0 if none.

public static int **getDeadChar** (int accent, int c)

Added in API level 1

Get the character that is produced by combining the dead key producing accent with the key producing character c. For example, getDeadChar('`', 'e') returns è. getDeadChar('^', ' ') returns '^' and getDeadChar('^', '^') returns '^'.

Parameters

accent The accent character. eg. '`'

c The basic character.

Returns

The combined character, or 0 if the characters cannot be combined.

public char **getDisplayLabel** (int keyCode) Added in API level 1

Gets the primary character for this key. In other words, the label that is physically printed on it.

Parameters

keyCode The key code.

Returns

The display label character, or 0 if none (eg. for non-printing keys).

public KeyEvent[] **getEvents** (char[] chars) Added in API level 1

Get an array of KeyEvent objects that if put into the input stream could plausibly generate the provided sequence of characters. It is not guaranteed that the sequence is the only way to generate these events or that it is optimal.

This function is primarily offered for instrumentation and testing purposes. It may fail to map characters to key codes. In particular, the key character map for the `built-in keyboard` device id may be empty. Consider using the key character map associated with the `virtual keyboard` device id instead.

For robust text entry, do not use this function. Instead construct a `KeyEvent` with action code `ACTION_MULTIPLE` that contains the desired string using `KeyEvent(long, String, int, int)`.

Parameters

chars The sequence of characters to generate.

Returns

An array of `KeyEvent` objects, or null if the given char array can not be generated using the current key character map.

public boolean **getKeyData** (int keyCode,

KeyCharacterMap.KeyData results) Added in API level 1

> **This method was deprecated in API level 11.**
> instead use `getDisplayLabel(int)` (/reference/android/view/KeyCharacterMap.html#getDisplayLabel(int)), `getNumber(int)` (/reference/android/view/KeyCharacterMap.html#getNumber(int)) or `get(int, int)` (/reference/android/view/KeyCharacterMap.html#get(int, int)).

Get the character conversion data for a given key code.

Parameters

keyCode	The keyCode to query.
results	A `KeyCharacterMap.KeyData` instance that will be filled with the results.

Returns

True if the key was mapped. If the key was not mapped, results is not modified.

public int **getKeyboardType** ()

Added in API level 1

Gets the keyboard type. Returns `NUMERIC` (/reference/android/view/KeyCharacterMap.html#NUMERIC), `PREDICTIVE` (/reference/android/view/KeyCharacterMap.html#PREDICTIVE), `ALPHA` (/reference/android/view/KeyCharacterMap.html#ALPHA), `FULL` (/reference/android/view/KeyCharacterMap.html#FULL) or `SPECIAL_FUNCTION` (/reference/android/view/KeyCharacterMap.html#SPECIAL_FUNCTION).

Different keyboard types have different semantics. Refer to the documentation associated with the keyboard type constants for details.

Returns

The keyboard type.

public char **getMatch** (int keyCode, char[] chars)

Added in API level 1

Gets the first character in the character array that can be generated by the specified key code.

This is a convenience function that returns the same value as `getMatch(keyCode, chars, 0)` (/reference/android/view/KeyCharacterMap.html#getMatch(int, char[], int)).

Parameters

keyCode	The keycode.
chars	The array of matching characters to consider.

Returns

The matching associated character, or 0 if none.

public char **getMatch** (int keyCode, char[] chars, int metaState)

Added in API level 1

Gets the first character in the character array that can be generated by the specified key code. If there are multiple choices, prefers the one that would be generated with the specified meta key modifier state.

Parameters

keyCode	The key code.
chars	The array of matching characters to consider.
metaState	The preferred meta key modifier state.

Returns

The matching associated character, or 0 if none.

public int **getModifierBehavior** ()

Added in API level 11

Gets a constant that describes the behavior of this keyboard's modifier keys such as KEYCODE_SHIFT_LEFT (/reference/android/view/KeyEvent.html#KEYCODE_SHIFT_LEFT).

Currently there are two behaviors that may be combined:

- Chorded behavior: When the modifier key is pressed together with one or more character keys, the keyboard inserts the modified keys and then resets the modifier state when the modifier key is released.
- Toggled behavior: When the modifier key is pressed and released on its own it first toggles into a latched state. When latched, the modifier will apply to next character key that is pressed and will then reset itself to the initial state. If the modifier is already latched and the modifier key is pressed and release on its own again, then it toggles into a locked state. When locked, the modifier will apply to all subsequent character keys that are pressed until unlocked by pressing the modifier key on its own one more time to reset it to the initial state. Toggled behavior is useful for small profile keyboards designed for thumb typing.

This function currently returns MODIFIER_BEHAVIOR_CHORDED

(/reference/android/view/KeyCharacterMap.html#MODIFIER_BEHAVIOR_CHORDED) when the keyboard type (/reference/android/view/KeyCharacterMap.html#getKeyboardType()) is FULL (/reference/android/view/KeyCharacterMap.html#FULL) or SPECIAL_FUNCTION (/reference/android/view/KeyCharacterMap.html#SPECIAL_FUNCTION) and MODIFIER_BEHAVIOR_CHORDED_OR_TOGGLED (/reference/android/view/KeyCharacterMap.html#MODIFIER_BEHAVIOR_CHORDED_OR_TOGGLED) otherwise. In the future, the function may also take into account global keyboard accessibility settings, other user preferences, or new device capabilities.

Returns

The modifier behavior for this keyboard.

See Also

ERROR(#MODIFIER_BEHAVIOR_CHORDED}/{@link #MODIFIER_BEHAVIOR_CHORDED})
ERROR(#MODIFIER_BEHAVIOR_CHORDED_OR_TOGGLED}/{@link #MODIFIER_BEHAVIOR_CHORDED_OR_TOGGLED})

public char **getNumber** (int keyCode)

Added in API level 1

Gets the number or symbol associated with the key.

The character value is returned, not the numeric value. If the key is not a number, but is a symbol, the symbol is retuned.

This method is intended to to support dial pads and other numeric or symbolic entry on keyboards where certain keys serve dual function as alphabetic and symbolic keys. This method returns the number or symbol associated with the key independent of whether the user has pressed the required modifier.

For example, on one particular keyboard the keys on the top QWERTY row generate numbers when ALT is pressed such that ALT-Q maps to '1'. So for that keyboard when getNumber(int) (/reference/android/view/KeyCharacterMap.html#getNumber(int)) is called with KEYCODE_Q (/reference/android/view/KeyEvent.html#KEYCODE_Q) it returns '1' so that the user can type numbers without pressing ALT when it makes sense.

Parameters

keyCode The key code.

Returns

The associated numeric or symbolic character, or 0 if none.

public boolean **isPrintingKey** (int keyCode) Added in API level 1

Returns true if the specified key produces a glyph.

Parameters

keyCode The key code.

Returns

True if the key is a printing key.

public static KeyCharacterMap **load** (int deviceId) Added in API level 1

Loads the key character maps for the keyboard with the specified device id.

Parameters

deviceId The device id of the keyboard.

Returns

The associated key character map.

Throws

UnavailableException} if the key character map could not be loaded because it was malformed or the default key character map is missing from the system.

public void **writeToParcel** (Parcel out, int flags) Added in API level 16

Flatten this object in to a Parcel.

Parameters

out The Parcel in which the object should be written.

flags Additional flags about how the object should be written. May be 0 or `PARCELABLE_WRITE_RETURN_VALUE`.

Protected Methods

protected void **finalize** () Added in API level 1

Invoked when the garbage collector has detected that this instance is no longer reachable. The default implementation does nothing, but this method can be overridden to free resources.

Note that objects that override `finalize` are significantly more

expensive than objects that don't. Finalizers may be run a long time after the object is no longer reachable, depending on memory pressure, so it's a bad idea to rely on them for cleanup. Note also that finalizers are run on a single VM-wide finalizer thread, so doing blocking work in a finalizer is a bad idea. A finalizer is usually only necessary for a class that has a native peer and needs to call a native method to destroy that peer. Even then, it's better to provide an explicit `close` method (and implement `Closeable` (/reference/java/io/Closeable.html)), and insist that callers manually dispose of instances. This works well for something like files, but less well for something like a `BigInteger` where typical calling code would have to deal with lots of temporaries. Unfortunately, code that creates lots of temporaries is the worst kind of code from the point of view of the single finalizer thread.

If you *must* use finalizers, consider at least providing your own `ReferenceQueue` (/reference/java/lang/ref/ReferenceQueue.html) and having your own thread process that queue.

Unlike constructors, finalizers are not automatically chained. You are responsible for calling `super.finalize()` yourself.

Uncaught exceptions thrown by finalizers are ignored and do not terminate the finalizer thread. See *Effective Java* Item 7, "Avoid finalizers" for more.

Throws

Throwable