NAME

XkbOutOfRangeGroupInfo - Returns only the out-of-range processing information from the group_info field of an XkbSymMapRec structure

SYNOPSIS

unsigned char XkbOutOfRangeGroupInfo (unsigned char *grp_inf***);**

ARGUMENTS

grp_inf

Xkb description of interest

DESCRIPTION

XkbOutOfRangeGroupInfo returns only the out-of-range processing information from the *group_info* field of an XkbSymMapRec structure.

The *group_info* field of an XkbSymMapRec is an encoded value containing the number of groups of symbols bound to the key as well as the specification of the treatment of out-of-range groups. It is legal for a key to have zero groups, in which case it also has zero symbols and all events from that key yield NoSymbol. To obtain the number of groups of symbols bound to the key, use *XkbKeyNumGroups.* To change the number of groups bound to a key, use *XkbChangeTypesOfKey*. To obtain a mask that determines the treatment of out-of-range groups, use *XkbKeyGroupInfo* and *XkbOutOfRangeGroupInfo.*

The keyboard controls contain a *groups_wrap* field specifying the handling of illegal groups on a global basis. That is, when the user performs an action causing the effective group to go out of the legal range, the *groups_wrap* field specifies how to normalize the effective keyboard group to a group that is legal for the keyboard as a whole, but there is no guarantee that the normalized group will be within the range of legal groups for any individual key. The per-key *group_info* field specifies how a key treats a legal effective group if the key does not have a type specified for the group of concern. For example, the Enter key usually has just one group defined. If the user performs an action causing the global keyboard group to change to Group2, the *group_info* field for the Enter key describes how to handle this situation.

Out-of-range groups for individual keys are mapped to a legal group using the same options as are used for the overall keyboard group. The particular type of mapping used is controlled by the bits set in the *group_info* flag, as shown in Table 1.

Table 1 group_info Range Normalization

Bits set in group_info	Normalization method
XkbRedirectIntoRange	XkbRedirectIntoRange
XkbClampIntoRange	XkbClampIntoRange
none of the above	XkbWrapIntoRange

STRUCTURES

The KeySymMapRec structure is defined as follows:

```
#define XkbNumKbdGroups          4
#define XkbMaxKbdGroup           (XkbNumKbdGroups-1)

typedef struct {                /* map to keysyms for a single keycode */
    unsigned char      kt_index[XkbNumKbdGroups];  /* key type index for each group */
    unsigned char      group_info; /* # of groups and out of range group handling */
    unsigned char      width;      /* max # of shift levels for key */
    unsigned short     offset;     /* index to keysym table in syms array */
} XkbSymMapRec, *XkbSymMapPtr;
```

The XkbControlsRec structure is defined as follows:

```
#define XkbMaxLegalKeyCode     255
#define XkbPerKeyBitArraySize  ((XkbMaxLegalKeyCode+1)/8)


typedef struct {
    unsigned char   mk_dflt_btn;      /* default button for keyboard driven mouse */
    unsigned char   num_groups;       /* number of keyboard groups */
    unsigned char   groups_wrap;      /* how to wrap out-of-bounds groups */
    XkbModsRec      internal;         /* defines server internal modifiers */
    XkbModsRec      ignore_lock;      /* modifiers to ignore when checking for grab */
    unsigned int    enabled_ctrls;    /* 1 bit => corresponding boolean control enabled */
    unsigned short  repeat_delay;     /* ms delay until first repeat */
    unsigned short  repeat_interval;  /* ms delay between repeats */
    unsigned short  slow_keys_delay;  /* ms minimum time key must be down to be ok */
    unsigned short  debounce_delay;   /* ms delay before key reactivated */
    unsigned short  mk_delay;         /* ms delay to second mouse motion event */
```

```
    unsigned short  mk_interval;      /* ms delay between repeat mouse events */
    unsigned short  mk_time_to_max;   /* # intervals until constant mouse move */
    unsigned short  mk_max_speed;     /* multiplier for maximum mouse speed */
    short           mk_curve;         /* determines mouse move curve type */
    unsigned short  ax_options;       /* 1 bit => Access X option enabled */
    unsigned short  ax_timeout;       /* seconds until Access X disabled */
    unsigned short  axt_opts_mask;    /* 1 bit => options to reset on Access X timeout */
    unsigned short  axt_opts_values;  /* 1 bit => turn option on, 0=> off */
    unsigned int    axt_ctrls_mask;   /* which bits in enabled_ctrls to modify */
    unsigned int    axt_ctrls_values; /* values for new bits in enabled_ctrls */
    unsigned char   per_key_repeat[XkbPerKeyBitArraySize];  /* per key auto repeat */
  } XkbControlsRec, *XkbControlsPtr;
```

The XkbControlsRec structure is defined as follows:

```
  #define XkbMaxLegalKeyCode     255
  #define XkbPerKeyBitArraySize  ((XkbMaxLegalKeyCode+1)/8)


  typedef struct {
    unsigned char   mk_dflt_btn;      /* default button for keyboard driven mouse */
    unsigned char   num_groups;       /* number of keyboard groups */
    unsigned char   groups_wrap;      /* how to wrap out-of-bounds groups */
    XkbModsRec      internal;         /* defines server internal modifiers */
    XkbModsRec      ignore_lock;      /* modifiers to ignore when checking for grab */
    unsigned int    enabled_ctrls;    /* 1 bit => corresponding boolean control enabled */
    unsigned short  repeat_delay;     /* ms delay until first repeat */
    unsigned short  repeat_interval;  /* ms delay between repeats */
    unsigned short  slow_keys_delay;  /* ms minimum time key must be down to be ok */
    unsigned short  debounce_delay;   /* ms delay before key reactivated */
    unsigned short  mk_delay;         /* ms delay to second mouse motion event */
    unsigned short  mk_interval;      /* ms delay between repeat mouse events */
    unsigned short  mk_time_to_max;   /* # intervals until constant mouse move */
    unsigned short  mk_max_speed;     /* multiplier for maximum mouse speed */
    short           mk_curve;         /* determines mouse move curve type */
    unsigned short  ax_options;       /* 1 bit => Access X option enabled */
    unsigned short  ax_timeout;       /* seconds until Access X disabled */
    unsigned short  axt_opts_mask;    /* 1 bit => options to reset on Access X timeout */
    unsigned short  axt_opts_values;  /* 1 bit => turn option on, 0=> off */
    unsigned int    axt_ctrls_mask;   /* which bits in enabled_ctrls to modify */
```

```
    unsigned int    axt_ctrls_values; /* values for new bits in enabled_ctrls */
    unsigned char   per_key_repeat[XkbPerKeyBitArraySize];  /* per key auto repeat */
} XkbControlsRec, *XkbControlsPtr;
```

SEE ALSO

XkbChangeTypesOfKey(3), **XkbKeyGroupInfo**(3), **XkbOutOfRangeGroupInfo.**(3)