NAME

XkbGetKeyboardByName - Build a new keyboard description from a set of named components, and to optionally have the server use the resulting description to replace an active one

SYNOPSIS

XkbDescPtr XkbGetKeyboardByName (Display ****dpy*, unsigned int** *device_spec***,**
XkbComponentNamesPtr *names***, unsigned int** *want***, unsigned int** *need***, Bool** *load***);**

ARGUMENTS

dpy connection to X server

device_spec
device ID, or XkbUseCoreKbd

names
names of components to fetch

want
desired structures in returned record

need
mandatory structures in returned record

load
True => load into *device_spec*

DESCRIPTION

A client may request that the server fetch one or more components from its database and use those components to build a new server keyboard description. The new keyboard description may be built from scratch, or it may be built starting with the current keyboard description for a particular device. Once the keyboard description is built, all or part of it may be returned to the client. The parts returned to the client need not include all of the parts used to build the description. At the time it requests the server to build a new keyboard description, a client may also request that the server use the new description internally to replace the current keyboard description for a specific device, in which case the behavior of the device changes accordingly.

To build a new keyboard description from a set of named components, and to optionally have the server use the resulting description to replace an active one, use *XkbGetKeyboardByName.*

names contains a set of expressions describing the keyboard components the server should use to build

the new keyboard description. *want* and *need* are bit fields describing the parts of the resulting keyboard description that should be present in the returned XkbDescRec.

The individual fields in *names* are *component expressions* composed of keyboard component names (no wildcarding as may be used in *XkbListComponents),* the special component name symbol '%', and the special operator characters '+' and '|'. A component expression is parsed left to right, as follows:

- The special component name "computed" may be used in keycodes component expressions and refers to a component consisting of a set of keycodes computed automatically by the server as needed.

- The special component name "canonical" may be used in types component expressions and refers to a partial component defining the four standard key types: ALPHABETIC, ONE_LEVEL, TWO_LEVEL, and KEYPAD.

- The special component name '%' refers to the keyboard description for the device specified in device_spec or the keymap names component. If a keymap names component is specified that does not begin with '+' or '|' and does not contain '%', then '%' refers to the description generated by the keymap names component. Otherwise, it refers to the keyboard description for device_spec.

- The '+' operator specifies that the following component should override the currently assembled description; any definitions that are present in both components are taken from the second.

- The '|' operator specifies that the next specified component should augment the currently assembled description; any definitions that are present in both components are taken from the first.

- If the component expression begins with an operator, a leading '%' is implied.

- If any unknown or illegal characters appear anywhere in the expression, the entire expression is invalid and is ignored.

  For example, if *names->symbols* contained the expression "+de", it specifies that the default member of the "de" class of symbols should be applied to the current keyboard mapping, overriding any existing definitions (it could also be written "+de(default)").

  Here is a slightly more involved example: the expression "acme(ascii)+de(basic)|iso9995-3" constructs a German (de) mapping for the ASCII keyboard supplied by the "acme" vendor. The

new definition begins with the symbols for the ASCII keyboard for Acme *(acme(ascii)),* overrides them with definitions for the basic German keyboard *(de(basic)),* and then applies the definitions from the default iso9995-3 keyboard *(iso9995-3)* to any undefined keys or groups of keys (part three of the iso9995 standard defines a common set of bindings for the secondary group, but allows national layouts to override those definitions where necessary).

NOTE The interpretation of the above expression components (acme, ascii, de, basic, iso9995-3) is not defined by Xkb; only the operations and their ordering are.

Note that the presence of a keymap *names* component that does not contain '%' (either explicit or implied by virtue of an expression starting with an operator) indicates a description that is independent of the keyboard description for the device specified in *device_spec.* The same is true of requests in which the keymap names component is empty and all five other names components contain expressions void of references to '%'. Requests of this form allow you to deal with keyboard definitions independent of any actual device.

The server parses all non-NULL fields in *names* and uses them to build a keyboard description. However, before parsing the expressions in *names,* the server ORs the bits in *want* and *need* together and examines the result in relationship to the expressions in *names.* Table 1 identifies the components that are required for each of the possible bits in *want* or *need.* If a required component has not been specified in the *names structure (the corresponding field is NULL), the server substitutes the* expression "%", resulting in the component values being taken from *device_spec.* In addition, if *load* is True, the server modifies *names* if necessary (again using a "%" entry) to ensure all of the following fields are non-NULL: *types, keycodes, symbols,* and *compat.*

Table 1 Want and Need Mask Bits and Required Names Components

want or need mask bit	Required names Components	value
XkbGBN_TypesMask	Types	(1L<<0)
XkbGBN_CompatMapMask	Compat	(1L<<1)
XkbGBN_ClientSymbolsMask	Types + Symbols + Keycodes	(1L<<2)
XkbGBN_ServerSymbolsMask	Types + Symbols + Keycodes	(1L<<3)
XkbGBN_SymbolsMask	Symbols	(1L<<1)
XkbGBN_IndicatorMapMask	Compat	(1L<<4)

XkbGBN_KeyNamesMask	Keycodes	(1L<<5)
XkbGBN_GeometryMask	Geometry	(1L<<6)
XkbGBN_OtherNamesMask	Types + Symbols + Keycodes + Compat + Geometry	(1L<<7)
XkbGBN_AllComponentsMask		(0xff)

need specifies a set of keyboard components that the server must be able to resolve in order for *XkbGetKeyboardByName* to succeed; if any of the components specified in *need* cannot be successfully resolved, *XkbGetKeyboardByName* fails.

want specifies a set of keyboard components that the server should attempt to resolve, but that are not mandatory. If the server is unable to resolve any of these components, *XkbGetKeyboardByName* still succeeds. Bits specified in *want* that are also specified in *need* have no effect in the context of *want.*

If *load* is True, the server updates its keyboard description for *device_spec* to match the result of the keyboard description just built. If load is False, the server's description for device *device_spec* is not updated. In all cases, the parts specified by *want* and *need* from the just-built keyboard description are returned.

The *names* structure in an XkbDescRec keyboard description record contains one field for each of the five component types used to build a keyboard description. When a keyboard description is built from a set of database components, the corresponding fields in this *names* structure are set to match the expressions used to build the component.

Building a New Keyboard Description from the Server Database

The information returned to the client in the XkbDescRec is essentially the result of a series of calls to extract information from a fictitious device whose description matches the one just built. The calls corresponding to each of the mask bits are summarized in Table 2, together with the XkbDescRec components that are filled in.

Table 2 XkbDescRec Components Returned for Values of Want & Needs

Request (want+need)	Fills in Xkb components	Equivalent Function Call
XkbGBN_TypesMask	map.types	XkbGetUpdatedMap(dpy, Xk Xkb)

XkbGBN_ServerSymbolsMask	server	XkbGetUpdatedMap(dpy,
XkbAllClientInfoMask,		
Xkb)		
XkbGBN_ClientSymbolsMask	map, including	
	map.types	
XkbGetUpdatedMap(dpy, XkbAllServerInfoMask, Xkb)		
XkbGBN_IndicatorMaps	indicators	XkbGetIndicatorMap(dpy,
XkbAllIndicators,		
Xkb)		
XkbGBN_CompatMapMask	compat	XkbGetCompatMap(dpy, Xk
XkbGBN_GeometryMask	geom	XkbGetGeometry(dpy,
		Xkb)
XkbGBN_KeyNamesMask	names.keys	XkbGetNames(dpy, XkbKey
		|
	names.key_aliases	XkbKeyAliasesMask,
		Xkb)
XkbGBN_OtherNamesMask	names.keycodes	XkbGetNames(dpy, XkbAllN
		&
	names.geometry	~(XkbKeyNamesMask |
		XkbKeyAliasesMask),
	names.symbols	Xkb)
	names.types	
	map.types[*].lvl_names[*]	
	names.compat	
	names.vmods	
	names.indicators	
	names.groups	
	names.radio_groups	
	names.phys_symbols	

There is no way to determine which components specified in *want* (but not in *need)* were actually fetched, other than breaking the call into successive calls to *XkbGetKeyboardByName* and specifying individual components.

XkbGetKeyboardByName always sets *min_key_code* and *max_key_code* in the returned XkbDescRec structure.

XkbGetKeyboardByName is synchronous; it sends the request to the server to build a new keyboard description and waits for the reply. If successful, the return value is non-NULL. *XkbGetKeyboardByName* generates a BadMatch protocol error if errors are encountered when

building the keyboard description.

STRUCTURES

The complete description of an Xkb keyboard is given by an XkbDescRec. The component structures in the XkbDescRec represent the major Xkb components outlined in Figure 1.1.

```
typedef struct {
  struct _XDisplay * display;      /* connection to X server */
  unsigned short     flags;        /* private to Xkb, do not modify */
  unsigned short     device_spec;  /* device of interest */
  KeyCode            min_key_code; /* minimum keycode for device */
  KeyCode            max_key_code; /* maximum keycode for device */
  XkbControlsPtr     ctrls;        /* controls */
  XkbServerMapPtr    server;       /* server keymap */
  XkbClientMapPtr    map;          /* client keymap */
  XkbIndicatorPtr    indicators;   /* indicator map */
  XkbNamesPtr        names;        /* names for all components */
  XkbCompatMapPtr    compat;       /* compatibility map */
  XkbGeometryPtr     geom;         /* physical geometry of keyboard */
} XkbDescRec, *XkbDescPtr;
```

The *display* field points to an X display structure. The *flags field is private to the library: modifying flags* may yield unpredictable results. The *device_spec* field specifies the device identifier of the keyboard input device, or XkbUseCoreKeyboard, which specifies the core keyboard device. The *min_key_code* and *max_key_code* fields specify the least and greatest keycode that can be returned by the keyboard.

Each structure component has a corresponding mask bit that is used in function calls to indicate that the structure should be manipulated in some manner, such as allocating it or freeing it. These masks and their relationships to the fields in the XkbDescRec are shown in Table 3.

Table 3 Mask Bits for XkbDescRec

Mask Bit	XkbDescRec Field	Value
XkbControlsMask	ctrls	(1L<<0)
XkbServerMapMask	server	(1L<<1)

XkbIClientMapMask	map	(1L<<2)
XkbIndicatorMapMask	indicators	(1L<<3)
XkbNamesMask	names	(1L<<4)
XkbCompatMapMask	compat	(1L<<5)
XkbGeometryMask	geom	(1L<<6)
XkbAllComponentsMask	All Fields	(0x7f)

DIAGNOSTICS

BadMatch A compatible version of Xkb was not available in the server or an argument has correct type and range, but is otherwise invalid

SEE ALSO

XkbListComponents(3)