

X-I

# Xlib : les évènements

- **Evènements et masques**
- **Groupes d'évènements**
- **Distribution et propagation**
- **Structures pour les évènements**
- **Capture**
- **Masques et évènements**

X-I

## Evènements

**Xlib définit 33 évènements.**

- **Chaque évènement signale**
  - | **une *entrée* , c'est-à-dire un changement dû au clavier ou à la souris;**
  - | **ou un *changement* dans l'état (au sens large) du display.**
- **Les évènements sont envoyés, par le serveur, aux *clients*.**
- **Les évènements se présentent sous la forme d'une structure contenant**
  - | **le type de l'évènement;**
  - | **un ensemble d'informations pertinentes pour l'évènement :**
    - » **la date,**
    - » **la fenêtre à laquelle se rapporte l'évènement,**
    - » **les coordonnées du pointeur de la souris...**
- **Deux types d'évènements sont réservés pour le serveur :**
  - | **l'un pour la transmission des *erreurs*,**
  - | **l'autre pour les *réponses* aux demandes d'informations (transfert de tables, etc).**

X-I

## Masques d'évènements

- o Un client sélectionne les évènements qu'il veut recevoir (par `XSelectInput` par exemple)
  - | *pour* une fenêtre donnée,
  - | *par* un masque d'évènements qui est l'union logique de masques élémentaires.
- o Il n'y a pas bijection entre évènements et masques d'évènements:
  - | certains masques (comme `StructureNotifyMask`) sélectionnent plusieurs évènements;
  - | d'autres masques (comme `Button1MotionMask`) subdivisent des évènements;
  - | certains évènements (comme `MappingNotify`) ne sont pas masquables: un client les reçoit d'office;
  - | il existe même des masques (`OwnerGrabButtonMask`, `PointerMotionHintMask`) qui ne correspondent à aucun évènement, mais servent d'indications complémentaires !

X-I

## Groupes d'évènements

- o **Evènement souris :**

```
ButtonPress  
ButtonRelease  
MotionNotify  
EnterNotify  
LeaveNotify
```

- o **Evènements clavier:**

```
KeyPress  
KeyRelease  
FocusIn  
FocusOut
```

- o **Exposition :**

```
Expose  
GraphicsExpose  
NoExpose
```

- o **Notification (information d'un changement) :**

```
ConfigureNotify  
CirculateNotify  
CreateNotify  
DestroyNotify  
GravityNotify  
MapNotify  
MappingNotify  
ReparentNotify  
UnmapNotify  
VisibilityNotify  
KeymapNotify  
ColormapNotify
```

- o **Contrôle de structure (gestionnaire de fenêtres):**

```
CirculateRequest  
ConfigureRequest  
MapRequest  
ResizeRequest
```

- o **Communication entre clients :**

```
ClientMessage  
PropertyNotify  
SelectionClear  
SelectionNotify  
SelectionRequest
```

X-I

## Evènements dus à l'action de l'utilisateur

- o **Action sur le clavier**
  - `KeyPress`, `KeyRelease`
- o **Action sur la souris**
  - `ButtonPress`, `ButtonRelease`
- o **Déplacement de la souris**
  - `MotionNotify`
  - **(relativement à une fenêtre)** `EnterNotify`, `LeaveNotify`
- o **Modification du foyer du clavier**
  - `FocusIn`, `FocusOut`

X-I

## Evènements dus aux modifications dans l'affichage et la configuration

- **On distingue les évènements qui ont déjà eu lieu, suffixés par `Notify` et ceux qui sont adressé (en général au gestionnaire de fenêtres) en réponse à une requête d'un client et suffixés par `Request`.**
- **Exemples**
  - **Création et destruction d'une fenêtre :**  
`CreateNotify`, `DestroyNotify`
  - **Modification de la géométrie d'un fenêtre et de la hiérarchie :**  
`ConfigureNotify`, `ReparentNotify`
  - **Affichage potentiel de la fenêtre (elle peut être cachée par ses soeurs)**  
`MapNotify` et **inversement** `UnmapNotify`
  - **Une partie occultée de la fenêtre devient visible**  
`Expose`

## X-I

## Communication entre clients ressources du serveur

- o événements permettant à un client de communiquer avec un autre client (ex: le copier/coller) : X définit des primitives pour la communication entre clients, via le serveur, au moyen de *propriétés*, ou de *messages* de format presque libre, concoctés par les clients :

PropertyNotify	une propriété d'une fenêtre a changé
SelectionNotify	indique la fin de la transmission de données
SelectionClear	le propriétaire de la sélection vient de changer
ClientMessage	tout message de format libre, émanant d'un client

- Les messages sont des données envoyés par un client au serveur au moyen de la fonction XSendEvent qui joue simplement le rôle de facteur.
- 
- o événements avertissant de modifications survenant sur les structures internes (ressources du serveur)
    - table de couleurs ColormapNotify : la table de couleurs installée a changé.
    - table de touches du clavier MappingNotify : la table de correspondances entre numéro de touches et codes de touches a changé.

## X-I

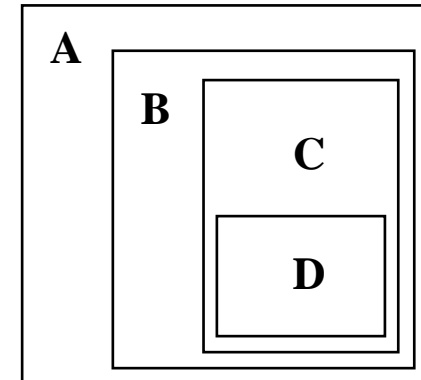
## Sélection des événements

- Un événement a lieu dans une fenêtre donnée qui en est le *siège* ou la *source*.
  - | Un événement lié à la souris a pour siège la fenêtre la plus interne contenant l'emplacement de la souris.
  - | Un événement lié au clavier a pour siège la fenêtre qui possède le *foyer du clavier*, sauf si la souris est dans une de ses descendantes; le siège est alors cette fenêtre.
- Un événement peut cependant être destiné une autre fenêtre qui en est le *destinataire*. Par exemple, une fenêtre pour laquelle `SubstructureNotifyMask` est sélectionné, est destinataire des événements de modification de structure dont le siège est une fille.
- Pour qu'une fenêtre reçoive un événement, elle doit l'avoir sélectionné, et être affichée. La sélection se fait :
  - par `XSelectInput(dpy, fenetre, masque)`
  - par affectation du champ `event_mask` de la structure `XSetWindowAttributes`
- Presque tous les événements peuvent être sélectionnés. Ceux qui ne sont pas sélectionnables (on dit "non masquables") sont envoyés d'office aux clients (par exemple `MappingNotify`).

## X-I

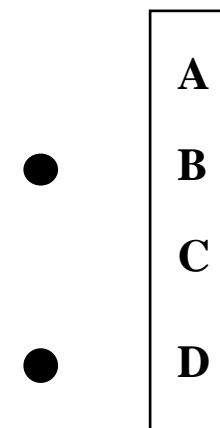
Propagation des évènements

- Lorsqu'un évènement se produit dans une fenêtre *f*, le serveur cherche les clients intéressés par cet évènement.
- Pour chaque client, le serveur remonte l'arbre des fenêtres, à partir de la fenêtre source vers la racine, à la recherche d'une fenêtre pour laquelle l'évènement a été sélectionné :
  - | si un masque correspondant a été défini pour la fenêtre *f*, l'évènement est envoyé;
  - | sinon, si un masque correspondant a été défini pour la *mère* de *f*, ou sinon pour la *grand-mère* etc, l'évènement est envoyé.
- L'évènement n'est donc pas envoyé si le client n'a choisi cet évènement sur aucune des ancêtres de la fenêtre *f*.
- Ce mécanisme de *propagation* ne s'applique qu'aux évènements provenant de la souris et du clavier.



Client X

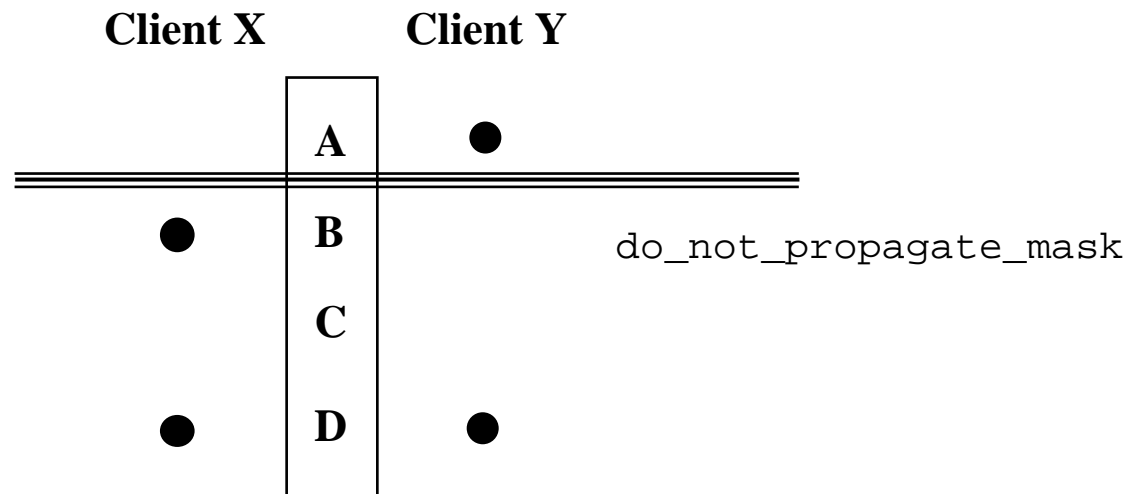
Client Y



## X-I

## Empêcher la propagation

- On peut empêcher la propagation des évènements par l'attribut de fenêtre `do_not_propagate_mask`.
- Les évènements couverts par le masque ne sont pas transmis à la mère.
- C'est une décision qui concerne une fenêtre, et elle s'applique à *tous* les clients, même si elle est prise par *un* client.
- Le client Y ne reçoit plus les évènements dont le siège est la fenêtre C:



## X-I

## Réception des évènements

- o Les évènements envoyés par le serveur au client sont emfilés dans une file.
- o Le client récupère le premier élément de la file par appel de la fonction  

```
void XNextEvent(Display *dpy, XEvent *evmt)
```
- o Si la file est vide, toutes les requêtes de la file du client sont envoyées au serveur; ceci peut être fait “manuellement” par  

```
void XFlush(Display *dpy)
```
- o et l’application se met en attente d’un évènement.
- o Le client peut se contenter de consulter la file  

```
int XPending(Display *dpy)
```

 donne le nombre d’évènements en attente ;  

```
void XPeekEvent(Display *dpy, XEvent *evmt)
```

  
**retourne le prochain évènement sans l’extraire de la file.**
- o Le client peut aussi *filtrer* les évènements  

```
void XIfEvent(Display *dpy, XEvent *evmt,
              Bool (*predicate)(),
              char *arguments)
```

  
**retourne le prochain évènement, s’il existe, qui vérifie la condition spécifiée par la procédure booléenne `predicate` avec les paramètres passés en arguments.**
- o **Variante non bloquante** : `XCheckIfEvent`.

X-I

## Informations liées aux évènements

- Les informations pertinentes à chaque évènement sont réunies dans une structure.
- Plusieurs évènements peuvent se partager la même structure.
- Les structures sont groupées dans une structure `union`, de sorte à partager certains champs entre les structures.
- Certaines informations sont communes à presque toutes les structures. Elles concernent
  - | le type de l'évènement;
  - | le numéro de la dernière requête traitée par le serveur;
  - | le moment, en millisecondes, où l'évènement s'est produit;
  - | le display concerné
  - | une fenêtre (qui peut être le siège, le destinataire)
- Des informations plus spécifiques concernent :
  - | le numéro du bouton enfoncé ou relâché
  - | les coordonnées du pointeur de la souris
  - | etc

## X-I

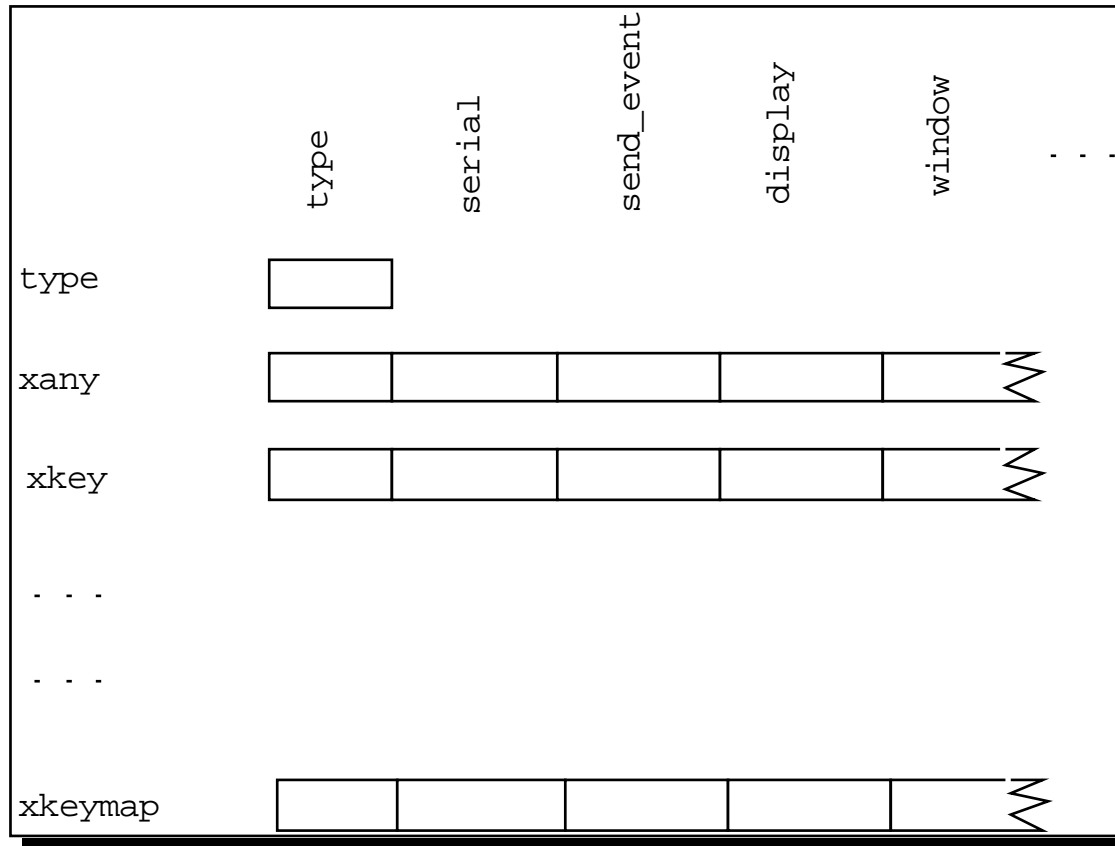
```

typedef union _XEvent {
    int type;                /* must not be changed; first element */
    XAnyEvent                xany;
    XKeyEvent                xkey;
    XButtonEvent             xbutton;
    XMotionEvent             xmotion;
    XCrossingEvent           xcrossing;
    XFocusChangeEvent        xfocus;
    XExposeEvent              xexpose;
    XGraphicsExposeEvent     xgraphicsexpose;
    XNoExposeEvent            xnoexpose;
    XVisibilityEvent         xvvisibility;
    XCreateWindowEvent       xcreatewindow;
    XDestroyWindowEvent     xdestroywindow;
    XUnmapEvent              xunmap;
    XMapEvent                xmap;
    XMapRequestEvent         xmaprequest;
    XReparentEvent           xreparent;
    XConfigureEvent          xconfigure;
    XGravityEvent            xgravity;
    XResizeRequestEvent      xresizerequest;
    XConfigureRequestEvent   xconfigurerequest;
    XCirculateEvent          xcirculate;
    XCirculateRequestEvent   xcirculaterequest;
    XPropertyEvent           xproperty;
    XSelectionClearEvent     xselectionclear;
    XSelectionRequestEvent   xselectionrequest;
    XSelectionEvent          xselection;
    XColormapEvent           xcolormap;
    XClientMessageEvent      xclient;
    XMappingEvent            xmapping;
    XErrorEvent              xerror;
    XKeymapEvent             xkeymap;
    long pad[24];
} XEvent;

```

X-I

union des types




**X-I**

Les champs *spécifiques* les plus importants sont, dans un évènement `evmt` :

- o **déclaré de type** `XButtonEvent` :
  - `evmt.x`
  - `evmt.y`
  - `evmt.button`
  - `evmt.state`
- o **déclaré type** `XKeyEvent` :
  - `evmt.keycode`
  - `evmt.state`
- o **déclaré de type** `XEvent`:
  - `evmt.xbutton.x`
  - `evmt.xbutton.y`
  - `evmt.xbutton.button`
  - `evmt.xbutton.state`
- o **déclaré type** `XEvent` :
  - `evmt.xkey.keycode`
  - `evmt.xkey.state`

```
typedef struct {
    int type;
    unsigned long serial;      /* # of last request processed by server */
    Bool send_event;         /* true if this came from a SendEvent request */
    Display *display;        /* Display the event was read from */
    Window window;          /* window on which event was requested in event mask */
} XAnyEvent;
```

X-I

## Exemple : KeyPress

```
typedef struct {
    int type; /* of event */
    unsigned long serial; /* # of last request processed by server */
    Bool send_event; /* true if this came from a SendEvent request */
    Display *display; /* Display the event was read from */
    Window window; /* "event" window it is reported relative to */
    Window root; /* root window that the event occurred on */
    Window subwindow; /* child window */
    Time time; /* milliseconds */
    int x, y; /* pointer x, y coordinates in event window */
    int x_root, y_root; /* coordinates relative to root */
    unsigned int state; /* key or button mask */
    unsigned int keycode; /* detail */
    Bool same_screen; /* same screen flag */
} XKeyEvent;
typedef XKeyEvent XKeyPressedEvent;
typedef XKeyEvent XKeyReleasedEvent;
```

## X-I

KeyPress ou KeyRelease (suite)

```
Window window;          /* "event" window it is reported relative to */
Window subwindow;      /* child window */
unsigned int state;    /* key or button mask */
unsigned int keycode;  /* detail */
```

---

window      **la fenêtre vers laquelle l'évènement a été propagé : fenêtre *destinataire*.**

subwindow    **vaut 0 si la fenêtre *siège* est la fenêtre *destinataire*,  
est la fille contenant la fenêtre *siège*, sinon.**

state        **donne l'état des touches d'altération (Shift, Control, CapsLock, Meta)**

keycode     **donne le numéro de la touche de clavier enfoncée ou relâchée**

---

**Exemple :** (KeyPress sélectionné sur A et D.)

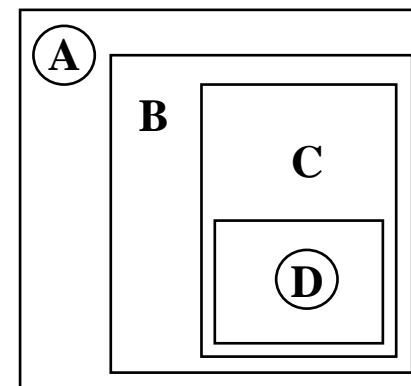
- Lorsque l'évènement se produit dans la fenêtre B ou C,

    | window = A

    | subwindow = B

- Lorsqu'il se produit dans A ou D,

    | subwindow = 0.



X-I

## Exemple : ButtonPress

```
typedef struct {
    int type; /* of event */
    unsigned long serial; /* # of last request processed by server */
    Bool send_event; /* true if this came from a SendEvent request */
    Display *display; /* Display the event was read from */
    Window window; /* "event" window it is reported relative to */
    Window root; /* root window that the event occurred on */
    Window subwindow; /* child window */
    Time time; /* milliseconds */
    int x, y; /* pointer x, y coordinates in event window */
    int x_root, y_root; /* coordinates relative to root */
    unsigned int state; /* key or button mask */
    unsigned int button; /* detail */
    Bool same_screen; /* same screen flag */
} XButtonEvent;
typedef XButtonEvent XButtonPressedEvent;
typedef XButtonEvent XButtonReleasedEvent;
```

X-I

## Les champs de XButtonEvent

<code>Window root;</code>	<b>racine de l'arbre dans lequel s'est produit l'évènement</b>
<code>Window window</code>	<b>fenêtre destinataire de l'évènement</b>
<code>Window subwindow;</code>	<b>fenêtre dans laquelle s'est produit l'évènement</b>
<code>Time time;</code>	<b>temps en millisecondes (49,7 jours sur un unsigned long)</b>
<code>int x, y;</code>	<b>position du pointeur dans la fenêtre destinataire</b>
<code>int x_root, y_root;</code>	<b>position du pointeur dans la fenêtre racine</b>
<code>unsigned int state;</code>	<b>masque fournissant l'état des <i>altérateurs</i> avant l'évènement</b>
<code>unsigned int button;</code>	<b>numéro du bouton enfoncé ou revé : Button1 à Button5.</b>
<code>Bool same_screen;</code>	<b>le pointeur est-il sur le même écran que window ?</b>

## X-I

## Capture

- **Capture** (“grab”): monopoliser la souris et/ou le clavier à l’usage d’un client unique (le “capteur”)
- **Effet** : tous les événement relevant de la capture (souris/clavier) sont
  - | envoyés uniquement au client capteur, quelle que soit la fenêtre où ils se sont produits;
  - | *pour* une fenêtre particulière, appelé la fenêtre *accaparente* (grab-window).
- Les autres fenêtres et les autres clients ne reçoivent rien, les masques de sélection sont inopérants.
- **Variation** : sur demande du client capteur (pour le `ButtonPress`, avec `OwnerGrabButtonMask`), le serveur peut moduler la fenêtre destinataire : les fenêtres destinataires sont celles où se trouve la souris.
- On peut demander explicitement la capture :
  - | capture *active* (ou immédiate): prend effet au moment où la requête est traitée (ou à la date passée en paramètre). Doit être arrêtée explicitement.
  - | capture *passive* (ou conditionnelle): prend effet lorsqu’une combinaison de touches et de boutons est enfoncée (menus !). S’arrête lorsque les touches et boutons sont relâchés.

X-I

## Capture automatique par ButtonPress

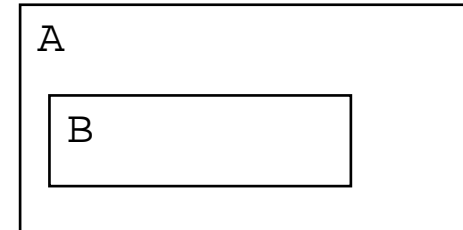
***Capture active*** : l'évènement `ButtonRelease` correspondant est envoyé au même client que celui qui a reçu l'évènement `ButtonPress`. Par conséquent,

- o l'évènement `ButtonRelease` n'est délivré à aucun autre client.
- o il est délivré pour la fenêtre qui a obtenu le `ButtonPress`, par défaut.
- o avec `OwnerGrabButtonMask`, le `ButtonRelease` il est délivré pour la fenêtre contenant le pointeur(mais toujours au client qui a reçu le `ButtonPress`).
  
- o La capture est *automatique*, i.e. n'a pas besoin d'être demandée.
- o ***Conséquence*** de l'automatisme : un seul client peut sélectionner un évènement `ButtonPress` sur une fenêtre. Sur la fenêtre racine, c'est le gestionnaire de fenêtres.

**X-I****Exemple****o Programme :**

```
Window A,B;
XSelectInput(dpy, A, ButtonPressMask);
XSelectInput(dpy, B, ButtonReleaseMask);

switch(evm.type){
  case ButtonPress :
    XMapWindow(dpy,B);
    break;
  case ButtonRelease :
    XUnmapWindow(dpy,B);
    break;
  ...
}
```



- o Question : comment faire disparaître la fenêtre B ?**

**o Variante 1 :**

```
XSelectInput(dpy, B, ButtonPressMask | ButtonReleaseMask);
```

**o Variante 2 :**

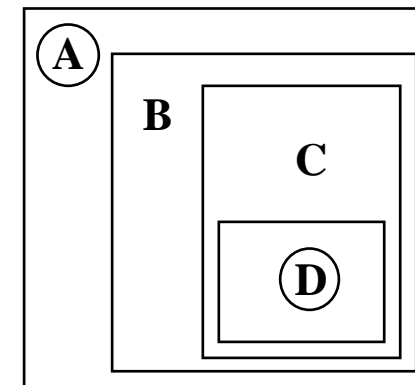
```
XSelectInput(dpy, A, ButtonPressMask | OwnerGrabButtonMask);
```

## X-I

Capture : étude des champs

- o window : **par défaut, le champ d'un évènement ButtonRelease contient la même fenêtre que celle du ButtonPress associé, même si l'évènement n'a pas eu lieu dans cette fenêtre : il y a eu *capture* du pointeur.**
- o **Exemple :**

o BP <b>dans</b> A w:A sw:0	BR <b>dans</b> A w:A sw:0	BR <b>hors</b> A w:A sw:0
o BP <b>dans</b> A w:A sw:0	BR <b>dans</b> B,C,D w:A sw:B	BR <b>hors</b> A w:A sw:0
o BP <b>dans</b> B,C w:A sw:B	BR <b>dans</b> B,C,D w:A sw:B	BR <b>d/h</b> A w:A sw:0
o BP <b>dans</b> D w:D sw:0	BR <b>dans</b> A,B,C,D w:D sw:0	BR <b>ou hors</b> A w:A sw:0
o BP <b>dans</b> D w:D sw:0	BR <b>dans</b> A,B,C w:A sw:B	BR <b>hors</b> A w:D sw:0



← OwnerGrabButtonMask  
sur D

**X-I**

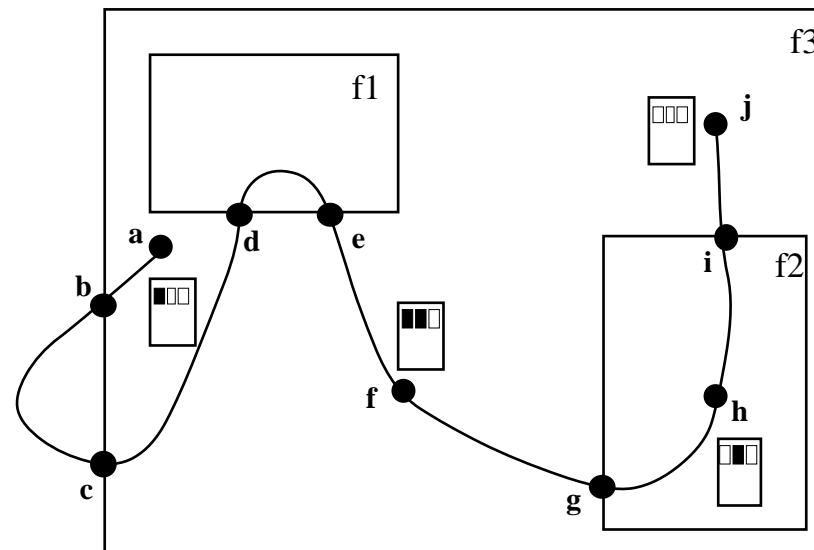
**Exemple**

La fenêtre f3 contient deux fenêtres filles, f1 et f2

o Evènements rapportés :

sans Ogbm      avec Ogbm sur f3

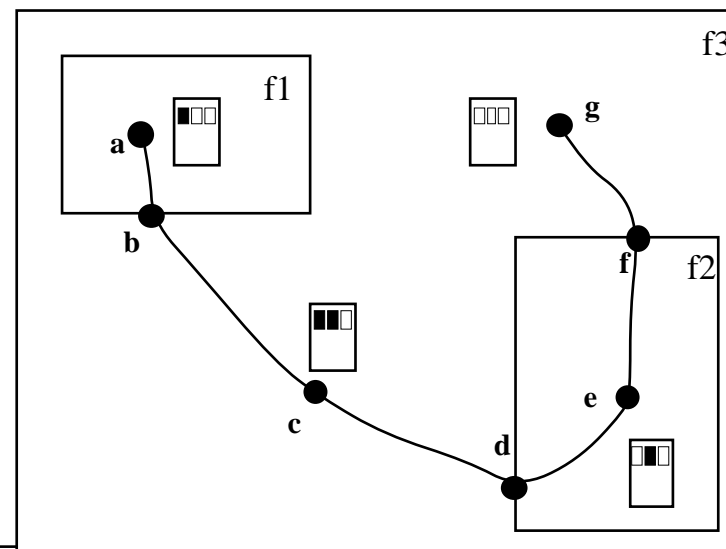
a	BP 3	BP 3
b	L 3	L 3
c	E 3	E 3
d	L 3	L 3 / E 1
e	E 3	L 1 / E 3
f	BP 3	BP 3
g	L 3	L 3 / E 2
h	BR 3-2	BR 2
i	E 3	L 2 / E 3
j	BR 3	BR 3



o Evènements rapportés :

sans Ogbm      avec Ogbm

a	BP 1	BP 1
b	L 1	L 1 / E 3
c	BP 1	BP 3
d	-	L 3 / E 2
e	BR 1	BR 2
f	-	L 2 / E 3
g	BR 1	BR 3
	L 1 / E 3	L 1 / E 3



X-I

## Capture sur une clé

**On peut faire une capture explicite, sur une clé, par la fonction :**

```
XGrabKey(dpy, code_touche, alterateurs, fen_accaparante,  
         evmts_capteur, mode_souris, mode_clavier);
```

- o `code_touche` : **un numéro de touche, par exemple**  
`XKeysymToKeycode(dpy, XK_x);`
- o `alterateurs` : **un masque comme** `ControlMask | ShiftMask`.
- o `evmt_capteur` : **est True ou False :**
  - | **si True, le client reçoit les évènements normalement;**
  - | **si False, la fenêtre accaparante reçoit tous les évènements;**
- o `mode_souris` **et** `mode_clavier` **sont** `GrabModeSync` **ou** `GrabModeAsync`:
  - | `GrabModeSync` : **le serveur met en réserve les évènements, sans les envoyer au clients (à utiliser à prudence).**
    - » **La souris ou le clavier sont dits “gelé”.**
    - » **Seul une requête explicite** `XAllowEvents` **permet l’envoi; après, la souris ou le clavier sont à nouveau gelés.**
  - | `GrabModeAsync` : **le serveur envoie les évènements.**

## Masques et évènements : souris

### *Mouvements de la souris*

- o **Différents masques pour sélectionner les évènements**
  - **PointerMask:** mouvement de la souris, sans restriction
  - **ButtonMask:** mouvement de la souris avec bouton enfoncé
  - **Button<n>Mask:** mouvement de la souris, avec le bouton <n> enfoncé
  - **EnterWindowMask:** entrée de la souris dans la fenêtre
  - **LeaveWindowMask:** sortie de la souris de la fenêtre
- o **Les 3 premiers masques sont associés à la même structure `XMotionEvent`, les deux derniers à la structure `XCrossingEvent`. L'entrée ou la sortie peut résulter d'un déplacement de la souris ou de la fenêtre. Tous les détails du cheminement dans l'arbre des fenêtres peuvent être récupérés.**
- o **Le masque `PointerMotionHintMask` n'est pas un masque comme les autres. Il indique qu'au lieu de recevoir tous les évènements de mouvement, un seul est envoyé au client, résumant une séquence de mouvements consécutifs.**
- o **La position du curseur peut être consultée par appel à `XQueryPointer`. Ceci permet d'ignorer les évènements intermédiaires.**

X-I

## Masques et évènements : souris

### *Boutons de la souris*

- o **Deux masques pour sélectionner les évènements**
  - ButtonPressMask:            **un bouton est enfoncé**
  - ButtonReleaseMask:        **un bouton est relâché.**
- o **La même structure** XButtonEvent **couvre les évènements** ButtonPress **et** ButtonRelease. **La structure a les synonymes** XButtonPressEvent **et** XButtonReleaseEvent.
- o **X prévoit jusqu'à 5 boutons. La détermination du bouton concerné se fait en examinant le champ** button. **Les constantes** Button1,..., Button5 **sont prédéfinies. Le champ** state **donne un masque contenant les altérateurs et les** *autres* **boutons enfoncés au moment de l'évènement.**

X-I

## Masques et évènements : clavier

- o **Différents masques pour sélectionner les évènements concernant le clavier:**
  - `KeyPressMask`: **enfonce une touche**
  - `KeyReleaseMask`: **relache une touche**
  - `FocusChangeMask`: **le foyer du clavier change**
  - `KeymapStateMask`: **état du clavier.**
  - **Sans masque, on obtient d'office les évènements `MappingNotify`.**
- o **La structure `XKeyEvent` associé aux évènements sélectionnés par les deux premiers masques contient le numéro de la touche, et l'état des altérateurs et des boutons de souris, au moment où l'évènement a eu lieu.**
- o **Le *foyer du clavier* est la fenêtre destinataire des évènements `KeyPress` et `KeyRelease`.**
- o **Le changement du foyer est en général laissé au gestionnaire de fenêtres, mais peut être gérée directement, par la fonction `XSetInputFocus`.**
- o **La fenêtre recevant effectivement les évènements clavier est la suivante:**
  - | **Si la fenêtre `s` contenant le pointeur de souris est descendante de la fenêtre foyer `f`, c'est la fenêtre `s`, sinon c'est la fenêtre `f`.**
  - | **Puis, il y a propagation usuelle de l'évènement.**
- o **Exemple : si le foyer est la fenêtre racine de l'écran, c'est la fenêtre contenant la souris qui est systématiquement choisie.**

X-I

## Masques et évènements : clavier

- o **Sans masque, on obtient d'office les évènements `MappingNotify`. Ils indiquent un changement de la correspondance entre numéros de touches (`Keycode`) et nom de touche symbolique (`Keysym`).**
  - | **Un tel changement peut être fait, à tout moment, par tout client.**
  - | **Il existe un client spécifique (`xmodmap`) pour personnaliser la correspondance.**
- o **La table de correspondance est stockée dans le serveur, et chargée dans le client au moment de la connexion.**
- o **Pour charger explicitement une table qui a été modifié, on utilise**  
`XRefreshKeyboardMapping(XMappingEvent * evmt)`
- o **Ainsi, toute bonne boucle d'évènement devrait contenir un morceau:**

```
switch(evmt.type) {  
    ...  
    case MappingNotify :  
        XRefreshKeyboardMapping(&(evmt.xmapping));  
        break;  
}
```

X-I

## Masques et évènements : clavier (fin)

- o **Le masque** `KeymapStateMask` **sélectionne la réception d'un évènement** `KeymapNotify`.
- o **Un tel évènement contient l'état du clavier, et est engendré lorsque le pointeur ou le foyer d'entrée entre dans une fenêtre. Ils sont envoyés immédiatement après un** `EnterNotify` **ou un** `FocusIn`.
- o **Le seul champ de l'évènement, de type** `XKeymapEvent`, **en plus des champs standard, est un champ défini comme**  

```
char key_vector[32]
```
- o **Chaque bit de ce vecteur (de 256 bits) représente une clé physique, l'indice étant égal au numéro de la touche (Keycode).**

## X-I

Masques et évènements : structure

- o Les masques `StructureNotifyMask` et `SubstructureNotifyMask` sélectionnent pratiquement les mêmes évènements, mais pas sur les mêmes fenêtres:
  - | `StructureNotifyMask` sélectionne sur la fenêtre sensibilisée;
  - | `SubstructureNotifyMask` sélectionne sur les *filles* de la fenêtre *f* sensibilisée. En d'autres termes, la fenêtre *f* est destinataire des modifications survenues dans les filles.
  
- o Une fenêtre peut ainsi connaître les changements intervenus dans ces filles.
- o Ces évènements ne sont pas propagés.

- |  |   |
|--|---|
| <ul style="list-style-type: none"> <li>o Masque<br/><code>SubstructureNotify</code></li> </ul> | <ul style="list-style-type: none"> <li>o Masque<br/><code>StructureNotify</code></li> </ul> |
|--|---|

```

CreateNotify
DestroyNotify
ConfigureNotify
CirculateNotify
GravityNotify
MapNotify
ReparentNotify
UnmapNotify
  
```

```

CreateNotify
ConfigureNotify
CirculateNotify
GravityNotify
MapNotify
ReparentNotify
UnmapNotify
  
```

## X-I

## Masques et évènements : graphisme

- o Le masque `ExposureMask` sélectionne les évènements `expose`.
- o Un tel évènement est engendré chaque fois qu'une partie d'une fenêtre devient visible. Plusieurs évènements `expose` peuvent être engendré par une seule action.
- o Le champ `count` compte le nombre d'évènement qui restent. Il suffit en général de redessiner si ce nombre est 0.
- o Les champs `x`, `y`, `width` et `height` dénotent les coordonnées du rectangle exposé. Il y a donc un évènement par rectangle.

## o Exposition :

```
Expose
GraphicsExpose
NoExpose
```

```
typedef struct {
    int type;
    unsigned long serial;      /* # of last request processed by server */
    Bool send_event;         /* true if this came from a SendEvent request */
    Display *display;        /* Display the event was read from */
    Window window;
    int x, y;
    int width, height;
    int count;                /* if non-zero, at least this many more */
} XExposeEvent;
```

X-I

## Masques et évènements : graphisme

- o Les évènements `GraphicsExpose` et `NoExpose` ne sont pas sélectionnables. Ils sont hautement spécialisés.
- o Ils sont engendrés après les primitives `XCopyArea` et `XCopyPlane`, et indiquent
  - | `GraphicsExpose` : qu'une partie de la région source n'était pas disponible au moment de la copie.
  - | `NoExpose` : qu'au contraire la région source était complètement découverte.

## X-I

Correspondance masque/événement/structure**Masques****Événements****Structures***pas de masque**pas d'événement*

xany

*pas de masque**pas d'événement*

xerror

NoEventMask

*pas d'événement**pas de structure*

KeyPressMask

KeyPress

xkey

KeyReleaseMask

KeyRelease

"

ButtonPressMask

ButtonPress

xbutton

ButtonReleaseMask

ButtonRelease

"

EnterWindowMask

EnterNotify

xcrossing

LeaveWindowMask

LeaveNotify

"

PointerMotionHintMask

*pas d'événement**pas de structure*

PointerMotionMask

MotionNotify

xmotion

Button1MotionMask

"

"

Button2MotionMask

"

"

Button3MotionMask

"

"

Button4MotionMask

"

"

Button5MotionMask

"

"

ButtonMotionMask

"

"

## X-I

suite...**Masques**

KeymapStateMask

ExposureMask

VisibilityChangeMask

StructureNotifyMask

"

"

"

"

"

"

ResizeRedirectMask

SubstructureNotifyMask

"

SubstructureRedirectMask

"

"

**Événements**

KeymapNotify

Expose

VisibilityNotify

CirculateNotify

ConfigureNotify

DestroyNotify

GravityNotify

MapNotify

UnmapNotify

ReparentNotify

ResizeRequest

les 7 événements comme  
pour StructureNotifyMask

CreateNotify

CirculateRequest

MapRequest

ConfigureRequest

**Structures**

xkeymap

xexpose

xvisibility

xcirculate

xconfigure

xdestroy

xgravity

xmap

xunmap

xreparent

xresizerequest

xcreatewindow

xcirculaterequest

xmaprequest

xconfigurerequest

## X-I

resulte...**Masques**

FocusChangeMask

"

PropertyChangeMask

ColormapChangeMask

OwnerGrabButtonMask

*pas de masque**pas de masque**pas de masque**pas de masque**pas de masque**pas de masque**pas de masque***Evénements**

FocusIn

FocusOut

PropertyNotify

ColormapNotify

*pas d'événement*

MappingNotify

SelectionClear

SelectionNotify

SelectionRequest

ClientMessage

GraphicsExpose

Noexpose

**Structures**

xfocus

xfocus

xproperty

xcolormap

*pas de structure*

xmapping

xselectionclear

xselection

xselectionrequest

xclient

xgraphicsexpose

xnoexpose