## NAME

XkbDeviceBellEvent - Creates a bell event for an X input extension device or for the keyboard, without ringing the corresponding bell

## SYNOPSIS

**Bool XkbDeviceBellEvent (Display ****display***, Window** *window***, unsigned int** *device_spec***, unsigned int** *bell_class***, unsigned int** *bell_id***, int** *percent***, Atom** *name***);**

## ARGUMENTS

*display*
connection to the X server

*window*
event window, or None

*device_spec*
device ID, or XkbUseCoreKbd

*bell_class*
input extension bell class for the event

*bell_id*
input extension bell ID for the event

*percent*
volume for the bell, which can range from -100 to 100 inclusive

*name*
a bell name, or NULL

## DESCRIPTION

The core X protocol allows only applications to explicitly sound the system bell with a given duration, pitch, and volume. Xkb extends this capability by allowing clients to attach symbolic names to bells, disable audible bells, and receive an event whenever the keyboard bell is rung. For the purposes of this document, the *audible* bell is defined to be the system bell, or the default keyboard bell, as opposed to any other audible sound generated elsewhere in the system. You can ask to receive XkbBellNotify events when any client rings any one of the following:

- The default bell

- Any bell on an input device that can be specified by a bell_class and bell_id pair

- Any bell specified only by an arbitrary name. (This is, from the server's point of view, merely a name, and not connected with any physical sound-generating device. Some client application must generate the sound, or visual feedback, if any, that is associated with the name.)

You can also ask to receive XkbBellNotify events when the server rings the default bell or if any client has requested events only (without the bell sounding) for any of the bell types previously listed.

You can disable audible bells on a global basis. For example, a client that replaces the keyboard bell with some other audible cue might want to turn off the AudibleBell control to prevent the server from also generating a sound and avoid cacophony. If you disable audible bells and request to receive XkbBellNotify events, you can generate feedback different from the default bell.

You can, however, override the AudibleBell control by calling one of the functions that force the ringing of a bell in spite of the setting of the AudibleBell control - *XkbForceDeviceBell* or *XkbForceBell.* In this case the server does not generate a bell event.

Just as some keyboards can produce keyclicks to indicate when a key is pressed or repeating, Xkb can provide feedback for the controls by using special beep codes. The AccessXFeedback control is used to configure the specific types of operations that generate feedback.

Bell Names

You can associate a name to an act of ringing a bell by converting the name to an Atom and then using this name when you call the functions listed in this chapter. If an event is generated as a result, the name is then passed to all other clients interested in receiving XkbBellNotify events. Note that these are arbitrary names and that there is no binding to any sounds. Any sounds or other effects (such as visual bells on the screen) must be generated by a client application upon receipt of the bell event containing the name. There is no default name for the default keyboard bell. The server does generate some predefined bells for the AccessX controls. These named bells are shown in the Table 1 below; the name is included in any bell event sent to clients that have requested to receive XkbBellNotify events.

Table 1 Predefined

Bells

---

| Action | Named Bell |
|---|---|
| Indicator turned on | AX_IndicatorOn |
| Indicator turned off | AX_IndicatorOff |
| More than one indicator changed state | AX_IndicatorChange |
| Control turned on | AX_FeatureOn |
| Control turned off | AX_FeatureOff |
| More than one control changed state | AX_FeatureChange |
| SlowKeys and BounceKeys about to be turned on or off | AX_SlowKeysWarning |
| SlowKeys key pressed | AX_SlowKeyPress |
| SlowKeys key accepted | AX_SlowKeyAccept |
| SlowKeys key rejected | AX_SlowKeyReject |
| Accepted SlowKeys key released | AX_SlowKeyRelease |
| BounceKeys key rejected | AX_BounceKeyReject |
| StickyKeys key latched | AX_StickyLatch |
| StickyKeys key locked | AX_StickyLock |
| StickyKeys key unlocked | AX_StickyUnlock |

Audible Bells

Using Xkb you can generate bell events that do not necessarily ring the system bell. This is useful if you need to use an audio server instead of the system beep. For example, when an audio client starts, it could disable the audible bell (the system bell) and then listen for XkbBellNotify events. When it receives a XkbBellNotify event, the audio client could then send a request to an

audio server to play a sound.

You can control the audible bells feature by passing the XkbAudibleBellMask to *XkbChangeEnabledControls*. If you set XkbAudibleBellMask on, the server rings the system bell when a bell event occurs. This is the default. If you set XkbAudibleBellMask off and a bell event occurs, the server does not ring the system bell unless you call *XkbForceDeviceBell* or *XkbForceBell.*

Audible bells are also part of the per-client auto-reset controls.

Bell Functions

Use the functions described in this section to ring bells and to generate bell events.

The input extension has two types of feedbacks that can generate bells - bell feedback and keyboard feedback. Some of the functions in this section have *bell_class* and *bell_id* parameters; set them as follows: Set *bell_class* to BellFeedbackClass or KbdFeedbackClass. A device can have more than one feedback of each type; set *bell_id* to the particular bell feedback of *bell_class* type.

Table 2 shows the conditions that cause a bell to sound or an XkbBellNotifyEvent to be generated when a bell function is called.

Table 2 Bell Sounding and Bell Event Generating

| Function called | AudibleBell | Server sounds a bell | Server sends an XkbBellNotifyEvent |
|---|---|---|---|
| XkbDeviceBell | On | Yes | Yes |
| XkbDeviceBell | Off | No | Yes |
| XkbBell | On | Yes | Yes |
| XkbBell | Off | No | Yes |
| XkbDeviceBellEvent | On or Off | No | Yes |
| XkbBellEvent | On or Off | No | Yes |
| XkbDeviceForceBell | On or | Yes | No |

| | | | |
|---|---|---|---|
| XkbForceBell | Off<br>On or<br>Off | Yes | No |

If a compatible keyboard extension isn't present in the X server, *XkbDeviceBellEvent* immediately returns False. Otherwise, *XkbDeviceBellEvent* causes an XkbBellNotify event to be sent to all interested clients and returns True. Set *percent* to be the volume relative to the base volume for the keyboard as described for *XBell.*

In addition, *XkbDeviceBellEvent* may generate Atom protocol errors as well as XkbBellNotify events. You can call *XkbBell* without first initializing the keyboard extension.

**RETURN VALUES**

| | |
|---|---|
| True | The *XkbDeviceBellEvent* sends an XkbBellNotify event to to all interested clients and returns True. |
| False | If a compatible keyboard extension isn't present in the X server, *XkbDeviceBellEvent* immediately returns False |

**STRUCTURES**

Xkb generates XkbBellNotify events for all bells except for those resulting from calls to *XkbForceDeviceBell* and *XkbForceBell.* To receive XkbBellNotify events under all possible conditions, pass XkbBellNotifyMask in both the *bits_to_change* and *values_for_bits* parameters to *XkbSelectEvents.*

The XkbBellNotify event has no event details. It is either selected or it is not. However, you can call *XkbSelectEventDetails* using XkbBellNotify as the *event_type* and specifying XkbAllBellEventsMask in *bits_to_change* and *values_for_bits.* This has the same effect as a call to *XkbSelectEvents.*

The structure for the XkbBellNotify event type contains:

```
typedef struct _XkbBellNotify {
    int           type;       /* Xkb extension base event code */
    unsigned long  serial;     /* X server serial number for event */
    Bool          send_event;  /* True => synthetically generated */
    Display *     display;     /* server connection where event generated */
    Time          time;       /* server time when event generated */
    int           xkb_type;    /* XkbBellNotify */
    unsigned int   device;     /* Xkb device ID, will not be XkbUseCoreKbd */
```

```
    int           percent;     /* requested volume as % of max */
    int           pitch;       /* requested pitch in Hz */
    int           duration;    /* requested duration in microseconds */
    unsigned int  bell_class;  /* X input extension feedback class */
    unsigned int  bell_id;     /* X input extension feedback ID */
    Atom          name;        /* "name" of requested bell */
    Window        window;      /* window associated with event */
    Bool          event_only;  /* False -> the server did not produce a beep */
} XkbBellNotifyEvent;
```

If your application needs to generate visual bell feedback on the screen when it receives a bell event, use the window ID in the XkbBellNotifyEvent, if present.

## SEE ALSO

**XBell**(3), **XkbBellNotify**(3), **XkbChangeEnabledControls**(3), **XkbDeviceBell**(3), **XkbForceBell**(3), **XkbForceDeviceBell**(3), **XkbSelectEvents**(3), **XkbSelectEventDetails**(3), **XkbUseCoreKbd**(3)