Extended Window Manager Hints

X Desktop Group

Final version 1.3

Table of Contents

Introduction

Version

This is Final version 1.3 of the Extended Window Manager Hints (EWMH) spec, updated May 13, 2005. The canonical home for this document is http://www.freedesktop.org, which also contains directions for reporting bugs or contributing to future versions.

What is this spec?

This spec defines interactions between window managers, applications, and the utilities that form part of a desktop environment. It builds on the Inter-Client Communication Conventions Manual [ICCCM], which defines window manager interactions at a lower level. The ICCCM does not provide ways to implement many features that modern desktop users expect. The GNOME and KDE desktop projects originally developed their own extensions to the ICCCM to support these features; this spec replaces those custom extensions with a standardized set of ICCCM additions that any desktop environment can adopt.

Language used in this specification

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119.

The key words "Window Manager" refer to a window manager which is adopting this specification. "Pager" refers to desktop utility applications, including pagers and taskbars. "Application" refers to other clients. "Clients" refers to Pagers and Applications ie. all X clients, except for the Window Manager.

Prerequisites for adoption of this specification

Window Managers and Clients which aim to fulfill this specification MUST adhere to the ICCCM on which this specification builds. If this specification explicitly modifies the ICCCM Window Managers and Clients MUST fulfill these modifications.

Non-ICCCM features

There is a number of window management features or behaviors which are not specified in the ICCCM, but are commonly met in modern window managers and desktop environments.

Additional States

The ICCCM allows window managers to implement additional window states, which will appear to clients as substates of NormalState and IconicState. Two commonly met examples are Maximized and Shaded. A window manager may implement these as proper substates of NormalState and IconicState, or it may treat them as independent flags, allowing e.g. a maximized window to be iconified and to re-appear as maximized upon de-iconification.

Maximization

Maximization is a very old feature of window managers. There was even a ZoomedState in early ICCCM drafts. Maximizing a window should give it as much of the screen area as possible (this may not be the full screen area, but only a smaller 'workarea', since the window manager may have reserved certain areas for other windows). A window manager is expected to remember the geometry of a maximized window and restore it upon de-maximization. Modern window managers typically allow separate horizontal and vertical maximization.

With the introduction of the Xinerama extension in X11 R6.4, maximization has become more involved. Xinerama allows a screen to span multiple monitors in a freely configurable geometry. In such a setting, maximizing a window would ideally not grow it to fill the whole screen, but only the monitor it is shown on. There are of course borderline cases for windows crossing monitor boundaries, and 'real' maximization to the full screen may sometimes be useful.

Shading

Some desktop environments offer shading (also known as rollup) as an alternative to iconification. A shaded window typically shows only the titlebar, the client window is hidden, thus shading is not useful for windows which are not decorated with a titlebar.

Modality

The WM_TRANSIENT_FOR hint of the ICCCM allows clients to specify that a toplevel window may be closed before the client finishes. A typical example of a transient window is a dialog. Some dialogs can be open for a long time, while the user continues to work in the main window. Other dialogs have to be closed before the user can continue to work in the main window. This property is called modality. While clients can implement modal windows in an ICCCM compliant way using the globally active input model, some window managers offer support for handling modality.

Large Desktops

The window manager may offer to arrange the managed windows on a desktop that is larger than the root window. The screen functions as a viewport on this large desktop. Different policies regarding the positioning of the viewport on the desktop can be implemented: The window manager may only allow the viewport position to change in increments of the screen size (paging) or it may allow arbitrary positions (scrolling).

To fulfill the ICCCM principle that clients should behave the same regardless whether a window manager is running or not, window managers which implement large desktops must interpret all client-provided geometries with respect to the current viewport.

Implementation note

There are two options for implementing a large desktop: The first is to keep the

managed windows (or, if reparenting, their frames) as children of the root window. Moving the viewport is achieved by moving all managed windows in the opposite direction.

The second alternative is to reparent all managed windows to a dedicated large window (somewhat inappropriately called a 'virtual root'). Moving the viewport is then achieved by moving the virtual root in the opposite direction.

Both alternatives are completely ICCCM compliant, although the second one may be somewhat problematic for clients trying to figure out the window manager decorations around their toplevel windows and for clients trying to draw background images on the root window.

Sticky windows

A window manager which implements a large desktop typically offers a way for the user to make certain windows 'stick to the glass', i.e. these windows will stay at the same position on the screen when the viewport is moved.

Virtual Desktops

Most X servers have only a single screen. The window manager may virtualize this resource and offer multiple so-called 'virtual desktops', of which only one can be shown on the screen at a time. There is some variation among the features of virtual desktop implementations. There may be a fixed number of desktops, or new ones may be created dynamically. The size of the desktops may be fixed or variable. If the desktops are larger than the root window, their viewports (see the section called “Large Desktops”) may be independent or forced to be at the same position.

A window manager which implements virtual desktops generally offers a way for the user to move clients between desktops. Clients may be allowed to occupy more than one desktop simultaneously.

Implementation note

There are at least two options for implementing virtual desktops. The first is to use multiple virtual roots (see the section called “Implementation note”) and change the current desktop by manipulating the stacking order of the virtual roots. This is completely ICCCM compliant, but has the issues outlined in the section called “Implementation note”

The second option is to keep all managed windows as children of the root window and unmap the frames of those which are not on the current desktop. Unmapped windows should be placed in IconicState, according to the ICCCM. Windows which are actually iconified or minimized should have the _NET_WM_STATE_HIDDEN property set, to communicate to pagers that the window should not be represented as "onscreen."

Pagers

A pager offers a different UI for window management tasks. It shows a miniature view of the desktop(s) representing managed windows by small rectangles and allows the

user to initiate various window manager actions by manipulating these representations. Typically offered actions are activation (see the section called "Activation"), moving, restacking, iconification, maximization and closing. On a large desktop, the pager may offer a way to move the viewport. On virtual desktops, the pager may offer ways to move windows between desktops and to change the current desktop.

Taskbars

A taskbar offers another UI for window management tasks. It typically represents client windows as a list of buttons labelled with the window titles and possibly icons. Pressing a button initiates a window manager action on the represented window, typical actions being activation and iconification. In environments with a taskbar, icons are often considered inappropriate, since the iconified windows are already represented in the taskbar.

Activation

In the X world, activating a window means to give it the input focus. This may not be possible if the window is unmapped, because it is on a different desktop. Thus, activating a window may involve additional steps like moving it to the current desktop (or changing to the desktop the window is on), deiconifying it or raising it.

Animated iconification

Some window managers display some form of animation when (de-)iconifying a window. This may be a line drawing connecting the corners of the window with the corners of the icon or the window may be opaquely moved and resized on some trajectory joining the window location and the icon location.

Window-in-window MDI

Window-in-window MDI is a multiple document interface known from MS Windows platforms. Programs employing it have a single top-level window which contains a workspace which contains the subwindows for the open documents. These subwindows are decorated with window manager frames and can be manipulated within their parent window just like ordinary top-level windows on the root window.

Layered stacking order

Some window managers keep the toplevel windows not in a single linear stack, but subdivide the stack into several layers. There is a lot of variation among the features of layered stacking order implementations. The number of layers may or may not be fixed. The layer of a toplevel window may be explicit and directly modifiable or derived from other properties of the window, e.g. the *type* of the window. The stacking order may or may not be strict, i.e. not allow the user to raise or lower windows beyond their layer.

Scope of this spec

This spec tries to address the following issues:

- Allow clients to influence their initial state with respect to maximization, shading, stickiness, desktop, stacking order.
- Improve the window managers ability to vary window decorations and maintain the stacking order by allowing clients to hint the window manager about the type of their windows.
- Enable pagers and taskbars to be implemented as separate clients and allow them to work with any compliant window manager.

This spec doesn't cover any of the following:

- Other IPC mechanisms like ICE or Corba.
- Window manager configuration.
- Window manager documentation.
- Clients appearing on a proper subset of desktops.
- Window-in-window MDI.

The window manager is supposed to be in charge of window management policy, so that there is consistent behavior on the user's screen no matter who wrote the clients.

The spec offers a lot of external control about window manager actions. This is intended mainly to allow pagers, taskbars and similar window manager UIs to be implemented as separate clients. "Ordinary" clients shouldn't use these except maybe in response to a direct user request (i.e. setting a config option to start maximized or specifying a -desk n command line argument).

Root Window Properties (and Related Messages)

Whenever this spec speaks about “sending a message to the root window”, it is understood that the client is supposed to create a ClientMessage event with the specified contents and send it by using a SendEvent request with the following arguments:

```
destination     root
propagate       False
event-mask      (SubstructureNotify|SubstructureRedirect)
event           the specified ClientMessage
```

_NET_SUPPORTED

```
_NET_SUPPORTED, ATOM[]/32
```

This property MUST be set by the Window Manager to indicate which hints it supports. For example: considering _NET_WM_STATE both this atom and all supported states e.g. _NET_WM_STATE_MODAL, _NET_WM_STATE_STICKY, would be listed. This assumes that backwards incompatible changes will not be made to the

hints (without being renamed).

_NET_CLIENT_LIST

```
_NET_CLIENT_LIST, WINDOW[]/32
_NET_CLIENT_LIST_STACKING, WINDOW[]/32
```

These arrays contain all X Windows managed by the Window Manager. _NET_CLIENT_LIST has initial mapping order, starting with the oldest window. _NET_CLIENT_LIST_STACKING has bottom-to-top stacking order. These properties SHOULD be set and updated by the Window Manager.

_NET_NUMBER_OF_DESKTOPS

```
_NET_NUMBER_OF_DESKTOPS, CARDINAL/32
```

This property SHOULD be set and updated by the Window Manager to indicate the number of virtual desktops.

A Pager can request a change in the number of desktops by sending a _NET_NUMBER_OF_DESKTOPS message to the root window:

```
_NET_NUMBER_OF_DESKTOPS
  message_type = _NET_NUMBER_OF_DESKTOPS
  format = 32
  data.l[0] = new_number_of_desktops
  other data.l[] elements = 0
```

The Window Manager is free to honor or reject this request. If the request is honored _NET_NUMBER_OF_DESKTOPS MUST be set to the new number of desktops, _NET_VIRTUAL_ROOTS MUST be set to store the new number of desktop virtual root window IDs and _NET_DESKTOP_VIEWPORT and _NET_WORKAREA must also be changed accordingly. The _NET_DESKTOP_NAMES property MAY remain unchanged.

If the number of desktops is shrinking and _NET_CURRENT_DESKTOP is out of the new range of available desktops, then this MUST be set to the last available desktop from the new set. Clients that are still present on desktops that are out of the new range MUST be moved to the very last desktop from the new set. For these _NET_WM_DESKTOP MUST be updated.

_NET_DESKTOP_GEOMETRY

```
_NET_DESKTOP_GEOMETRY width, height, CARDINAL[2]/32
```

Array of two cardinals that defines the common size of all desktops (this is equal to the screen size if the Window Manager doesn't support large desktops, otherwise it's equal to the virtual size of the desktop). This property SHOULD be set by the Window Manager.

A Pager can request a change in the desktop geometry by sending a _NET_DESKTOP_GEOMETRY client message to the root window:

```
_NET_DESKTOP_GEOMETRY
  message_type = _NET_DESKTOP_GEOMETRY
```

```
  format = 32
  data.l[0] = new_width
  data.l[1] = new_height
  other data.l[] elements = 0
```

The Window Manager MAY choose to ignore this message, in which case _NET_DESKTOP_GEOMETRY property will remain unchanged.

_NET_DESKTOP_VIEWPORT

```
_NET_DESKTOP_VIEWPORT x, y, CARDINAL[][2]/32
```

Array of pairs of cardinals that define the top left corner of each desktop's viewport. For Window Managers that don't support large desktops, this MUST always be set to (0,0).

A Pager can request to change the viewport for the current desktop by sending a _NET_DESKTOP_VIEWPORT client message to the root window:

```
_NET_DESKTOP_VIEWPORT
  message_type = _NET_DESKTOP_VIEWPORT
  format = 32
  data.l[0] = new_vx
  data.l[1] = new_vy
  other data.l[] elements = 0
```

The Window Manager MAY choose to ignore this message, in which case _NET_DESKTOP_VIEWPORT property will remain unchanged.

_NET_CURRENT_DESKTOP

```
_NET_CURRENT_DESKTOP desktop, CARDINAL/32
```

The index of the current desktop. This is always an integer between 0 and _NET_NUMBER_OF_DESKTOPS - 1. This MUST be set and updated by the Window Manager. If a Pager wants to switch to another virtual desktop, it MUST send a _NET_CURRENT_DESKTOP client message to the root window:

```
_NET_CURRENT_DESKTOP
  message_type = _NET_CURRENT_DESKTOP
  format = 32
  data.l[0] = new_index
  data.l[1] = timestamp
  other data.l[] elements = 0
```

Note that the timestamp may be 0 for clients using an older version of this spec, in which case the timestamp field should be ignored.

_NET_DESKTOP_NAMES

```
_NET_DESKTOP_NAMES, UTF8_STRING[]
```

The names of all virtual desktops. This is a list of NULL-terminated strings in UTF-8 encoding [UTF8]. This property MAY be changed by a Pager or the Window Manager at any time.

Note: The number of names could be different from _NET_NUMBER_OF_DESKTOPS. If it is less than _NET_NUMBER_OF_DESKTOPS, then the desktops with high numbers are unnamed. If it is larger than _NET_NUMBER_OF_DESKTOPS, then the excess names outside of the _NET_NUMBER_OF_DESKTOPS are considered to be reserved in case the number of desktops is increased.

Rationale: The name is not a necessary attribute of a virtual desktop. Thus the availability or unavailability of names has no impact on virtual desktop functionality. Since names are set by users and users are likely to preset names for a fixed number of desktops, it doesn't make sense to shrink or grow this list when the number of available desktops changes.

_NET_ACTIVE_WINDOW

```
_NET_ACTIVE_WINDOW, WINDOW/32
```

The window ID of the currently active window or None if no window has the focus. This is a read-only property set by the Window Manager. If a Client wants to activate another window, it MUST send a _NET_ACTIVE_WINDOW client message to the root window:

```
_NET_ACTIVE_WINDOW
  window  = window to activate
  message_type = _NET_ACTIVE_WINDOW
  format = 32
  data.l[0] = source indication
  data.l[1] = timestamp
  data.l[2] = requestor's currently active window, 0 if none
  other data.l[] elements = 0
```

Source indication should be 1 when the request comes from an application, and 2 when it comes from a pager. Clients using older version of this spec use 0 as source indication, see the section called “Source indication in requests” for details. The timestamp is Client's last user activity timestamp (see _NET_WM_USER_TIME) at the time of the request, and the currently active window is the Client's active toplevel window, if any (the Window Manager may be e.g. more likely to obey the request if it will mean transferring focus from one active window to another).

Depending on the information provided with the message, the Window Manager may decide to refuse the request (either completely ignore it, or e.g. use _NET_WM_STATE_DEMANDS_ATTENTION).

_NET_WORKAREA

```
_NET_WORKAREA, x, y, width, height CARDINAL[][4]/32
```

This property MUST be set by the Window Manager upon calculating the work area for each desktop. Contains a geometry for each desktop. These geometries are specified relative to the viewport on each desktop and specify an area that is completely contained within the viewport. Work area SHOULD be used by desktop applications to place desktop icons appropriately.

The Window Manager SHOULD calculate this space by taking the current page minus space occupied by dock and panel windows, as indicated by the _NET_WM_STRUT or _NET_WM_STRUT_PARTIAL properties set on client windows.

_NET_SUPPORTING_WM_CHECK

```
_NET_SUPPORTING_WM_CHECK, WINDOW/32
```

The Window Manager MUST set this property on the root window to be the ID of a child window created by himself, to indicate that a compliant window manager is active. The child window MUST also have the _NET_SUPPORTING_WM_CHECK property set to the ID of the child window. The child window MUST also have the _NET_WM_NAME property set to the name of the Window Manager.

Rationale: The child window is used to distinguish an active Window Manager from a stale _NET_SUPPORTING_WM_CHECK property that happens to point to another window. If the _NET_SUPPORTING_WM_CHECK window on the client window is missing or not properly set, clients SHOULD assume that no conforming Window Manager is present.

_NET_VIRTUAL_ROOTS

```
_NET_VIRTUAL_ROOTS, WINDOW[]/32
```

To implement virtual desktops, some Window Managers reparent client windows to a child of the root window. Window Managers using this technique MUST set this property to a list of IDs for windows that are acting as virtual root windows. This property allows background setting programs to work with virtual roots and allows clients to figure out the window manager frame windows of their windows.

_NET_DESKTOP_LAYOUT

```
_NET_DESKTOP_LAYOUT, orientation, columns, rows, starting_corner CARDINAL[4]/32

  #define _NET_WM_ORIENTATION_HORZ 0
  #define _NET_WM_ORIENTATION_VERT 1

  #define _NET_WM_TOPLEFT     0
  #define _NET_WM_TOPRIGHT    1
  #define _NET_WM_BOTTOMRIGHT 2
  #define _NET_WM_BOTTOMLEFT  3
```

This property is set by a Pager, not by the Window Manager. When setting this property, the Pager must own a manager selection (as defined in the ICCCM 2.8). The manager selection is called _NET_DESKTOP_LAYOUT_Sn where n is the screen number. The purpose of this property is to allow the Window Manager to know the desktop layout displayed by the Pager.

_NET_DESKTOP_LAYOUT describes the layout of virtual desktops relative to each other. More specifically, it describes the layout used by the owner of the manager selection. The Window Manager may use this layout information or may choose to ignore it. The property contains four values: the Pager orientation, the number of desktops in the X direction, the number in the Y direction, and the starting corner of

the layout, i.e. the corner containing the first desktop.

Note: In order to inter-operate with Pagers implementing an earlier draft of this document, Window Managers should accept a _NET_DESKTOP_LAYOUT property of length 3 and use _NET_WM_TOPLEFT as the starting corner in this case.

The virtual desktops are arranged in a rectangle with `rows` rows and `columns` columns. If `rows` times `columns` does not match the total number of desktops as specified by _NET_NUMBER_OF_DESKTOPS, the highest-numbered workspaces are assumed to be nonexistent. Either `rows` or `columns` (but not both) may be specified as 0 in which case its actual value will be derived from _NET_NUMBER_OF_DESKTOPS.

When the orientation is _NET_WM_ORIENTATION_HORZ the desktops are laid out in rows, with the first desktop in the specified starting corner. So a layout with four columns and three rows starting in the _NET_WM_TOPLEFT corner looks like this:

```
+--+--+--+--+
| 0| 1| 2| 3|
+--+--+--+--+
| 4| 5| 6| 7|
+--+--+--+--+
| 8| 9|10|11|
+--+--+--+--+
```

With starting_corner _NET_WM_BOTTOMRIGHT, it looks like this:

```
+--+--+--+--+
|11|10| 9| 8|
+--+--+--+--+
| 7| 6| 5| 4|
+--+--+--+--+
| 3| 2| 1| 0|
+--+--+--+--+
```

When the orientation is _NET_WM_ORIENTATION_VERT the layout with four columns and three rows starting in the _NET_WM_TOPLEFT corner looks like:

```
+--+--+--+--+
| 0| 3| 6| 9|
+--+--+--+--+
| 1| 4| 7|10|
+--+--+--+--+
| 2| 5| 8|11|
+--+--+--+--+
```

With starting_corner _NET_WM_TOPRIGHT, it looks like:

```
+--+--+--+--+
| 9| 6| 3| 0|
+--+--+--+--+
|10| 7| 4| 1|
+--+--+--+--+
|11| 8| 5| 2|
+--+--+--+--+
```

The numbers here are the desktop numbers, as for _NET_CURRENT_DESKTOP.

_NET_SHOWING_DESKTOP

```
_NET_SHOWING_DESKTOP desktop, CARDINAL/32
```

Some Window Managers have a "showing the desktop" mode in which windows are hidden, and the desktop background is displayed and focused. If a Window Manager supports the _NET_SHOWING_DESKTOP hint, it MUST set it to a value of 1 when the Window Manager is in "showing the desktop" mode, and a value of zero if the Window Manager is not in this mode.

If a Pager wants to enter or leave the mode, it MUST send a _NET_SHOWING_DESKTOP client message to the root window requesting the change:

```
_NET_SHOWING_DESKTOP
  message_type = _NET_SHOWING_DESKTOP
  format = 32
  data.l[0] = boolean 0 or 1
  other data.l[] elements = 0
```

The Window Manager may choose to ignore this client message.

Other Root Window Messages

_NET_CLOSE_WINDOW

```
_NET_CLOSE_WINDOW
```

Pagers wanting to close a window MUST send a _NET_CLOSE_WINDOW client message request to the root window:

```
_NET_CLOSE_WINDOW
  window = window to close
  message_type = _NET_CLOSE_WINDOW
  format = 32
  data.l[0] = timestamp
  data.l[1] = source indication
  other data.l[] elements = 0
```

The Window Manager MUST then attempt to close the window specified. See the section called “Source indication in requests” for details on the source indication.

Rationale: A Window Manager might be more clever than the usual method (send WM_DELETE message if the protocol is selected, XKillClient otherwise). It might introduce a timeout, for example. Instead of duplicating the code, the Window Manager can easily do the job.

_NET_MOVERESIZE_WINDOW

```
_NET_MOVERESIZE_WINDOW
  window = window to be moved or resized
  message_type = _NET_MOVERESIZE_WINDOW
  format = 32
  data.l[0] = gravity and flags
  data.l[1] = x
  data.l[2] = y
  data.l[3] = width
  data.l[4] = height
```

The low byte of data.l[0] contains the gravity to use; it may contain any value allowed for the WM_SIZE_HINTS.win_gravity property: NorthWest (1), North (2), NorthEast (3), West (4), Center (5), East (6), SouthWest (7), South (8), SouthEast (9) and Static (10). A gravity of 0 indicates that the Window Manager should use the gravity specified in WM_SIZE_HINTS.win_gravity. The bits 8 to 11 indicate the presence of x, y, width and height. The bits 12 to 15 indicate the source (see the section called “Source indication in requests”), so 0001 indicates the application and 0010 indicates a Pager or a Taskbar. The remaining bits should be set to zero.

Pagers wanting to move or resize a window may send a _NET_MOVERESIZE_WINDOW client message request to the root window instead of using a ConfigureRequest.

Window Managers should treat a _NET_MOVERESIZE_WINDOW message exactly like a ConfigureRequest (in particular, adhering to the ICCCM rules about synthetic ConfigureNotify events), except that they should use the gravity specified in the message.

Rationale: Using a _NET_MOVERESIZE_WINDOW message with StaticGravity allows Pagers to exactly position and resize a window including its decorations without knowing the size of the decorations.

_NET_WM_MOVERESIZE

```
_NET_WM_MOVERESIZE
  window = window to be moved or resized
  message_type = _NET_WM_MOVERESIZE
  format = 32
  data.l[0] = x_root
  data.l[1] = y_root
  data.l[2] = direction
  data.l[3] = button
  data.l[4] = source indication
```

This message allows Clients to initiate window movement or resizing. They can define their own move and size "grips", whilst letting the Window Manager control the actual operation. This means that all moves/resizes can happen in a consistent manner as defined by the Window Manager. See the section called “Source indication in requests” for details on the source indication.

When sending this message in response to a button press event, button SHOULD indicate the button which was pressed, x_root and y_root MUST indicate the position of the button press with respect to the root window and direction MUST indicate whether this is a move or resize event, and if it is a resize event, which edges of the window the size grip applies to. When sending this message in response to a key event, the direction MUST indicate whether this this is a move or resize event and the other fields are unused.

```
#define _NET_WM_MOVERESIZE_SIZE_TOPLEFT      0
#define _NET_WM_MOVERESIZE_SIZE_TOP          1
#define _NET_WM_MOVERESIZE_SIZE_TOPRIGHT     2
#define _NET_WM_MOVERESIZE_SIZE_RIGHT        3
#define _NET_WM_MOVERESIZE_SIZE_BOTTOMRIGHT  4
#define _NET_WM_MOVERESIZE_SIZE_BOTTOM       5
#define _NET_WM_MOVERESIZE_SIZE_BOTTOMLEFT   6
```

```
#define _NET_WM_MOVERESIZE_SIZE_LEFT         7
#define _NET_WM_MOVERESIZE_MOVE              8   /* movement only */
#define _NET_WM_MOVERESIZE_SIZE_KEYBOARD     9   /* size via keyboard */
#define _NET_WM_MOVERESIZE_MOVE_KEYBOARD    10   /* move via keyboard */
```

The Client MUST release all grabs prior to sending such message.

The Window Manager can use the button field to determine the events on which it terminates the operation initiated by the _NET_WM_MOVERESIZE message. Since there is a race condition between a client sending the _NET_WM_MOVERESIZE message and the user releasing the button, Window Managers are advised to offer some other means to terminate the operation, e.g. by pressing the ESC key.

_NET_RESTACK_WINDOW

```
_NET_RESTACK_WINDOW
```

Pagers wanting to restack a window SHOULD send a _NET_RESTACK_WINDOW client message request to the root window:

```
_NET_RESTACK_WINDOW
  window = window to restack
  message_type = _NET_RESTACK_WINDOW
  format = 32
  data.l[0] = source indication
  data.l[1] = sibling window
  data.l[2] = detail
  other data.l[] elements = 0
```

This request is similar to ConfigureRequest with CWSibling and CWStackMode flags. It should be used only by pagers, applications can use normal ConfigureRequests. The source indication field should be therefore set to 2, see the section called “Source indication in requests” for details.

Rationale: A Window Manager may put restrictions on configure requests from applications, for example it may under some conditions refuse to raise a window. This request makes it clear it comes from a pager or similar tool, and therefore the Window Manager should always obey it.

_NET_REQUEST_FRAME_EXTENTS

```
_NET_REQUEST_FRAME_EXTENTS
  window = window for which to set _NET_FRAME_EXTENTS
  message_type = _NET_REQUEST_FRAME_EXTENTS
```

A Client whose window has not yet been mapped can request of the Window Manager an estimate of the frame extents it will be given upon mapping. To retrieve such an estimate, the Client MUST send a _NET_REQUEST_FRAME_EXTENTS message to the root window. The Window Manager MUST respond by estimating the prospective frame extents and setting the window's _NET_FRAME_EXTENTS property accordingly. The Client MUST handle the resulting _NET_FRAME_EXTENTS PropertyNotify event. So that the Window Manager has a good basis for estimation, the Client MUST set any window properties it intends to set *before* sending this message. The Client MUST be able to cope with imperfect estimates.

Rationale: A client cannot calculate the dimensions of its window's frame before the window is mapped, but some toolkits need this information. Asking the window manager for an estimate of the extents is a workable solution. The estimate may depend on the current theme, font sizes or other window properties. The client can track changes to the frame's dimensions by listening for _NET_FRAME_EXTENTS PropertyNotify events.

Application Window Properties

_NET_WM_NAME

```
_NET_WM_NAME, UTF8_STRING
```

The Client SHOULD set this to the title of the window in UTF-8 encoding. If set, the Window Manager should use this in preference to WM_NAME.

_NET_WM_VISIBLE_NAME

```
_NET_WM_VISIBLE_NAME, UTF8_STRING
```

If the Window Manager displays a window name other than _NET_WM_NAME the Window Manager MUST set this to the title displayed in UTF-8 encoding.

Rationale: This property is for Window Managers that display a title different from the _NET_WM_NAME or WM_NAME of the window (i.e. xterm <1>, xterm <2>, ... is shown, but _NET_WM_NAME / WM_NAME is still xterm for each window) thereby allowing Pagers to display the same title as the Window Manager.

_NET_WM_ICON_NAME

```
_NET_WM_ICON_NAME, UTF8_STRING
```

The Client SHOULD set this to the title of the icon for this window in UTF-8 encoding. If set, the Window Manager should use this in preference to WM_ICON_NAME.

_NET_WM_VISIBLE_ICON_NAME

```
_NET_WM_VISIBLE_ICON_NAME, UTF8_STRING
```

If the Window Manager displays an icon name other than _NET_WM_ICON_NAME the Window Manager MUST set this to the title displayed in UTF-8 encoding.

_NET_WM_DESKTOP

```
_NET_WM_DESKTOP desktop, CARDINAL/32
```

Cardinal to determine the desktop the window is in (or wants to be) starting with 0 for the first desktop. A Client MAY choose not to set this property, in which case the Window Manager SHOULD place it as it wishes. 0xFFFFFFFF indicates that the window SHOULD appear on all desktops.

The Window Manager should honor _NET_WM_DESKTOP whenever a withdrawn window requests to be mapped.

The Window Manager should remove the property whenever a window is withdrawn but it should leave the property in place when it is shutting down, e.g. in response to losing ownership of the WM_Sn manager selection.

Rationale: Removing the property upon window withdrawal helps legacy applications which want to reuse withdrawn windows. Not removing the property upon shutdown allows the next Window Manager to restore windows to their previous desktops.

A Client can request a change of desktop for a non-withdrawn window by sending a _NET_WM_DESKTOP client message to the root window:

```
_NET_WM_DESKTOP
  window  = the respective client window
  message_type = _NET_WM_DESKTOP
  format = 32
  data.l[0] = new_desktop
  data.l[1] = source indication
  other data.l[] elements = 0
```

See the section called “Source indication in requests” for details on the source indication. The Window Manager MUST keep this property updated on all windows.

_NET_WM_WINDOW_TYPE

```
_NET_WM_WINDOW_TYPE, ATOM[]/32
```

This SHOULD be set by the Client before mapping to a list of atoms indicating the functional type of the window. This property SHOULD be used by the window manager in determining the decoration, stacking position and other behavior of the window. The Client SHOULD specify window types in order of preference (the first being most preferable) but MUST include at least one of the basic window type atoms from the list below. This is to allow for extension of the list of types whilst providing default behavior for Window Managers that do not recognize the extensions.

Rationale: This hint is intended to replace the MOTIF hints. One of the objections to the MOTIF hints is that they are a purely visual description of the window decoration. By describing the function of the window, the Window Manager can apply consistent decoration and behavior to windows of the same type. Possible examples of behavior include keeping dock/panels on top or allowing pinnable menus / toolbars to only be hidden when another window has focus (NextStep style).

```
_NET_WM_WINDOW_TYPE_DESKTOP, ATOM
_NET_WM_WINDOW_TYPE_DOCK, ATOM
_NET_WM_WINDOW_TYPE_TOOLBAR, ATOM
_NET_WM_WINDOW_TYPE_MENU, ATOM
_NET_WM_WINDOW_TYPE_UTILITY, ATOM
_NET_WM_WINDOW_TYPE_SPLASH, ATOM
_NET_WM_WINDOW_TYPE_DIALOG, ATOM
_NET_WM_WINDOW_TYPE_NORMAL, ATOM
```

_NET_WM_WINDOW_TYPE_DESKTOP indicates a desktop feature. This can include a single window containing desktop icons with the same dimensions as the screen,

allowing the desktop environment to have full control of the desktop, without the need for proxying root window clicks.

_NET_WM_WINDOW_TYPE_DOCK indicates a dock or panel feature. Typically a Window Manager would keep such windows on top of all other windows.

_NET_WM_WINDOW_TYPE_TOOLBAR and _NET_WM_WINDOW_TYPE_MENU indicate toolbar and pinnable menu windows, respectively (i.e. toolbars and menus "torn off" from the main application). Windows of this type may set the WM_TRANSIENT_FOR hint indicating the main application window.

_NET_WM_WINDOW_TYPE_UTILITY indicates a small persistent utility window, such as a palette or toolbox. It is distinct from type TOOLBAR because it does not correspond to a toolbar torn off from the main application. It's distinct from type DIALOG because it isn't a transient dialog, the user will probably keep it open while they're working. Windows of this type may set the WM_TRANSIENT_FOR hint indicating the main application window.

_NET_WM_WINDOW_TYPE_SPLASH indicates that the window is a splash screen displayed as an application is starting up.

_NET_WM_WINDOW_TYPE_DIALOG indicates that this is a dialog window. If _NET_WM_WINDOW_TYPE is not set, then windows with WM_TRANSIENT_FOR set MUST be taken as this type.

_NET_WM_WINDOW_TYPE_NORMAL indicates that this is a normal, top-level window. Windows with neither _NET_WM_WINDOW_TYPE nor WM_TRANSIENT_FOR set MUST be taken as this type.

_NET_WM_STATE

```
_NET_WM_STATE, ATOM[]
```

A list of hints describing the window state. Atoms present in the list MUST be considered set, atoms not present in the list MUST be considered not set. The Window Manager SHOULD honor _NET_WM_STATE whenever a withdrawn window requests to be mapped. A Client wishing to change the state of a window MUST send a _NET_WM_STATE client message to the root window (see below). The Window Manager MUST keep this property updated to reflect the current state of the window.

The Window Manager should remove the property whenever a window is withdrawn, but it should leave the property in place when it is shutting down, e.g. in response to losing ownership of the WM_Sn manager selection.

Rationale: Removing the property upon window withdrawal helps legacy applications which want to reuse withdrawn windows. Not removing the property upon shutdown allows the next Window Manager to restore windows to their previous state.

Possible atoms are:

```
_NET_WM_STATE_MODAL, ATOM
_NET_WM_STATE_STICKY, ATOM
_NET_WM_STATE_MAXIMIZED_VERT, ATOM
```

```
_NET_WM_STATE_MAXIMIZED_HORZ, ATOM
_NET_WM_STATE_SHADED, ATOM
_NET_WM_STATE_SKIP_TASKBAR, ATOM
_NET_WM_STATE_SKIP_PAGER, ATOM
_NET_WM_STATE_HIDDEN, ATOM
_NET_WM_STATE_FULLSCREEN, ATOM
_NET_WM_STATE_ABOVE, ATOM
_NET_WM_STATE_BELOW, ATOM
_NET_WM_STATE_DEMANDS_ATTENTION, ATOM
```

An implementation MAY add new atoms to this list. Implementations without extensions MUST ignore any unknown atoms, effectively removing them from the list. These extension atoms MUST NOT start with the prefix _NET.

_NET_WM_STATE_MODAL indicates that this is a modal dialog box. If the WM_TRANSIENT_FOR hint is set to another toplevel window, the dialog is modal for that window; if WM_TRANSIENT_FOR is not set or set to the root window the dialog is modal for its window group.

_NET_WM_STATE_STICKY indicates that the Window Manager SHOULD keep the window's position fixed on the screen, even when the virtual desktop scrolls.

_NET_WM_STATE_MAXIMIZED_{VERT,HORZ} indicates that the window is {vertically,horizontally} maximized.

_NET_WM_STATE_SHADED indicates that the window is shaded.

_NET_WM_STATE_SKIP_TASKBAR indicates that the window should not be included on a taskbar. This hint should be requested by the application, i.e. it indicates that the window by nature is never in the taskbar. Applications should not set this hint if _NET_WM_WINDOW_TYPE already conveys the exact nature of the window.

_NET_WM_STATE_SKIP_PAGER indicates that the window should not be included on a Pager. This hint should be requested by the application, i.e. it indicates that the window by nature is never in the Pager. Applications should not set this hint if _NET_WM_WINDOW_TYPE already conveys the exact nature of the window.

_NET_WM_STATE_HIDDEN should be set by the Window Manager to indicate that a window would not be visible on the screen if its desktop/viewport were active and its coordinates were within the screen bounds. The canonical example is that minimized windows should be in the _NET_WM_STATE_HIDDEN state. Pagers and similar applications should use _NET_WM_STATE_HIDDEN instead of WM_STATE to decide whether to display a window in miniature representations of the windows on a desktop.

Implementation note: if an Application asks to toggle _NET_WM_STATE_HIDDEN the Window Manager should probably just ignore the request, since _NET_WM_STATE_HIDDEN is a function of some other aspect of the window such as minimization, rather than an independent state.

_NET_WM_STATE_FULLSCREEN indicates that the window should fill the entire screen and have no window decorations. Additionally the Window Manager is responsible for restoring the original geometry after a switch from fullscreen back to normal window. For example, a presentation program would use this hint.

_NET_WM_STATE_ABOVE indicates that the window should be on top of most windows (see the section called “Stacking order” for details).

_NET_WM_STATE_BELOW indicates that the window should be below most windows (see the section called “Stacking order” for details).

_NET_WM_STATE_ABOVE and _NET_WM_STATE_BELOW are mainly meant for user preferences and should not be used by applications e.g. for drawing attention to their dialogs (the Urgency hint should be used in that case, see the section called “Urgency”).'

_NET_WM_STATE_DEMANDS_ATTENTION indicates that some action in or with the window happened. For example, it may be set by the Window Manager if the window requested activation but the Window Manager refused it, or the application may set it if it finished some work. This state may be set by both the Client and the Window Manager. It should be unset by the Window Manager when it decides the window got the required attention (usually, that it got activated).

To change the state of a mapped window, a Client MUST send a _NET_WM_STATE client message to the root window:

```
window  = the respective client window
message_type = _NET_WM_STATE
format = 32
data.l[0] = the action, as listed below
data.l[1] = first property to alter
data.l[2] = second property to alter
data.l[3] = source indication
other data.l[] elements = 0
```

This message allows two properties to be changed simultaneously, specifically to allow both horizontal and vertical maximization to be altered together. l[2] MUST be set to zero if only one property is to be changed. See the section called “Source indication in requests” for details on the source indication. l[0], the action, MUST be one of:

```
_NET_WM_STATE_REMOVE        0    /* remove/unset property */
_NET_WM_STATE_ADD           1    /* add/set property */
_NET_WM_STATE_TOGGLE        2    /* toggle property  */
```

See also the implementation notes on urgency and fixed size windows.

_NET_WM_ALLOWED_ACTIONS

```
_NET_WM_ALLOWED_ACTIONS, ATOM[]
```

A list of atoms indicating user operations that the Window Manager supports for this window. Atoms present in the list indicate allowed actions, atoms not present in the list indicate actions that are not supported for this window. The Window Manager MUST keep this property updated to reflect the actions which are currently "active" or "sensitive" for a window. Taskbars, Pagers, and other tools use _NET_WM_ALLOWED_ACTIONS to decide which actions should be made available to the user.

Possible atoms are:

```
_NET_WM_ACTION_MOVE, ATOM
_NET_WM_ACTION_RESIZE, ATOM
_NET_WM_ACTION_MINIMIZE, ATOM
_NET_WM_ACTION_SHADE, ATOM
_NET_WM_ACTION_STICK, ATOM
_NET_WM_ACTION_MAXIMIZE_HORZ, ATOM
_NET_WM_ACTION_MAXIMIZE_VERT, ATOM
_NET_WM_ACTION_FULLSCREEN, ATOM
_NET_WM_ACTION_CHANGE_DESKTOP, ATOM
_NET_WM_ACTION_CLOSE, ATOM
```

An implementation MAY add new atoms to this list. Implementations without extensions MUST ignore any unknown atoms, effectively removing them from the list. These extension atoms MUST NOT start with the prefix _NET.

Note that the actions listed here are those that the *Window Manager* will honor for this window. The operations must still be requested through the normal mechanisms outlined in this specification. For example, _NET_WM_ACTION_CLOSE does not mean that clients can send a WM_DELETE_WINDOW message to this window; it means that clients can use a _NET_CLOSE_WINDOW message to ask the Window Manager to do so.

Window Managers SHOULD ignore the value of _NET_WM_ALLOWED_ACTIONS when they initially manage a window. This value may be left over from a previous Window Manager with different policies.

_NET_WM_ACTION_MOVE indicates that the window may be moved around the screen.

_NET_WM_ACTION_RESIZE indicates that the window may be resized. (Implementation note: Window Managers can identify a non-resizable window because its minimum and maximum size in WM_NORMAL_HINTS will be the same.)

_NET_WM_ACTION_MINIMIZE indicates that the window may be iconified.

_NET_WM_ACTION_SHADE indicates that the window may be shaded.

_NET_WM_ACTION_STICK indicates that the window may have its sticky state toggled (as for _NET_WM_STATE_STICKY). Note that this state has to do with viewports, not desktops.

_NET_WM_ACTION_MAXIMIZE_HORZ indicates that the window may be maximized horizontally.

_NET_WM_ACTION_MAXIMIZE_VERT indicates that the window may be maximized vertically.

_NET_WM_ACTION_FULLSCREEN indicates that the window may be brought to fullscreen state.

_NET_WM_ACTION_CHANGE_DESKTOP indicates that the window may be moved between desktops.

_NET_WM_ACTION_CLOSE indicates that the window may be closed (i.e. a WM_DELETE_WINDOW message may be sent).

_NET_WM_STRUT

```
_NET_WM_STRUT, left, right, top, bottom, CARDINAL[4]/32
```

This property is equivalent to a _NET_WM_STRUT_PARTIAL property where all start values are 0 and all end values are the height or width of the logical screen. _NET_WM_STRUT_PARTIAL was introduced later than _NET_WM_STRUT, however, so clients MAY set this property in addition to _NET_WM_STRUT_PARTIAL to ensure backward compatibility with Window Managers supporting older versions of the Specification.

_NET_WM_STRUT_PARTIAL

```
_NET_WM_STRUT_PARTIAL, left, right, top, bottom, left_start_y, left_end_y,
right_start_y, right_end_y, top_start_x, top_end_x, bottom_start_x,
bottom_end_x,CARDINAL[12]/32
```

This property MUST be set by the Client if the window is to reserve space at the edge of the screen. The property contains 4 cardinals specifying the width of the reserved area at each border of the screen, and an additional 8 cardinals specifying the beginning and end corresponding to each of the four struts. The order of the values is left, right, top, bottom, left_start_y, left_end_y, right_start_y, right_end_y, top_start_x, top_end_x, bottom_start_x, bottom_end_x. All coordinates are root window coordinates. The client MAY change this property at any time, therefore the Window Manager MUST watch for property notify events if the Window Manager uses this property to assign special semantics to the window.

If both this property and the _NET_WM_STRUT property are set, the Window Manager MUST ignore the _NET_WM_STRUT property values and use instead the values for _NET_WM_STRUT_PARTIAL. This will ensure that Clients can safely set both properties without giving up the improved semantics of the new property.

The purpose of struts is to reserve space at the borders of the desktop. This is very useful for a docking area, a taskbar or a panel, for instance. The Window Manager should take this reserved area into account when constraining window positions - maximized windows, for example, should not cover that area.

The start and end values associated with each strut allow areas to be reserved which do not span the entire width or height of the screen. Struts MUST be specified in root window coordinates, that is, they are *not* relative to the edges of any view port or Xinerama monitor.

For example, for a panel-style Client appearing at the bottom of the screen, 50 pixels tall, and occupying the space from 200-600 pixels from the left of the screen edge would set a bottom strut of 50, and set bottom_start_x to 200 and bottom_end_x to 600. Another example is a panel on a screen using the Xinerama extension. Assume that the set up uses two monitors, one running at 1280x1024 and the other to the right running at 1024x768, with the top edge of the two physical displays aligned. If the panel wants to fill the entire bottom edge of the smaller display with a panel 50 pixels tall, it should set a bottom strut of 306, with bottom_start_x of 1280, and bottom_end_x of 2303. Note that the strut is relative to the screen edge, and not the edge of the xinerama monitor.

Rationale: A simple "do not cover" hint is not enough for dealing with e.g. auto-hide panels.

Notes: An auto-hide panel SHOULD set the strut to be its minimum, hidden size. A "corner" panel that does not extend for the full length of a screen border SHOULD only set one strut.

_NET_WM_ICON_GEOMETRY

```
_NET_WM_ICON_GEOMETRY, x, y, width, height, CARDINAL[4]/32
```

This optional property MAY be set by stand alone tools like a taskbar or an iconbox. It specifies the geometry of a possible icon in case the window is iconified.

Rationale: This makes it possible for a Window Manager to display a nice animation like morphing the window into its icon.

_NET_WM_ICON

```
_NET_WM_ICON CARDINAL[][2+n]/32
```

This is an array of possible icons for the client. This specification does not stipulate what size these icons should be, but individual desktop environments or toolkits may do so. The Window Manager MAY scale any of these icons to an appropriate size.

This is an array of 32bit packed CARDINAL ARGB with high byte being A, low byte being B. The first two cardinals are width, height. Data is in rows, left to right and top to bottom.

_NET_WM_PID

```
_NET_WM_PID CARDINAL/32
```

If set, this property MUST contain the process ID of the client owning this window. This MAY be used by the Window Manager to kill windows which do not respond to the _NET_WM_PING protocol.

If _NET_WM_PID is set, the ICCCM-specified property WM_CLIENT_MACHINE MUST also be set. While the ICCCM only requests that WM_CLIENT_MACHINE is set " to a string that forms the name of the machine running the client as seen from the machine running the server" conformance to this specification requires that WM_CLIENT_MACHINE be set to the fully-qualified domain name of the client's host.

See also the implementation notes on killing hung processes.

_NET_WM_HANDLED_ICONS

```
_NET_WM_HANDLED_ICONS
```

This property can be set by a Pager on one of its own toplevel windows to indicate that the Window Manager need not provide icons for iconified windows, for example if it is a taskbar and provides buttons for iconified windows.

_NET_WM_USER_TIME

```
_NET_WM_USER_TIME CARDINAL/32
```

This property contains the XServer time at which last user activity in this window took place.

Clients should set this property on every new toplevel window, before mapping the window, to the timestamp of the user interaction that caused the window to appear. A client that only deals with core events, might, for example, use the timestamp of the last KeyPress or ButtonPress event. ButtonRelease and KeyRelease events should not generally be considered to be user interaction, because an application may receive KeyRelease events from global keybindings, and generally release events may have later timestamp than actions that were triggered by the matching press events. Clients can obtain the timestamp that caused its first window to appear from the DESKTOP_STARTUP_ID environment variable, if the app was launched with startup notification. If the client does not know the timestamp of the user interaction that caused the first window to appear (e.g. because it was not launched with startup notification), then it should not set the property for that window. The special value of zero on a newly mapped window can be used to request that the window not be initially focused when it is mapped.

If the client has the active window, it should also update this property on the window whenever there's user activity.

Rationale: This property allows a Window Manager to alter the focus, stacking, and/or placement behavior of windows when they are mapped depending on whether the new window was created by a user action or is a "pop-up" window activated by a timer or some other event.

_NET_FRAME_EXTENTS

```
_NET_FRAME_EXTENTS, left, right, top, bottom, CARDINAL[4]/32
```

The Window Manager MUST set _NET_FRAME_EXTENTS to the extents of the window's frame. left, right, top and bottom are widths of the respective borders added by the Window Manager.

Window Manager Protocols

_NET_WM_PING

This protocol allows the Window Manager to determine if the Client is still processing X events. This can be used by the Window Manager to determine if a window which fails to close after being sent WM_DELETE_WINDOW has stopped responding or has stalled for some other reason, such as waiting for user confirmation. A Client SHOULD indicate that it is willing to participate in this protocol by listing _NET_WM_PING in the WM_PROTOCOLS property of the client window.

A Window Manager can use this protocol at any time by sending a client message as follows:

```
type = ClientMessage
window = the respective client window
message_type = WM_PROTOCOLS
format = 32
data.l[0] = _NET_WM_PING
data.l[1] = timestamp
data.l[2] = the respective client window
other data.l[] elements = 0
```

A participating Client receiving this message MUST send it back to the root window immediately, by setting window = root, and calling XSendEvent with the same event mask like all other root window messages in this specification use. The Client MUST NOT alter any field in the event other than the window. This includes all 5 longs in the data.l[5] array. The Window Manager can uniquely identify the ping by the timestamp and the data.l[2] field if necessary. Note that some older clients may not preserve data.l[2] through data.l[4].

The Window Manager MAY kill the Client (using _NET_WM_PID) if it fails to respond to this protocol within a reasonable time.

See also the implementation notes on killing hung processes.

_NET_WM_SYNC_REQUEST

This protocol uses the XSync extension (see the protocol specification and the library documentation) to let client and window manager synchronize the repaint of the window manager frame and the client window. A client indicates that it is willing to participate in the protocol by listing _NET_WM_SYNC_REQUEST in the WM_PROTOCOLS property of the client window and storing the XID of an XSync counter in the property _NET_WM_SYNC_REQUEST_COUNTER. The initial value of this counter is not defined by this specification.

A window manager uses this protocol by preceding a ConfigureNotify event sent to a client by a client message as follows:

```
type = ClientMessage
window = the respective client window
message_type = WM_PROTOCOLS
format = 32
data.l[0] = _NET_WM_SYNC_REQUEST
data.l[1] = timestamp
data.l[2] = low 32 bits of the update request number
data.l[3] = high 32 bits of the update request number
other data.l[] elements = 0
```

After receiving one or more such message/ConfigureNotify pairs, and having handled all repainting associated with the ConfigureNotify events, the client MUST set the _NET_WM_SYNC_REQUEST_COUNTER to the 64 bit number indicated by the data.l[2] and data.l[3] fields of the last client message received.

By using either the Alarm or the Await mechanisms of the XSync extension, the window manager can know when the client has finished handling the ConfigureNotify events. The window manager SHOULD not resize the window faster than the client can keep up.

The update request number in the client message is determined by the window manager subject to the restriction that it MUST NOT be 0. The number is generally intended to be incremented by one for each message sent. Since the initial value of the XSync counter is not defined by this specification, the window manager MAY set the value of the XSync counter at any time, and MUST do so when it first manages a new window.

Implementation notes

Desktop/workspace model

This spec assumes a desktop model that consists of one or more completely independent desktops which may or may not be larger than the screen area. When a desktop is larger than the screen it is left to the Window Manager if it will implement scrolling or paging.

File Manager desktop

This spec suggests implementing the file manager desktop by mapping a desktop-sized window (no shape) to all desktops, with _NET_WM_WINDOW_TYPE_DESKTOP. This makes the desktop focusable and greatly simplifies implementation of the file manager. It is also faster than managing lots of small shaped windows. The file manager draws the background on this window. There should be a root property with a window handle for use in applications that want to draw the background (xearth).

Implementing enhanced support for application transient windows

If the WM_TRANSIENT_FOR property is set to None or Root window, the window should be treated as a transient for all other windows in the same group. It has been noted that this is a slight ICCCM violation, but as this behavior is pretty standard for many toolkits and window managers, and is extremely unlikely to break anything, it seems reasonable to document it as standard.

Urgency

Windows expecting immediate user action should indicate this using the urgency bit in the WM_HINTS.flags property, as defined in the ICCCM.

Fixed size windows

Windows can indicate that they are non-resizable by setting minheight = maxheight and minwidth = maxwidth in the ICCCM WM_NORMAL_HINTS property. The Window Manager MAY decorate such windows differently.

Pagers and Taskbars

This specification attempts to make reasonable provisions for window manager independent pagers and taskbars. Window Managers that require / desire additional

functionality beyond what can be achieved using the mechanisms set out in this specification may choose to implement their own pagers, which communicate with the Window Manager using further, window manager specific hints, or some other means.

Pagers should decide whether to show a miniature version of a window using the following guidelines:

- If either _NET_WM_STATE_SKIP_PAGER or _NET_WM_STATE_HIDDEN are set on a window, then the pager should not show that window.
- The pager may choose not to display windows with certain semantic types; this spec has no recommendations, but common practice is to avoid displaying _NET_WM_WINDOW_TYPE_DOCK for example.
- If the _NET_WM_STATE_SKIP_PAGER and _NET_WM_STATE_HIDDEN hints are not present, and the Window Manager claims to support _NET_WM_STATE_HIDDEN, then the window should be shown if it's in either NormalState or IconicState.
- For Window Managers that do not support _NET_WM_STATE_HIDDEN, the pager should not show windows in IconicState. These Window Managers are probably using an older version of this specification.

Window Geometry

Window manager implementors should refer to the ICCCM for definitive specifications of how to handle MapRequest and ConfigureRequest events. However, since these aspects of the ICCCM are easily misread, this document offers the following clarifications:

- Window Managers MUST honor the win_gravity field of WM_NORMAL_HINTS for both MapRequest *and* ConfigureRequest events (ICCCM Version 2.0, §4.1.2.3 and §4.1.5)
- When generating synthetic ConfigureNotify events, the position given MUST be the top-left corner of the client window in relation to the origin of the root window (i.e., ignoring win_gravity) (ICCCM Version 2.0, §4.2.3)
- Window Managers maintain a reference point for each client window and place the window relative to this reference point depending on the window's win_gravity as follows:

win_gravity:	placed at the reference point
StaticGravity	the left top corner of the client window
NorthWestGravity	the left top corner of the frame window
NorthGravity	the center of the frame window's top side
NorthEastGravity	the right top corner of the frame window
EastGravity	the center of the frame window's right side
SouthEastGravity	the right bottom corner of the frame window

SouthGravity	the center of the frame window's bottom side
SouthWestGravity	the left bottom corner of the frame window
WestGravity	the center of the frame window's left side
CenterGravity	the center of the frame window

- Applications are free to change their win_gravity setting at any time.

  If an Application changes its win_gravity then the Window Manager should adjust the reference point, so that the client window will not move as the result. For example if the Application's win_gravity was NorthWestGravity and reference point was at the top-left corner of the frame window, then after change of win_gravity to SouthEastGravity the reference point should be adjusted to point to the lower-right corner of the frame.

  ### Note

  Changing the win_gravity for a single configure request and back afterwards is unlikely to work as intended, due to a race condition. The window manager sees a property notify for WM_NORMAL_HINTS, followed by the configure request, followed by another property notify for WM_NORMAL_HINTS. By the time the window manager gets around to request the changed WM_NORMAL_HINTS in response to the first property notify, the server may have already processed the second property change.

  If the window manager supports it, applications should use _NET_MOVERESIZE_WINDOW with a specified gravity to avoid this problem.

- If the Application requests a new position (x, y) (and possibly also a new size), the Window Manager calculates a new reference point (ref_x, ref_y), based on the client window's (possibly new) size (width, height), border width (bw) and win_gravity as explained in the table below.

  The Window Manager will use the new reference point until the next request for a new position.

win_gravity:	ref_x:	ref_y:
StaticGravity	x	y
NorthWestGravity	x-bw	y-bw
NorthGravity	x+(width/2)	y-bw
NorthEastGravity	x+width+bw	y-bw
EastGravity	x+width+bw	y+(height/2)
SouthEastGravity	x+width+bw	y+height+bw
SouthGravity	x+(width/2)	y+height+bw
SouthWestGravity	x-bw	y+height+bw

WestGravity	x-bw	y+(height/2)
CenterGravity	x+(width/2)	y+(height/2)

- If an Application requests just a new size, its reference point does not move. So for example if client window has win_gravity SouthEastGravity and is resized, the bottom right corner of its frame will not move but instead the top left corner will be adjusted by the difference in size.
- When calculating the reference point at the time of initial placement, the Window Manager should take the initial window's size into consideration, as if it was the frame for this window.

Window-in-Window MDI

The authors of this specification acknowledge that there is no standard method to allow the Window Manager to manage windows that are part of a Window-in-Window MDI application. Application authors are advised to use some other form of MDI, or to propose a mechanism to be included in a future revision of this specification.

Killing Hung Processes

If processes fail to respond to the _NET_WM_PING protocol _NET_WM_PID may be used in combination with the ICCCM specified WM_CLIENT_MACHINE to attempt to kill a process.

WM_CLIENT_MACHINE is usually set by calling XSetWMProperties(). The hostname for the current host can be be retrieved using gethostname(), when gethostname() is not available on the platform implementors may use the value of the nodename field of struct utsname as returned by uname(). Note also that the value of WM_CLIENT_MACHINE is not guaranteed to be a fully fully-qualified domain name of the host. An example of how to retrieve the hostname:

```
int net_get_hostname (char *buf, size_t maxlen)
{
#ifdef HAVE_GETHOSTNAME
        if (buf == NULL) return 0;

        gethostname (buf, maxlen);
        buf [maxlen - 1] = '\0';

        return strlen(buf);
#else
        struct utsname name;
        size_t len;

        if (buf == NULL) return 0;

        uname (&name);
        len = strlen (name.nodename);

        if (len >= maxlen) len = maxlen - 1;
        strncpy (buf, name.nodename, len);
        buf[len] = '\0';

        return len;
```

```
#endif
}
```

Stacking order

To obtain good interoperability between different Desktop Environments, the following layered stacking order is recommended, from the bottom:

- windows of type _NET_WM_TYPE_DESKTOP
- windows having state _NET_WM_STATE_BELOW
- windows not belonging in any other layer
- windows of type _NET_WM_TYPE_DOCK (unless they have state _NET_WM_TYPE_BELOW) and windows having state _NET_WM_STATE_ABOVE
- focused windows having state _NET_WM_STATE_FULLSCREEN

Windows that are transient for another window should be kept above this window.

The window manager may choose to put some windows in different stacking positions, for example to allow the user to bring currently a active window to the top and return it back when the window looses focus.

Source indication in requests

Some requests from Clients include type of the Client, for example the _NET_ACTIVE_WINDOW message. Currently the types can be 1 for normal applications, and 2 for pagers and other Clients that represent direct user actions (the Window Manager may decide to treat requests from applications differently than requests that are result of direct user actions). Clients that support only older version of this spec will have 0 as their source indication, thus not specifying their source at all. This also may mean that some of the fields in the message comply only with the older specification version.

References

[UTF8]

F. Yergeau,"UTF-8, a transformation format of ISO 10646", RFC 2279

[ICCCM]

David Rosenthal and Stuart W. Marks, "Inter-Client Communication Conventions Manual (Version 2.0)", X Consortium Standard, X Version 11, Release 6.3

Copyright

Contributors

Sasha Vasko

Bradley T. Hughes

Dominik Vogt

Havoc Pennington

Jeff Raven

Jim Gettys

John Harper

Julian Adams

Matthias Ettrich

Micheal Rogers

Nathan Clemons

Tim Janik

Tomi Ollila

Sam Lantinga

The Rasterman

Paul Warren

Owen Taylor

Marko Macek

Greg Badros

Matthias Clasen

David Rosenthal

Lubos Lunak

Rob Adams

Thomas Fitzsimmons

Olivier Chapuis

Søren Sandmann

Change history

Changes since 1.2

- Added source indication to _NET_CLOSE_WINDOW, _NET_WM_MOVERESIZE, _NET_MOVERESIZE_WINDOW, _NET_WM_DESKTOP and _NET_WM_STATE message.
- Added _NET_WM_SYNC_REQUEST to allow synchronized repaint of application window and window manager frame during opaque resize.
- Added _NET_REQUEST_FRAME_EXTENTS and _NET_FRAME_EXTENTS to allow a client to retrieve its window's frame extents.
- Added new state _NET_WM_STATE_DEMANDS_ATTENTION.
- Added timestamp, source indication and requestor's active window fields to the _NET_ACTIVE_WINDOW message.
- Added _NET_RESTACK_WINDOW message.
- Added new property _NET_WM_STRUT_PARTIAL to allow partial-width struts.
- Rewrote the implementation notes on "Window Movement", retitled it to "Window Geometry".
- Rewrote the implementation notes on "Urgency", making it clear that the hint is not just about dialogs.
- Fixed the specification of the X and Y members of _NET_DESKTOP_LAYOUT and renamed them to columns and row for clarity.
- Change the description of _NET_WM_STATE_MODAL to no longer require apps to break the ICCCM for group-modal windows, but still support the WM_TRANSIENT_FOR=root dialect.

- Specified that (yet) unused fields in client messages must be set to 0.
- _NET_WM_PING message now has the client window identified in data.l[2] field.
- Added _NET_WM_USER_TIME to detect user activity in windows.
- Explicitly specify that the window manager should restore original geometry when _NET_WM_STATE_FULLSCREEN is reset.

Changes since 1.1

- Changed WM_CLIENT_NAME(STRING) from suggested to required for _NET_WM_PID.
- Specification and sample code for the content of WM_CLIENT_NAME(STRING).
- Added _NET_WM_WINDOW_TYPE_SPLASH, _NET_WM_WINDOW_TYPE_UTILITY.
- Added _NET_WM_STATE_FULLSCREEN.
- Added _NET_WM_ALLOWED_ACTIONS.
- Added _NET_WM_STATE_HIDDEN and clarified purpose of _NET_WM_STATE_SKIP_PAGER and _NET_WM_STATE_SKIP_TASKBAR. Changed section on virtual desktop implementation to suggest ICCCM compliance regarding IconicState, using _NET_WM_STATE_HIDDEN to avoid confusion. Added implementation note for pagers on when to display a window.
- Added button field and new directions for keyboard-initiated actions to the _NET_WM_MOVERESIZE message.
- Added advice on removing _NET_WM_STATE and _NET_WM_DESKTOP when a window is withdrawn.
- Added _NET_DESKTOP_LAYOUT to allow a Pager to specify inter-desktop geometry.
- Added _NET_SHOWING_DESKTOP.
- Added _NET_WM_STATE_ABOVE and _NET_WM_STATE_BELOW and a recommended layered stacking order.
- Added _NET_MOVERESIZE_WINDOW.
- Improve markup of citations.
- Explain _NET_DESKTOP_GEOMETRY and _NET_WM_HANDLED_ICONS in more detail and improve the explanation of WM_CLIENT_MACHINE in the section called “Killing Hung Processes”.
- Add Lubos Lunak to the list of contributors.

Changes since 1.0

- Fix doctype, add author info, update data.
- Change specification description wording to be more inclusive, and to reflect the joint nature of the specification.
- Fix miscellaneous typographical, grammar and spelling errors.
- Clarified _NET_SUPPORTED to include ALL atoms, not just the property names.
- Various corrections to use of MUST and SHOULD.
- Fix problem in _NET_WM_ICON where 'bytes' should have been 'cardinals'
- Replaced ISO-8559-1 characters with entities.

Changes since 1.0pre5

- Change history moved to end.
- UTF-8 Reference updated.
- Window Gravity information updated.
- Copyright Added.
- Minor typo corrections.

Changes since 1.0pre4

- Clarified the interpretation of client-provided geometries on large desktops.
- Added more explanation for _NET_DESKTOP_NAMES.
- Added _NET_WM_ICON_NAME and _NET_WM_VISIBLE_ICON_NAME.
- Tried to improve the wording of _NET_WM_STRUT explanation.
- Changed _NET_WORKAREA to an array of viewport-relative geometries.
- Updated list of “dependent” properties for _NET_NUMBER_OF_DESKTOPS to include _NET_WORKAREA and _NET_DESKTOP_VIEWPORT.
- Tidied formatting of all client messages.

Changes since 1.0pre3

- Added information about common non-ICCCM features.
- Added explanation of sending messages to the root window.
- Removed XA_ prefix from type names.

- Clarified that “mapping order” refers to inital mapping and specify the directions of both orders.
- Clarified that desktops have a common size specified by _NET_DESKTOP_GEOMETRY.
- Rewrote explanation of _NET_DESKTOP_VIEWPORT.
- Tidied formatting of _NET_CURRENT_DESKTOP.
- Replaced “window handle” by “window ID”.
- Tidied formatting of _NET_WORKAREA.
- Rewrote the motivation for _NET_VIRTUAL_ROOTS.
- Added advice on Pointer grabs to _NET_WM_MOVERESIZE.
- Fixed typos in _NET_WM_STATE.
- Added _NET_WM_STATE_SKIP_PAGER.
- Tidied formatting of _NET_WM_STRUT.
- Tidied formatting of _NET_WM_ICON_GEOMETRY.

Changes since 1.0pre2

- _NET_SET_NUMBER_OF_DESKTOPS -> _NET_NUMBER_OF_DESKTOPS for consistency.
- _NET_WM_VISIBLE_NAME_STRING -> _NET_WM_VISIBLE_NAME for consistency.
- _NET_WM_STATE: added explanation of permitted extensions. Added explanation of being set / not set.
- Spellchecked, corrected various typos.
- UTF8 -> UTF-8 for consistency.
- added references to the ICCCM an UTF-8 (incomplete).
- added data and event formats where missing.
- clarified _NET_SUPPORTING_WM_CHECK.
- fixed formatting of _NET_CLOSE_WINDOW message.

Changes since 1.0pre1

- Removed implementation note concerning Gnome's (potential) file manager behavior.

- The Window Movement section of the implementation notes has been revised.

Changes since 1.9f

- Revised revision number for first accepted release 1.9XX -> 1.0preXX.
- Prerequisites for adoption of this specification added.
- Tidied formatting of _NET_CURRENT_DESKTOP for consistency.
- Tidied formatting of _NET_ACTIVE_WINDOW for consistency. Removed doubled text.
- Tidied formatting of _NET_WM_DESKTOP for consistency.
- Killing Hung Processes implementation note added. _NET_WM_PID and _NET_WM_PING now link to this.
- Clarified x_root and y_root meaning for _NET_WM_MOVERESIZE.
- Added contributor list.

Changes since 1.9e

- Added _NET_WM_VISIBLE_NAME_STRING
- Removed ambiguity from _NET_NUMBER_OF_DESKTOPS and _NET_DESKTOP_NAMES in combination.
- Set _NET_WM_MOVERESIZE format to 32 for consistency.
- Removed _NET_PROPERTIES.
- Removed comment from _NET_WM_MOVERESIZE.

Changes since 1.9d

- Added _NET_VIRTUAL_ROOTS
- Added note about ICCCM compliant window moves.
- Added _NET_WM_HANDLED_ICONS
- Added _NET_SUPPORTING_WM_CHECK
- Removed degrees of activation

Changes since 1.9c

- Removed packaging of hints into 2 X properties. Jim Gettys points out that the performance gains of fewer round trips can be better achieved using Xlib routines.

- Clarified that _NET_DESKTOP_VIEWPORT is in pixels
- _NET_DESKTOP_VIEWPORT is now an array, one for each desktop, to allow for different active viewports on different desktops
- _NET_WM_STRUT now only applies on desktops on which the client is visible
- Introduced RFC 2119 language, and attempted to clarify the roles of the Window Manager, Pagers and Applications
- Added _NET_WM_NAME
- _NET_DESKTOP_NAMES now in UTF8
- Desktops now start from 0
- Added _NET_WM_PID
- Added _NET_WM_PING protocol
- Added _NET_WM_STATE_SKIP_TASKBAR

Changes since 1.9b

- Removed _NET_NUMBER_OF_DESKTOPS client message, as it overlaps unnecessarily with _NET_{INSERT/DELETE}_DESKTOP.
- Replaced _NET_WM_LAYER and _NET_WM_HINTS with _NET_WM_WINDOW_TYPE functional hint.
- Changed _NET_WM_STATE to a list of atoms, for extensibility.
- Expanded description of _NET_WORKAREA and _NET_WM_STRUT.
- Removed _NET_WM_SIZEMOVE_NOTIFY protocol.
- Added degrees of activation to _NET_ACTIVE_WINDOW client message
- Added _NET_WM_ICON
- My comments are in [[ ]]. Comments from Marko's draft are in [[MM:]]