TR-I-0091

X Window System, Version 11 ポケット・ガイド

- Xlib 編 -

X11 Pocket Guide - Xlib -

田中 孝明
Takaharu TANAKA

1989.8

概要

本稿はプログラマ向けに「X Window System, Version 11 ポケット・ガイド」として作成したテキストを再構成したものである。

MIT で開発された X Window System は VAX をはじめ多くのコンピュータ上にインプリメントされ、事実上、グラフィック/ウィンドウ・システムの業界標準となっている。また、UNIX の標準団体である OSF のグラフィカル・ユーザインタフェースにも採用されている。

本稿が研究支援システムおよびデモンストレーション・システムを開発する際の一助となれば幸いである。なお、本稿は「X Window System, Version 11 ポケット・ガイド - Toolkit 編 -」(TR-I-0092) との二部構成となっている。

ATR 自動翻訳電話研究所
ATR Interpreting Telephony Research Laboratories

Table of Contents

Chapter 1

X Window System 概要

X (X Window System, Version 11) は MIT で開発されたネットワーク・トランスペアレントなウィンドウ・システムです。X はグラフィック処理をサーバとクライアントに分離する事によって計算処理と表示を別々のコンピュータで行えるようにしています。そしてサーバとクライアントは X Protocol というプロトコルによって通信します。

サーバとクライアント

X サーバはビットマップディスプレイを持ったワークステーションなどのコンピュータ上で動き，ウィンドウやグラフィックの出力，およびユーザからの入力（キーボードとマウスなど）の処理を行ないます。X サーバは複数のクライアントからの要求を同時に受け付け，処理することができます。サーバとクライアントを同じコンピュータ上に置くことも，ネットワークでつながった別々のコンピュータ上に置くこともできます。

X サーバに対してグラフィックの出力を要求したり，ユーザからの入力に応じたアプリケーションの実際の処理を行うのがクライアントです。クライアントは X Protocol を用いて X サーバと通信します。X のアプリケーションは，Xlib という C 言語のサブルーチン・ライブラリ（およびその上位のツールキット・ライブラリ）を用いてクライアント・プログラムとして書かれます。

X はサーバとクライアントをネットワークで分離することによって，大型のホスト上で実行しているアプリケーションの結果を安価なワークステーションでグラフィカルに表示することを可能にしています。複数のホスト上で実行しているアプリケーションを 1 台のワークステーションから操作することもできます。

ウィンドウ

X はオーバーラップ型のウィンドウ・システムです。すべてのウィンドウは親子関係で結ばれています。あるウィンドウは必ず 1 つのペアレント・ウィンドウを持ち，同時に複数のサブ・ウィンドウを持つことができます。そして，サブ・ウィンドウはペアレント・ウィンドウによってクリッピングされ，ペアレント・ウィンドウの範囲を越えて表示されることはありません。

また，ディスプレイのバックグラウンドもルート・ウィンドウと呼ばれるひとつのウィンドウです。ルート・ウィンドウはアプリケーションのトップレベル・ウィンドウのペアレント・ウィンドウとなります。

イベントとしての入力

X ではユーザからの入力は，イベントという形でサーバからクライアントに渡されます。クライアントのプログラムはサーバから送られて来るイベントを順次処理することによってユーザとのインターフェースをとります。まだ処理されていないイベントは Xlib 内のキュー上に保持されます。

サーバから送られてくるイベントはユーザの入力によるもの以外にも，ウィンドウの状態が変化したことを知らせるものや，クライアント間のデータのやり取りのためのものなどがあります。

ウィンドウの書き直し

サーバはウィンドウに出力されたグラフィックやテキストの内容について，基本的にはなにも感知しません。ですから，たとえば自分のウィンドウの手前にあったウィンドウが移動して今まで隠れていた部分が見えるようになったような場合，クライアント自身でその部分の書き直しを行わないといけません。サーバは書き直しが必要になったウィンドウに対して，Expose イベントというイベントを起こします。

出力バッファと非同期入力

クライアントの Xlib はサーバとの通信の負荷を減らすために，通常の出力はバッファリングするようになっています。ですから，Xlib の関数による出力は XSync や XNextEvent などの関数によって明示的にフラッシュしない限りサーバに渡されません。また，サーバからのリプライやイベントなども非同期に処理されるようになっています。

ピックスマップ（pixmap）

ピックスマップはディスプレイのオフ・スクリーンに作られる仮想的な描画領域です。これは頻繁に使うイメージやパターンの表示を高速にするために利用されます。ウィンドウとピックスマップを合わせて drawable と呼びます。Xlib のグラフィック関数はウィンドウだけでなく，ピックスマップ上にも描画できるようになっています。また，ピックスマップのうち単一のプレーンのみのものをビットマップ（bitmap），塗りつぶしに用いられるものをタイル（tile）と呼びます。

グラフィクス・コンテキスト（Graphics Context）

グラフィック出力やテキスト出力の色や線の太さなどの属性はグラフィクス・コンテキスト（GC）という形で指定します。GC を用いることによって複雑な属性の指定を一度に行なうことができます。また，GC はサーバ上で保持されるため，同じ GC を繰り返し使う場合には効率的となります。

リソース

いくつかの Xlib 関数はリソース ID を返します。これらの実体はサーバ側で保持，管理されています。リソースには Window, Font, Pixmap, Cursor, GContext などがあります。カーソルやフォントなどのリソースはクライアント間で共有できます。

ツールキット・ライブラリ（X Toolkit）

X には Xlib の上位に X Toolkit というライブラリ・ルーチンが用意されています。これは Xlib でのウィンドウの作成やイベントの処理を自動的に行ないます。それに加えて，ツールキット・ライブラリは widget という概念によって X のユーザ・インターフェースとプログラム環境を拡張します。widget は内部データとそのデータに対するユーザの操作，そして個々の widget がどうふるまえばよいのかを一括して管理しています。

Chapter 2

Xlib 関数

Xlib は X でウィンドウ操作をしたり，文字やグラフィックの出力，マウス入力，キーボード入力などを行なうための最も基本的な関数のライブラリです。さらに，Xlib はグラフィック・コンテキスト，テキスト操作，イベント処理など多くの高度な機能も含んでいます。

Xlib の各関数は X Protocol に直接対応して作られています。

2.1. ディスプレイの操作

Xlib ではディスプレイを開くという操作によって，サーバとの接続を行います。これには次の関数を用います。

```
Display *XOpenDisplay (display_name)
  char  *display_name;
```

Display は X11/Xlib.h で定義されています。接続に失敗した場合は 0 が返されます。この Display は Xlib の他の関数で，接続したサーバを指定するために必ず用います。引き数の display_name は次のように指定します。

```
hostname:number.screen_number
```

hostname:	サーバのホスト名。ローカルホストの UNIX ドメインを用いる場合は unix:, DECnet を用いる場合は nodename:: とする
number	ホスト上のサーバの番号
.screen_number	サーバの管理するスクリーンの番号（省略可能）

display_name に NULL や "" を指定した場合には，DISPLAY という環境変数の内容がとられます。

ディスプレイを閉じる（サーバとの接続を終了する）には

```
XCloseDisplay (display)
  Display  *display;
```

を用います。

ディスプレイとスクリーンに関する属性を得るための種々のマクロおよび関数が用意されています。

RootWindow (display, screen_number), RootWindowOfScreen (screen) はディスプレイとスクリーンのルート・ウィンドウのウィンドウ ID を返します（以下同様にスクリーンに対するマクロも用意されている。また，関数の方は XRootWindow (display, screen_number) などのような関数名になっている）。

BlackPixel (display, screen_number), WhitePixel (display, screen_number) はディスプレイであらかじめ用意されている黒色と白色のピクセル値を返します。

DefaultGC (display, screen_number), DefaultVisual (display, screen_number) でそれぞれデフォルトの GC とビジュアルを返します。

また，ScreenOfDisplay (display) と DisplayOfScreen (screen) は Display と Screen の相互変換を行ないます。

2.2. ウィンドウの操作

ウィンドウは，アプリケーションがグラフィクスやテキストを出力するためにディスプレイ上に作成する矩形の領域です。ウィンドウは X におけるもっとの基本的な要素で，イベントなどもウィンドウを対象に起ります。

2.2.1. ウィンドウの作成と削除

ウィンドウの作成は次の関数で行ないます。

```
Window XCreateWindow (display, parent, x, y, width, height, border_width,
                      depth, class, visual, valuemask, attributes)
  Display  *display;
  Window  parent;
  int  x, y;
  unsigned int  width, height;
  unsigned int  border_width;
  int  depth;
  unsigned int  class;
  Visual  *visual
  unsigned long  valuemask;
  XSetWindowAttributes  *attributes;

Window XCreateSimpleWindow (display, parent, x, y, width, height,
                            border_width, border, background)
  Display  *display;
  Window  parent;
  int  x, y;
  unsigned int  width, height, border_width;
  unsigned long  border;
  unsigned long  background;
```

それぞれの戻り値はウィンドウ ID です。

XCreateWindow はおもにアプリケーションのトップレベル・ウィンドウを作るために，XCreateSimpleWindow はそのサブ・ウィンドウのために用います。XCreateSimpleWindow で作成したウィンドウはペアレント・ウィンドウの属性を受け継ぎます。

引き数の display には接続したディスプレイ，parent はペアレント・ウィンドウのウィンドウ ID，x，y はウィンドウの位置（ペアレント・ウィンドウ上での座標，左上が原点），width，height はウィンドウの幅と高さ，border_width はウィンドウのボーダー（枠）の幅を指定します。

depth はウィンドウで用いるプレーンの数です（0 を指定するとペアレント・ウィンドウの値がとられる）。class はウィンドウの入出力クラスで，これには InputOutput（入力と出力ができる通常のウィンドウ）と InputOnly（キーおよびマウス入力のみのもの），CopyFromParent（ペアレント・ウィンドウのクラスと同じクラスとする）の指定があります。visual はウィンドウでの色の使い方を指定するものです（カラーマップの操作の項を参照のこと）。attributes ではウィンドウの各種の属性を指定し，valuemask でどの属性を指定したいのかを示します（CWBackPixmap，CWBitGravity などのマスク）。このウィンドウの属性は

```
typedef struct {
    Pixmap  background_pixmap;     /* background or None or ParentRelative */
    unsigned  long background_pixel; /* background pixel */
    Pixmap  border_pixmap;         /* border of the window */
    unsigned  long border_pixel;   /* border pixel value */
    int  bit_gravity;              /* one of bit gravity values */
    int  win_gravity;              /* one of the window gravity values */
    int  backing_store;            /* NotUseful, WhenMapped, Always */
    unsigned  long backing_planes; /* planes to be preseved if possible */
    unsigned  long backing_pixel;  /* value to use in restoring planes */
    Bool  save_under;              /* should bits under be saved? (popups) */
    long  event_mask;              /* set of events that should be saved */
    long  do_not_propagate_mask;   /* set of events that should not propagate */
    Bool  override_redirect;       /* boolean value for override-redirect */
    Colormap  colormap;            /* color map to be associated with window */
    Cursor  cursor;                /* cursor to be displayed (or None) */
} XSetWindowAttributes;
```

という構造体で指定します。各メンバの意味は次の通り（かっこ内は valuemask で指定するマスクとデフォルトの値）。

background_pixmap, background_pixel
: ウィンドウのバックグランド（背景）の色およびタイル・パターン。ピックスマップとピクセル値のどちらかを指定する（CWBackPixmap, CWBackPixel : None）

border_pixmap, border_pixel
: ボーダー（枠）の色およびタイル・パターン。バックグランド

と同様にどちらかを指定する（CWBorderPixmap, CWBorder-Pixel : CopyFromParent）

bit_gravity　ウィンドウがリサイズされた時，元の内容をどういう位置に持っていくか指定。NotrhWestGravity, NorthWestGravity, CenterGravity, StaticGravity などがある（CWBitGravity : ForgetGravity）

win_gravity　ウィンドウがリサイズされた時，サブ・ウィンドウをどういう位置に配置するかの指定。bit_gravity のものに加えてUnmap-Gravity がある（CWWinGravity : NorthWestGravity）

backing_sotre, backing_palnes, packing_pixel
サーバによるウィンドウの内容の保存の指定。backing_sotre には NotUseful, WhenMapped, Always がある（CWBackingStore, CWBackingPlanes, CWBackingPixel : NotUseful）

save_under　そのウィンドウによって隠された領域の内容の保存の指定（CWSaveUnder : False）

event_mask　ウィンドウが処理するイベントのイベント・マスク（CWEventMask : NoEventMask）

do_not_propagete_mask
サブ・ウィンドウに渡させないイベントのイベント・マスク（CWDontPropagate : NoEventMask）

override_redirect
ウィンドウ・マネージャによるウィンドウの装飾の禁止（CWOverrideRedirect : False）

colormap　ウィンドウが使用するカラーマップ（CWColormap : CopyFrom-Parent）

cursor　マウス・カーソルのウィンドウ上での形（CWCursor : None）

ウィンドウとサブ・ウィンドウの削除（破壊）は

```
XDestroyWindow (display, w)
  Display  *display;
  Window  w;

XDestroySubwindows (display, w)
   Display  *display;
   Window  w;
```

で行います。ペアレント・ウィンドウが削除されるとサブ・ウィンドウも自動的に削除されます。

2.2.2. ウィンドウの属性の変更

すでにあるウィンドウの幅，高さ，位置，ボーダーなど幾何学的な値（geometry）を変更するには XMoveWindow (display, w, x, y), XResizeWindow (display, w, width, height) などの関数を用います。また，XConfigureWindow (display, w, value_mask, values)（ここでの values は XWindowChanges という構造体）で一度に複数の値を変更することもできます。

ウィンドウの他の属性も同様に XSetWindowBackGround (display, w, background_pixel), XSetWindowBorderPixmap (display, w, border_pixmap) など個別のものと XChangeWindowAttrributes (display, w, valuemask, attributs)（attributes は XSetWindowAttributes）で変更することができます。

2.2.3. ウィンドウのマップ

作成されたウィンドウはディスプレイにマップするまでスクリーンに表示されません。また，イベントも発生しません。ウィンドウのマップ，アンマップは

```
XMapWindow (display, w)
  Display  *display;
  Window  w;

XMapSubwindows (display, w)
  Display  *display;
  Window  w;

XUnmapWindow (display, w)
  Display  *display;
  Window  w;

XUnmapSubwindows (display, w)
  Display  *display;
  Window  w;
```

で行ないます。サブ・ウィンドウはマップされていても，ペアレント・ウィンドウが実際にマップされるまでは表示されません。また，ペアレント・ウィンドウがアンマップされると，サブ・ウィンドウは見えなくなります。

2.2.4. その他のウィンドウ操作

この他にもウィンドウの大きさや位置などの属性を調べたり，ウィンドウの property list（それぞれのウィンドウに対して名前付きのデータをサーバに管理させる仕組み）を操作する関数などがあります。

2.3. グラフィクス・コンテキスト（GC）

グラフィックや文字を出力する場合に必要となる種々の属性はグラフクス・コンテキスト（GC）を用いて指定します。GC はサーバで描画属性をバッファリングし，出力を高速化するために考え出されたものです。すべてのグラフィック関数には GC を指定します。

GC を作るには

```
GC XCreateGC (display, d, valuemask_create, values)
  Display  *display;
  Drawable  d;
  unsigned long  valuemask_create;
  XGCValues  *values;
```

を用います。XGCValues で GC の各属性を指定し，valuemask_create（GCFunction, GCFont など）のマスクでどの属性を指定したいのかを指示します。各グラフィック関数はそれぞれの必要なものを用いて描画を行ないます。XGCValues は次のようになっています。

```
typedef struct {
    int  function;              /* logical operation */
    unsigned long  plane_mask;  /* plane mask */
    unsigned long  foreground;  /* foreground pixel */
    unsigned long  background;  /* background pixel */
    int  line_width;            /* line width */
    int  line_style;            /* LineSolid, LineOnOffDash, LineDoubleDash */
    int  cap_style;             /* CapNotLast, CapButt,
                                   CapRound, CapProjecting */
    int  join_style;            /* JoinMiter, JoinRound, JoinBevel */
    int  fill_style;            /* FillSolid, FillTiled,
                                   FillStippled, FillOpaeueStippled */
    int  fill_rule;             /* EvenOddRule, WindingRule */
    int  arc_mode;              /* ArcChord, ArcPieSlice */
    Pixmap  tile;               /* tile pixmap for tiling operations */
    Pixmap  stipple;            /* stipple 1 plane pixmap for stipping */
    int  ts_x_origin;           /* offset for tile or stipple operations */
    int  ts_y_origin;
    Font  font;                 /* default text font for text operations */
    int  subwindow_mode;        /* ClipByChildren, IncludeInferiors */
    Bool  graphics_exposures;   /* boolean, should exposures be generated */
    int  clip_x_origin;         /* origin for clipping */
    int  clip_y_origin;
    Pixmap  clip_mask;          /* bitmap clipping; other calls for rects */
    int  dash_offset;           /* patterned/dashed line information */
    char  dashes;
} XGCValues;
```

各メンバの意関は次の通り（かっこ内は valuemask_create のマスクとデフォルト）。

function　　グラフィック・ファンクション。GXand, GXor, GXandInverted など描こうとするビットと元のビッドの論理演算の全組み合わせが用意されている（GCFunction：GXcopy）

plane_mask　　描画処理の対象となるプレーンのマスク（GCPlaneMask：AllPlanes()）

foreground　　フォアグランドのピクセル値（GCForeground：0）

background　　バックグランドのピクセル値（GCBackground：1）

line_width　　線の幅（ピクセル数）。0 でディスプレイでサポートしている最も細い線となる（GCLineWidth：0）

line_style　　線の種類。LineSolid, LineDoubleDash, LineOnOffDash がある（GCLineStyle：LineSolid）

cap_style	線の始点終点の形状。CapNotLast, CapButt, CapRound, CapProjecting がある（GCCapStyle：CapButt）
join_style	折れ線の角の形状。JinMiter, JoinRound, JoinBevel がある（GCJoinStyle：JoinMiter）
fill_style	塗りつぶしの方法。FillTiled, FillTiled, FillOpaqueStippled, FillStippled がある（GCFillStyle：FillSolid）
fill_rule	XFillPolygon での塗りつぶしのアルゴリズム。EvenOddRule, WindingRule がある（GCFillRule：EvenOddRule）
arc_mode	XFillArc での弧の塗りつぶしの方法。ArcPieSlice と ArcChord がある（GCArcMode：ArcPieSlice）
tile	FillTiled で用いるタイルパターン（GCTile：なし）
stipple	FillStippled と FillOpaqueStippled で用いるバックグランドのパターン（GCStipple）
ts_x_origin, ts_y_origin	タイルと stipple の始点（GCTileStipXOrigin, GCTileStipYOrigin：0, 0）
font	テキスト出力時のフォント ID（GCFont：サーバによる）
subwindow_mode	サブ・ウィンドウがあった場合の処理（GCSubwindowMode：ClipByChildren）
graphics_exposures	XCopyArea と XCopyPlane の際に GraphicsExpose イベントを起こさせるかどうか（GCGraphicsExposures：True）
clip_x_origin, clip_y_origin	clip mask の始点（GCClipXOrigin, GCClipYOrigin：0, 0）
clip_mask	描画をクリッピングして出力するためのマスク（GCClipMask：None）
dash_offset	破線のパターンの開始位置（GCDashOffset：0）
dashes	破線のパターン（GCDashList：4）

GC の属性のうち，破線のパターンは XSetDashes (display, gc, dash_offset, dash_list, n) で指定します。タイルと stipple については XQueryBestSize (display, class, whitch_screen, width, height, width_return, height_return) でディスプレイに合った大きさを求めてから作成します（class は TileSape, CursorShape, StippleShape のいずれか）。

XChangeGC (display, gc, valuemask_change, values) ですでにある GC の属性を変更することができます。また，GC の一部分だけを変更するために XSetFunction (display, gc, function), XLineAttributes (display, gc, line_width, line_style, cap_style, join_style) などの関数も用意されています。XCopyGC (display, src, valuemask_copy, dest) で他の GC の属性をコピーすることができます。

GC の削除は XFreeGC (display, gc) で行ないます。

GC という値は実際には Xlib で作られる構造体へのポインタです。GC のリソース ID（GContext）を得るには XGContextFromGC (gc) を用います。

2.4. グラフィック関数

2.4.1. 点，線，弧の出力

X にはグラフィック関数として，点，直線，折れ線，セグメント（複数の直線），矩形，多角形，円弧のための関数が用意されています。また，それらの塗りつぶしを行なう関数もあります。GC で塗りつぶしのタイル・パターンや，線の幅，接続点の形などを指定します。描画はウィンドウだけではなく，ピックスマップに対しても行うことができます。ウィンドウとピックスマップを合わせて drawable と呼びます。

1 つまたは複数の点の出力は

```
XDrawPoint (display, d, gc, x, y)
  Display  *display;
  Drawable  d;
  GC  gc;
  int  x, y;

XDrawPoints (display, d, gc, points, npoints, mode)
  Display  *display;
  Drawable  d;
  GC  gc;
  XPoint  *points;
  int  npoints
  int  mode;
```

mode　points の座標値の指定法。CoordModeOrigin で絶対指定，CoordModePrevious で前の座標値からの相対指定となる

で行ないます。XPoint は次のような構造体です。

```
typedef struct {
  short  x, y;
} XPoint;
```

同様に，連続した直線（折れ線）を描く

```
XDrawLine (display, d, gc, x1, y1, x2, y2),
XDrawLines (display, d, gc, points, npoints, mode),
```

複数の直線を描く

```
XDrawSegments (display, d, gc, segments, nsegments),
```

矩形を描く

```
XDrawRectangle (display, d, gc, x, y, width, height),
XDrawRectangles (display, d, gc, rectrangles, nrectangles),
```

円弧を描く

```
XDrawArc (display, d, gc, x, y, width, height, angle1, angle2),
XDrawArcs (display, d, gc, arcs, narcs)
```

x, y	円弧を囲む矩形の左上の頂点の座標
width,	円弧（円弧を囲む矩形）の幅と高さ
angle1	弧の始まる位置。3 時の方向から数えた角度。単位は度の 1/64（360° が 23040 となりちょうど signed short に収まる）
angle2	angle1 から相対的に反時計回りで数えた弧の終点の角度。単位は angle1 と同じ

があります。XPoint と同様な構造体

```
typedef struct {
  short  x1, y1, x2, y2;
} XSegment;

typedef struct {
  short  x, y;
  unsigned short  width, height;
} XRectangle;

typedef struct {
  short  x, y;
  unsigned short  width, height;
  short  angle1, angle2;
} XArc;
```

があります。

また，塗りつぶしでは，矩形の

```
XFillRectangle (display, d, gc, x, y, width, height),
XFillRectangles (display, d, gc, rectrangles, nrectangles)
```

多角形の

```
XFillPolygon (display, d, gc, points, npoints, shape, mode)
```

shape　点の並び方の指定。凸の多角形では Convex，凸でない場合は Nonconvex，辺が交わっている場合は Complex を指定する。前者の指定ほど高速なアルゴリズムが使われる

円弧の

```
XFillArc (display, d, gc, x, y, width, height, angle1, angle2),
XFillArcs (display, d, gc, arcs, narcs)
```

があります。

2.4.2. 領域のクリアとコピー

ウィンドウの一部分または全体をクリアするには

```
XClearArea (display, w, x, y, width, height, exposures)
  Display  *display;
  Window  w;
  int  x, y;
  unsigned int  width, height;
  Bool  exposures;

XClearWindow (display, w)
  Display  *display;
  Window  w;
```

exposures　そのウィンドウに対して Expose イベントを起こすかどうかの指定

を用います。ピックスマップにはバックグランドがないのでこの関数は使えません。

drawable のある領域を他の場所にコピーするには

```
XCopyArea (display, src, dest, gc,
        src_x, src_y, width, height, dest_x, dest_y)
  Display  *display;
  Drawable  src, dest;
  GC  gc;
  int  src_x, src_y;
  unsigned int  width, height;
  int  dest_x, dest_y;
```

を用います。また，1 プレーンのみのコピーには XCopyPlane (display, src, dest, gc, src_x, src_y, width, height, dest_x, dest_y) を用います。

2.5. テキスト出力関数

2.5.1. フォントの操作

X では色々な大きさや形の文字を出力することができます。フォントを用いてテキストの文字の種類を指定します。フォントも X のリソースの 1 つです。フォントを使用するためには，まずそのフォントをロードし，その ID を得る必要があります。これには

```
Font XLoadFont (display, name)
  Dsiplay  *display;
  char  *name;
```

を用います。フォントは複数のクライアント間で共有されます。フォントの解放は XUnloadFont (display, font) で行います。

フォントは多数の属性を持っています。これを得るには

```
XFontStruct *XQueryFont (display, font_ID)
  Display  *display;
  XID  font_ID;
```

を用います。font_ID にはフォント ID の他にそのフォントを持つ GC の ID を指定することもできます。XLoadQueryFont (display, name) でロードと同時に属性を得ることもできます。XFontStruct は

```
typedef struct {
   XExtData     *ext_data;       /* hook for extension to hang data */
   Font         fid;             /* Font id for this font */
   unsigned     direction;       /* hint about direction the font is painted */
   unsigned     min_char_or_byte2; /* first character */
   unsigned     max_char_or_byte2; /* last character */
   unsigned     min_byte1;       /* first row that exists */
   unsigned     max_byte1;       /* last row that exists */
   Bool         all_chars_exist; /* flag if all characters have non-zero size*/
   unsigned     default_char;    /* char to print for undefined character */
   int          n_properties     /* how many properties there are */
   XFontProp    *properties;     /* pointer to array of additional properties*/
   XCharStruct  min_bounds;      /* minimum bounds over all existing char*/
   XCharStruct  max_bounds;      /* maximum bounds over all existing char*/
   XCharStruct  *per_char;       /* first_char to last_char information */
   int          ascent;          /* log. extent above baseline for spacing */
   int          descent          /* log. descent below baseline for spacing */
} XFontStruct;
```

となっています。properties には，ここにあるのもの以外の細かな属性（NORM_SPACE, SUBSCRIPT_X など）がプロパティの形で入っています。これは XGetFontProperty

(font_struct, atom, value_return) で調べることができます。

X のフォントには 8bit と 16bit の 2 種類のフォーマットのフォントがあります。8bit のフォントは主に英字および数字など（ASCII 文字）を出力するためのものです。16bit のフォントは漢字などアジア系文字を出力するためのものです。8bit のフォントでは XFontStruct の min_byte1 と max_byte1 が 0 です。

XListFonts (display, pattern, maxnames, actual_count_return) (pattern はフォント名のワイルドカード指定) でフォント名の一覧を, XListFontsWithInfo (display, pattern, naxnames, count_return, info_return) でフォント名と属性の一覧を得ることもできます。

フォント名や属性を解放するには XFreeFontInfo (names, free_info, actual_count) や XFreeFontNames (list) を用います。

2.5.2. テキストの出力

X のフォントには 8bit と 16bit の 2 種類がありますが, テキスト出力関数もそれぞれのために別々に用意されています。

最も単純なテキスト出力関数は

```
XDrawString (display, d, gc, x, y, string, length)
  Display  *display;
  Drawable  d;
  GC  gc;
  int  x, y;
  char  *string;
  int  length;

XDrawString16 (display, d, gc, x, y, string, length)
  Display  *display;
  Drawable  d;
  GC  gc;
  int  x, y;
  XChar2b  *string;
  int  length;
```

です。XChar2b は

```
typedef struct {    /* normal 16 bit characters are two bytes */
  unsigned char  byte1;
  unsigned char  byte2;
} XChar2b;
```

となっています。

XDrawImageString (display, d, gc, x, y, string, length) と XDrawImageString16 (display, d, gc, x, y, string, length) はテキストの背景をバックグランドの色で塗りつぶします。

XDrawText (display, d, gc, x, y, items, nitems) と XDrawText16 (display, d, gc, x, y, items, nitems) は item として

```
typedef struct {
  char  *chars;     /* pointer to string */
  int  nchars;      /* number of characters */
  int  delta;       /* delta between strings */
  Font  font;       /* font to print it in, None don't change */
} XTextItem;
```

の配列（もしくは XTextItem16 の配列）を指定します。これによって，フォントを切り替えたり，文字の間隔を変化させた出力を一度に行なうことができます。

文字列を出力した際のディスプレイ上の長さ（ピクセル数）を計算するには

```
int XTextWidth (font_struct, string, count)
  XFontStruct  *font_struct;
  char  *string;
  int  count;
```

と XTextWidth16 を用いることができます。また，高さを含めた情報を計算するには XFontExtents (font_struct, string, nchars, direction_return, font_acent_return, font_decent_return, overall_return) と XFontExtents16 を用います。これらの関数は XFontStruct を用いてクライアントで計算が行なわれますが，直接サーバに問い合わせる XQueryTextExtents (display, font_ID, string, ...) と XQueryTextExtents16 という関数もあります。

2.6. 色の操作

X は複数のプレーンを持ったカラー表示（または濃淡表示）のディスプレイをサポートします。ディスプレイ上の色を指定するために，一つ一つのプレーンを各ビットに割り当てて表わした，ピクセル値（pixel value）を用います。

2.6.1. カラーマップの操作

ウィンドウで用いる複数の色を設定，管理するためにカラーマップというリソースを用います。カラーマップは各色の RGB の値や，その色に対応するピクセル値，その色をアプリケーションから変更できるかどうかなどの情報をまとめて持っています。カラーマップは次の関数で作成します。

```
Colormap XCreateColormap (display, w, visual, alloc)
  Display  *display;
  Window  w;
  Visual  *visual;
  int  alloc;
```

ここで，visual はスクリーンでサポートしている色の使い方を指定するものです（後述）。alloc は read/write のカラーセルを XCreateColormap の呼び出しと同時にアロケートするかどうかの指定です（AllocAll または AllocNone）。window はサーバのどのスクリーンかを示すためのに使われます。

カラーマップをウィンドウに対応付けるには XCreateWindow でウィンドウを作成する時に属性として指定するか，もしくは

```
XSetWindowColormap (display, w, cmap)
  Display  *display;
  Window  w;
  Colormap  cmap;
```

で行ないます。また，カラーマップを削除するには

```
XFreeColormap (display, cmap)
  Display  *display;
  Colormap  cmap;
```

を用います。

Visual は次のように定義されています。

```
typedef struct {
  XExtData  *ext_data;  /* hook for extension to hang data */
  VisualID  visualid;   /* visual id of this visual */
  int  class;           /* class of screen (monochrome, etc.) */
  unsigned long  red_mask, green_mask, blue_mask; /* mask values */
  int  bits_per_rgb;    /* log base 2 of distinct color values */
  int  map_entries;     /* color map entries */
} Visual;
```

visualid　　ビジュアルのリソース ID

class　　ビジュアル・クラス。StaticGray, GrayScale, StaticColor, PseudoColor, TrueColor, DirectColor の 6 種類

red_mask, green_mask, blue_mask
　　RGB それぞれに固定的に割り当てるプレーンのマスク

bits_pre_rgb　　RGB 全体でのプレーン数

map_entries　　使用できる色の数

ビジュアル・クラスは，カラー対応のディスプレイかどうか，ディスプレイの色の変更ができるかどうかおよび，ピクセル値と RGB の対応付けの自由度によって，次のものがあります。ただし，ディスプレイのハードウェアによって使えるものは限られます（マルチプレーンの VAXstation では StaticColor と PseudoColor）。

	color		gray-scale	
	read-only	read/write	read-only	read/write
undecomposed colormap	StaticColor	PseudoColor	StaticGray	GrayScale
decomposed colormap	TrueColor	DirectColor		

サーバのデフォルトのビジュアルは DefaultVisual (display, screen), デフォルトのカラーマップは DefaultColormap (display, screen) で得ることができます。これらは通常，ディスプレイでサポートしている最も自由度の高いものになっています。また，サーバでサポートしているビジュアルを検索するための XGetVisualInfo (...) と XMatchVisualInfo (...) という関数もあります。

2.6.2. カラーセルの操作

ウィンドウで用いる一つ一つの色にそれぞれカラーセルが割り当てられています。カラーセルとピクセル値は一対一に対応しています。カラーセルを操作するには次の構造体を用います。

```
typedef struct {
  unsigned  long pixel;              /* pixel value */
  unsigned short  red, green, blue;  /* rgb values */
  char  flags;                       /* DoRed, DoGreen, DoBlue */
  char  pad;
} XColor;
```

pixel　ピクセル値

red, green, blue
　色の RGB の値。0xffff で最も明るく，0x8000 で中間調，0 で最も暗い指定となる

flag　read/write のカラーセルで RGB を独立に操作するためのマスク。DoRed, DoGreen, DoBlue の 3 つの論理和で指定

read-only のカラーセルを得る（アロケートする）には

```
Status XAllocColor (display, cmap, screen_in_out)
  Display  *display;
  Colormap  cmap;
  XColor  *screen_in_out;
```

screen_in_out　欲しい色の RGB の値を指定。呼び出し後，実際に割り当てられた色の RGB の値とピクセル値が返る

を用います。また，色を名前で指定する XAllocNamedColor (display, cmap, color_name, visual_def_return, exact_def_return) と色の名前から RGB の値を求める XLookupColor (display, cmap, color_name, visual_def_return, exact_def_return) といった関数があります。

PseudoColor のカラーセルを得るには

```
Status XAllocColorCells (display, cmap, contig,
                         plane_masks_return, nplanes,
                         pixels_return, ncolors)
  Display  *display;
  Colormap  cmap;
  Bool  contig;
  unsigned long  plane_masks_return[];
  unsigned int  nplanes;
  unsigned long  pixels_return[];
  unsigned int  ncolors;
```

contig	複数プレーンを連続して取るかどうかを指定
nplanes, ncolors	取りたいプレーンの数と色の数を指定。ncolors * 2^nplanes 個の色が取られる
plane_masks_return, pixels_return	得られたプレーンのプレーン・マスクとピクセル値が返る。大きさはそれぞれ nplanes と ncolors

を用います。また，DirectColor のカラーセルには

```
Status XAllocColorPlanes (display, cmap, contig,
       pixels_return, ncolors,
       nreds, ngreens, nblues,
       rmask_return, gmask_return, bmask_return)
  Display  *display;
  Colormap  cmap;
  Bool  contig;
  unsigned long  pixels_return[];
  int  ncolors;
  int  nreds, ngreens, nblues;
  unsigned long  *rmask_return, *gmask_return, *bmask_return;
```

を用います。これら read/write のカラーセルの色の設定は

```
XStoreColor (display, cmap, color)
  Display  *display;
  Colormap  cmap;
  Color  *color;

XStoreColors (display, cmap, color, ncolors)
  Display  *display;
  Colormap  cmap;
  Color  color[];
  int  ncolors;
```

や，XStoreNamedColor (display, cmap, color, pixel, flags) で行ないます。

カラーセルの解放は XFreeColors (display, cmap, pixels, npixels, planes) で行ないます。

2.6.3. その他のカラーマップ関数

カラーマップはディスプレイでインストールされないと実際の表示に反映されません。サーバはインストールされたカラーマップを required list として，順次反映していきます。カラーマップをインストールするには XInstallColormap (display, cmap)，インストールをやめるには XUninstallColormap (display, cmap) を用います。また，カラーマップの状態が変化すると ColormapNotify イベントが起こります。通常，カラーマップのインストールはウィンドウ・マネージャが行なうことになっています。

フル・カラーをクライアント間で共有するための標準的なカラーマップとしてXStandardColormap というプロパティも用意されています。

2.7. ピックスマップの操作

X のリソースの 1 つであるピックスマップはサーバのオフ・スクリーン上に作られる drawable です。1 プレーンのピックスマップをビットマップ，バックグラウンドなどの塗りつぶしのために用いられるものをタイルと呼びます。ピックスマップはタイル・パターンとして出力したり，カーソルに使用したりすることができます。

ピックスマップの作成，削除は次の関数で行ないます。

```
Pixmap XCreatePixmap (display, d, width, height, depth)
  Display  *display;
  Drawable  d;
  unsigned int  width, height;
  unsigned int  depth;

XFreePixmap (display, pixmap)
  Display  *display;
  Pixmap  pixmap;
```

d　　どのスクリーンかを示すために指定

depth　　pixmap で用いるスクリーンのプレーン数

ビットマップデータ（ビットマップファイル）からビットマップやピックスマップを作ることもできます。これには XCreateBitmapFromData (display, d, data, width,

height) やXCreatePixmapFromBitmapData (display, d, data, width, height, fg, bg, depth) を用います。

ピックスマップをウィンドウに表示するには XCopyArea (display, src, dest, gc, src_x, src_y, width, height, dest_x, dest_y) や XCopyPlane (...) の関数を用います。

2.8. カーソルの操作

X ではウィンドウ毎にマウス・カーソルの形状を決めることができます。

2.8.1. カーソルの作成

まず，カーソルをリソースとしてサーバに登録します。カーソルをビットマップから作る場合，

```
Cursor XCreatePixmapCursor (display, source, mask,
                            foreground, background, x, y)
  Display  *display;
  Pixmap  source, mask;
  XColor  *foreground, *background;
  unsigned int  x, y;
```

source, mask　カーソルのフォアグランドとバックグランドのマスクの形状を指定するビットマップ

x, y　カーソルの指す位置の相対座標（ホット・スポット）

を用います。また，あらかじめ定義されているカーソル（カーソル・フォント）を使用する場合は

```
#include <X11/cursorfont.h>
Cursor XCreateFontCursor (display, shape)
  Display  *display;
  unsigned int  shape;
```

shape　カーソル・フォントの番号。XC_left_ptr, XC_hand, XC_pencil などが cursorfont.h に定義されているある

を用います。文字のフォントからカーソルを作る XCreateGlyphCursor (display, source_font, mask_font, source_char, mask_char, foreground, background) という関数もあります。

カーソルの色を変更するには

```
XRecolorCursor (display, cursor, foreground, bakcground)
  Display  *display;
  Cursor   cursor
  XColor   *forecgound, *background;
```

を用います。カーソルの削除は XFreeCursor (display, cursor) で行ないます。

2.8.2. カーソルとウィンドウの対応付け

ウィンドウにカーソルを対応付けるということは，そのウィンドウでのカーソルの形を決めるということです。これは XCreateWindow の際に指定するか，

```
XDefineCursor (display, w, curosr)
  Display  display;
  Window   w;
  Cursor   cursor;
```

で行ないます。また，XUndefineCursor (display, w) で設定をやめることができます。カーソルの設定されていないウィンドウではそのペアレント・ウィンドウのカーソルが用いられます。

2.9. イベントの処理

Xlib による描画などの操作はリクエストという形でサーバに送られます。送られたリクエストはサーバによって処理され，その結果としてサーバはリプライやイベントを返します。

リプライというのは処理に失敗したとか，クライアントからの問い合せに対する結果を知らせるためのメッセージのことです。イベントはリプライとは違ってクライアントの要求とは直接関係ないタイミングで送られるメッセージです。イベントの発生する要因には，ユーザによるキーボードの操作，マウスの移動，ウィンドウの状態の変化による書き直しなどいくつかの種類があります。

通常リプライは Xlib 関数の戻り値としてアプリケーションに渡されますが，イベントは受け取るための操作を明示的にしてやらないといけません。また，あらかじめウィンドウ毎に受け取るべきイベントを決めておくことができるようになっています。

2.9.1. イベント・タイプ

イベントはイベント・タイプによってそれがどういう種類のものなのかが示されます。イベント・タイプには次のものがあります。

キーボード・イベント（キーボードのキーの押し離し，シフトキーなどの状態をクライアントに知らせる）
KeyPress, KeyRelese

マウスの移動とボタンの操作
ButtonPress, ButtonRelese, MotionNotify

ウィンドウへのポインタの出入り
EnterNotify, LeaveNotify

キーマップの状態の変化
KeymapNotify

expose イベント（ウィンドウの書き直しが必要になった場合，その位置と大きさを知らせる）
Expose, GraphicsExpose, NoExpose

サブ・ウィンドウの構造の制御のためのもの（サブ・ウィンドウの大きさなどの変化を親ウィンドウにあらかじめ知らせる）
CirculateRequest, ConfigureRequest, MapRequest, ResizeRequest

ウィンドウの状態の変化を知らせるもの
CirculateNotify, ConfigureNotify, CreateNotify, DesttroyNotify, GravityNotify, MapNotify, MappingNotify, ReparentNotify, Unmap-Notify, VisibilityNotify

カラーマップの状態の変化を知らせるもの
ColormapNotify

クライアント間のデータのやり取りのためのもの
ClientMessage, PropertyNotify, SelectionClear, SelectionNotify, SelectionRequest

実際，イベントはそれぞれのイベント・タイプに対応した構造体（XKeyPressedEvent やXExposeEvent など）として返されます。また，これらの構造体の一般形として XEvent という共用体も用意されています。XEvent やこれらの構造体の先頭には type というフィールドがあり，ここでイベント・タイプを識別します。他の内容はイベント・タイプによって異なっています。さらに，内容が同じイベントについては共通の構造体（XKeyPressedEvent と XKeyReleasedEvent に対する XKeyEvent など）も用意されています。

2.9.2. イベント・マスク

アプリケーションの処理の対象となるイベントはウィンドウ毎に指定されます。その際，どのイベントを有効にするのかを指定するのに用いるのがイベント・マスクです。主なイベントのイベント・タイプ，イベント・マスク，イベント構造体の対応は次のようになっています。

event type	event mask	structure	generic structure
MotionNotify	ButtonMotionMask Button1MotionMask Button2MotionMask Button3MotionMask Button4MotionMask Button5MotionMask	XPointerMovedEvent	XMotionEvent
ButtonPress ButtonRelease	ButtonPressMask ButtonReleaseMask	XButtonPressedEvent XButtonReleasedEvent	XButtonEvent
KeyPress KeyRelease	KeyPressMask	XKeyPressedEvent XKeyReleasedEvent	XKeyEvent
Expose GraphicsExpose NoExpose	ExposureMask ---- ----	XExposeEvent XGraphicsExposeEvent XNoExposeEvent	
EnterNotify LeaveNotify	EnterWindowMask LeaveWindowMask	XEnterWindowEvent XLeaveWindowEvent	XCrossingEvent

MotionNotify イベントについては，イベント・マスクで特定のボタンだけを指定することができます。また，GaphicsExpose イベントと NoExposeEvent についてはイベント・マスクでなく，GC の graphics_exposures で指定します。

2.9.3. イベント処理関数

ウィンドウで受け付けるイベントを指定するにはイベント・マスクを用います。このイベント・マスクはウィンドウ作成時の XSetWindowAttributes の event_mask で指定するか，次の関数で指定するためのものです。

```
XSelectInput (display, w, event_mask)
  Display  *display;
  Window  w;
  unsigned long  event_mask;
```

ここで指定しなかったマスクに対応するイベントはウィンドウに送られません（一部のイベントを除く）。

サーバから送られてきたイベントは Xlib 内のイベント・キューに蓄えられます。これらを処理するためには，そこから順次イベントを取り出してやらないといけません。これは通常，次の関数で行ないます。

```
XNextEvent (display, event_return)
  Display  *display;
  XEvent  *event_return;
```

XNextEvent はまず，出力バッファにあるリクエストをすべてサーバに送り，それらに対するリプライを受け取ります。そして，イベント・キューの先頭のイベントをアプリケーションに返し，イベント・キューから削除します。イベント・キューが空の場合は次のイベントが来るまで待ちます。イベントをキューから削除せずアプリケーションに返す XPeekEvent (display, event_return) という関数もあります。

また，ウィンドウやイベント・マスクを指定してそれに合うもののみを得る XWindowEvent (display, w, event_mask, event_return) と XMaskEvent (display, event_mask, event_return) という関数もあります。イベントがイベント・キューになくても待ち続けない（False を返す）XCheckWindowEvent (display, w, event_mask, event_return), XCheckMaskEvent (display, event_mask, event_return), XCheckTypedEvent (display, event_type, event_return), XCheckTypedWindowEvent (display, w, event_type, event_return) などの関数もあります。イベント・キューの長さを得るには XPending (display) や XEventQueued (display, more) を使います。

Xlib の出力バッファの内容を明示的にフラッシュするには

```
XFlush (display)
  Display  *display;

XSync (display, discard)
  Display  *display;
  int  discard;
```

discard　イベント・キューに残っているイベントを捨てるかどうかの指定。0 でそのまま，1 で無効にする

を用います。

2.9.4. キー・イベントとポインタ・イベントの処理

KeyPress, KeyRelese, ButtonPress, ButtonRelese イベントは XSelectInput に対してそれぞれ KeyPressMask, KeyReleseMask, ButtonPressMask, ButtonReleseMask を指定することによって有効にします。MotionNotify イベントについては次の 4 種類の指定があります。

Button1MotoinMask ～ Button5MotionMask
　　指定したボタンを押しながらの移動

ButtonMotionMask
　　どれか 1 つのボタンを押しながらの移動

PointerMotionMask
　　ポインタが移動すれば，ボタンの状態に関係なくイベントが発生

PointerMotionHintMask
　　キーやボタンの状態が変化した時に MotionNotify を起こし，それ以外の移動ではイベントを起こさない。マウスの位置は XQueryPoiner や XGetMotionEvents 関数で得る

イベントの構造体はそれぞれ XKeyPressedEvent, XKeyRelesedEvent, XButtonPressedEvent, XButtonRelesedEvent, XPointerMovedEvent です。汎用の XMotionEvent は次のように定義されています。

```
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 reported relative to */
  Window  root;           /* root window that the event occured 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 */
  char  is_hint;          /* detail */
  Bool  same_screen;      /* same screen flag */
} XMotionEvent;
typedef XMotionEvent XPointerMovedEvent;
```

state はマウスのボタンとシフト・キーなどのモディファイア・キーの状態を示しています。これには Button1Mask～Button5Mask, ShiftMask, LockMask, ControlMask, Mod1Mask～Mod5Mask があります。キー・イベントではさらに keycode というフィールドにキーボード上のキーの番号（ASCII コードではない）が入ります。ボタン・イベントでは button フィールドに状態の変化したボタンのマスク（Button1～Button5）が入ります。

ボタン・イベントでは一度ボタンが押されると，そのボタンが離されるまでマウスが自動的にそのウィンドウで占有された状態になります（automatic grabbing）。これは 1 回のボタン操作が複数のウィンドウに影響を及ぼさないようにし，マウスによるドラッギングの操作を可能にします。

2.9.5. exposure イベントの処理

オーバーラップ型のウィンドウ・システムでは他のウィンドウとの位置関係によって，自分のウィンドウの一部が隠された状態になる場合があります。このような領域にグラフィックや文字の出力を行なってもディスプレイに表示されることはありません。また，自分の前に新たなウィンドウが出現した場合にはその領域が消されてしまいます。

X のサーバはこのように表示がなされなかった部分の図形に対して，基本的には特別な処置は施しません。つまり，表示できなかった部分はそのままにして，その時点で表示可能な部分についてのみ，出力処理を行ないます。隠されていた部分が見えるようになった場合，サーバは Expose イベントを起こして，クライアントに再表示を促します（ただし，backing-store をサポートしているサーバでは自動的に書き直しが行なわれます。この場合でも，サーバの資源が不足しているなどの理由で Expose イベントが発生する可能性はあります）。

このように Expose イベントはウィンドウのある領域の再表示が必要になった場合や，ウィンドウがマップされた場合に起こります。これらの領域は互いに共通部分を持たない複数の矩形に分解されてクライアントに渡されます。

Expose イベントは XSelectInput に対して ExposureMask を渡すことによって有効にすることができます。

```
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;
```

構造体 ExposeEvent に書き直すべき領域の x, y, width, height が入ります。また，

count には後に続く Expose イベントの数が入ります。これは複数の矩形に分解された領域を 1 つの矩形として扱いたい場合などに利用できます。

ウィンドウのマップの直後，無条件に描画を行なうと，そのウィンドウの内容が破壊されていないにもかかわらず，もう 1 度余分に描画してしまう事があります。これはマップによって，今まで見えていなかったウィンドウが見えるようになるために，Expose イベントが発生するからです。このような場合にはまず，マップに先立って Expose イベントを有効にしておき，描画は Expose イベントの到着によってのみ行なうというようにすると，余計な描画は行われません。こうすることによって，ウィンドウ・マネージャによるウィンドウのマップと，アプリケーションの描画のタイミングを合わせることもできます。

GraphicsExpose イベントは XCopyArea 関数や XCopyPlane 関数を用いて領域をコピーする際，内容の定まらない部分をサーバがクライアントに問い合わせるために起こります。これは，コピー元の領域が別のウィンドウで隠されていたとか，ウィンドウの外の領域であったような場合です。NoExposeEvent イベントはこれらの関数の結果， GraphicsExpose が必要でなかった場合，そのことを知らせるために起こります。この 2 つのイベントを受け取るためには GC の graphics_exposures を True（デフォルト）にします。構造体 XGraphicsExposeEvent と XNoExposeEvent には drawable, x, y, width, height, count, major_code（CopyArea または CopyPlane がセットされる）のフィールドがあります。

2.10. その他の関数

Xlib にはこの他に次のような関数があります。

ウィンドウのプロパティを操作する関数
　XGetWindowProperty, XChangeProperty, XDeleteProperty など

セレクションのための関数
　XSetSelectionOwner, XGetSelectionOwner, XConvertSelection など

ウィンドウ・マネージャのための関数
　カラーマップをインストールする XInstallColormap, ポインタとキーボードを専有する XGrabPointer, XGrabButton, XGrabKeyboard, XSetInputFocus, キーボードとポインタの設定をする XChangeKeyboardControl, XChangePointerControl, 他のホストのアクセス・コントロールを行う XAddHosts, XRemoveHosts など

ウインドウ・マネージャのための設定をするための関数
　XSetStandardProperties, XStoreName, XSetWMHints, XGetStandardColormap など

キーボード・イベントを処理する関数
　XLookupKeysym, XLookupString, XRefreshKeyboardMapping, XRebindKeysym など

region を操作する関数
XCreateRegion, XPolygonRegion, XSetRegion, XIntersectRegion, XPointInRegion など

ビット・イメージを操作する関数
XCreateImage, XPutPixel, XSubImage, XPutImage, XGetImage など

リソース・マネージャ関数
XrmInitialize, XrmGetFileDatabase, XrmParseCommand, XrmPutLineResource, XrmGetResource など

2.11. Xlib を用いたプログラミング

Xlib は X の最も低レベルなライブラリです。このため，ウィンドウの作成，GC の設定，イベントの処理など，細かな処理をすべてアプリケーションで行なう必要があります。

Xlib を用いたプログラムでは <X11/Xlib.h> をインクルードします。コンパイル時には cc コマンドのオプションとして -lX11 を指定します。プログラムを実行する際には通常，DISPLAY という環境変数にサーバの指定 (host_name:0) を行ないます。

例として Xlib を用いた "Hello world!" プログラムを示します。このプログラムはマウスの MB1 を押すことによって終了します。また，ウィンドウの内容が壊れた場合の書き直し処理も行なっています。

```
/*
 * hw_xlib.c
 *      sample implementation of hw on Xlib
 *      to compile this, enter;
 *              % cc hw_xlib.c -o hw_xlib -lX11
 *
 * Tat001       18-Jul-1989     for x11pg
 */

#include <X11/Xlib.h>
#include <X11/cursorfont.h>

static char     *hello_world = {"Hello world!"};

main ()
{
        Display *display;
        Window  hw;
        GC      gc;
        Cursor  cursor;
        XSetWindowAttributes    attr;
        int     attr_mask;

        /* make connection to server */
        display = XOpenDisplay ("");

        /* create gc for window */
        gc = DefaultGC (display, 0);

        /* create cursor for window */
        cursor = XCreateFontCursor (display, XC_hand1);

        /* set attribute for window */
        attr.background_pixel = XWhitePixel (display, 0);
        attr.event_mask       = ButtonPressMask | ButtonReleaseMask |
                                ExposureMask;
        attr.cursor           = cursor;
        attr_mask = CWBackPixel | CWEventMask | CWCursor;

        /* create and map widnow */
        hw = XCreateWindow (display, RootWindow(display,0),
                        200, 200, 200, 100, 1,  /* x, y, w, h, bw */
                        CopyFromParent, InputOutput,    /* depth, class, */
                        DefaultVisual(display,0),       /* visual, */
                        attr_mask, &attr);
        XMapWindow (display, hw);

        /* event processing loop */
        for (;;) {
```

```
                XEvent  event;

                /* get next event */
                XNextEvent (display, &event);

                switch (event.type) {
                case Expose:
                        {
                                Window  window =
                                 ((XExposeEvent*)&event)->window;
                                XDrawString (display, window, gc,
                                        50, 20,
                                        hello_world, strlen(hello_world));
                                break;
                        }

                case ButtonRelease:
                        exit ();
                }
        }

}

/* end of hw_xlib.c */
```