

Extended Window Manager Hints

X Desktop Group(<http://www.freedesktop.org>)

10 March 2001

Table of Contents

1. Introduction.....	1
2. Non-ICCCM features	2
3. Root Window Properties (+Related Messages).....	6
4. Other Root Window Messages.....	9
5. Application Window Properties	10
6. Window Manager Protocols	15
7. Implementation notes	15
8. References.....	18
9. Copyright.....	18
10. Contributors	19
11. Change history	20

1. Introduction

1.1. Version

This is version 1.1 of the Extended Window Manager Hints (EWMH) spec, updated 10 March 2001.

1.2. 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 ICCCM [2], which defines WM (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.

1.3. 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 + Applications ie. all X clients, except for the Window Manager.

1.4. Prerequisites for adoption of this specification

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

2. Non-ICCCM features

There is a number of window management features or behaviours which are not specified in the ICCCM, but are commonly met in modern Window Managers and Desktop Environments.

2.1. 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.

2.1.1. 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.

2.1.2. 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.

2.2. Modality

The Window Manager `_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.

2.3. 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 to change the viewport position 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.

2.3.1. 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.

2.4. 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.

2.5. 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 Section 2.3) 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.

2.5.1. Implementation note

There are at least two options for implementing virtual desktops. The first is to use multiple virtual roots (see Section 2.3.1) 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 Section 2.3.1

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. This puts the clients in an undefined ICCCM state, since they are unviewable, but not iconic. In practice, this seems to cause no problems and the ICCCM compliant alternative to iconify all clients on non-current desktops (without showing their icons) is clearly not acceptable.

2.6. 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 Section 2.8), 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.

2.7. 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.

2.8. 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.

2.9. 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.

2.10. 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.

2.11. Scope of this spec

This spec tries to address the following issues:

- Allow clients to influence their initial state with respect to maximization, shading, stickyness, desktop.
- Improve the Window Managers ability to vary window decorations 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.
- Geometry between desktops.
- 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 behaviour 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` cmdline argument).

3. Root Window Properties (+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
```

3.1. _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).

3.2. _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.

3.3. _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 change in the desktops number 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
```

The Window Manager is free to honor or reject this request. If 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. If number of desktops is shrinking then 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.

3.4. `_NET_DESKTOP_GEOMETRY`

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

Array of two cardinals that defines the common size of all desktops. 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
```

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

3.5. `_NET_DESKTOP_VIEWPORT`

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

Array of pairs of cardinals that define the top left corner of each desktops 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
```

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

3.6. `_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
```

3.7. `_NET_DESKTOP_NAMES`

`_NET_DESKTOP_NAMES`, UTF-8_STRING[]

The names of all virtual desktops. This is a list of NULL-terminated strings in UTF-8 [1] encoding. 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 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.

3.8. `_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 (for example, a taskbar) 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] = 0 /* may be used later */
```

3.9. **_NET_WORKAREA**

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

This property **MUST** be set by WM 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` property set on client windows.

3.10. **_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 the WM, to indicate that a compliant WM 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.

3.11. **_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 WM frame windows of their windows.

4. Other Root Window Messages

4.1. **_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] = 0 /* may be used later */

```

The Window Manager **MUST** then attempt to close the window specified.

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.

4.2. _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

```

This message allows an application to initiate window movement or resizing. This allows the application to define its own move and size "grips", whilst letting the window manager control the actual move/resize. This means that all moves / resizes can happen in a consistent manner as defined by the WM.

When sending this message, x_root and y_root **MUST** indicate the position of the mouse click 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.

```

#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 */

```

The client **MUST** release all grabs on Pointer events, prior to sending such message.

5. Application Window Properties

5.1. _NET_WM_NAME

```

_NET_WM_NAME, UTF-8_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.

5.2. _NET_WM_VISIBLE_NAME

`_NET_WM_VISIBLE_NAME`, UTF-8_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: 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). This property allows taskbars / pagers to display the same title as the window manager.

5.3. _NET_WM_ICON_NAME

`_NET_WM_ICON_NAME`, UTF-8_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`.

5.4. _NET_WM_VISIBLE_ICON_NAME

`_NET_WM_VISIBLE_ICON_NAME`, UTF-8_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.

5.5. _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 as it wishes. `0xFFFFFFFF` indicates that the window SHOULD appear on all desktops/workspaces.

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

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.1[0] = new_desktop
```

The Window Manager **MUST** keep this property updated on all windows.

5.6. **_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 behaviour 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 behaviour for window managers that do not recognise the extensions.

Rationale: This hint is intend 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 behaviour to windows of the same type. Possible examples of behaviour 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_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.

_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** are set **MUST** be taken as this type.

5.7. **_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.

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
```

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. The `WM_TRANSIENT_FOR` hint **MUST** be set to indicate which window the dialog is a modal for, or set to the root window if the dialog is a 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} maximised.

`_NET_WM_STATE_SHADED` indicates that the window is shaded.

`_NET_WM_SKIP_TASKBAR` indicates that the window should not be included on a taskbar.

`_NET_WM_SKIP_PAGER` indicates that the window should not be included on a pager.

To change the state of a mapped window, a Client **MUST** send a `_NET_WM_STATE` client message to the root window (window is the respective window, type `_NET_WM_STATE`, format 32, `l[0]=<the action, as listed below>`, `l[1]=<First property to alter>`, `l[2]=<Second property to alter>`). This message allows two properties to be changed simultaneously, specifically to allow both horizontal and vertical maximisation to be altered together. `l[2]` **MUST** be set to zero if only one property is to be changed. `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.

5.8. `_NET_WM_STRUT`

```
_NET_WM_STRUT, left, right, top, bottom, CARDINAL[4]/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 a 4 cardinals specifying the width of the reserved area at each border of the screen. The order of the borders is left, right, top, bottom. The client **MAY** change this property anytime, therefore the Window Manager **MUST** watch out for property notify events.

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 know about this reserved space in order to be able to preserve the space. Also maximized windows should not cover that reserved space.

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.

5.9. **_NET_WM_ICON_GEOMETRY**

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

This optional property **MAY** be set by standalone 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.

5.10. **_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. First two cardinals are width, height. Data is in rows, left to right and top to bottom.

5.11. **_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.

See also the implementation notes on killing hung processes.

5.12. **_NET_WM_HANDLED_ICONS**

`_NET_WM_HANDLED_ICONS`

This property can be set by clients to indicate that the Window Manager need not provide icons for iconified windows, for example if the client is a taskbar and provides buttons for iconified windows.

6. Window Manager Protocols

6.1. `_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
```

A participating Client receiving this message **MUST** send it back to the root window immediately, by setting `window = root`, and calling `XSendEvent`. The Client **MUST NOT** alter the timestamp, as this can be used by the Window Manager to uniquely identify the ping.

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.

7. Implementation notes

7.1. 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.

7.2. 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).

7.3. 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 behaviour is pretty standard for many toolkits and window managers, and is extremely unlikely to break anything, it seems reasonable to document it as standard.

7.4. Urgency

Dialog boxes should indicate their urgency level (information or warning) using the urgency bit in the WM_HINTS.flags property, as defined in the ICCCM.

7.5. 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.

7.6. Pagers and Taskbars

This specification attempts to make reasonable provisions for WM 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 communicates with the Window Manager using further, WM-specific hints, or some other means.

7.7. Window Movement

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 honour the win_gravity field of WM_NORMAL_HINTS for both MapRequest_and_ConfigureRequest events [1]
- Applications are free to change their win_gravity setting at any time

If application changes its gravity then Window manager should adjust the reference point, so that client window will not move as the result. For example if client's gravity was NorthWestGravity and

reference point was at the top-left corner of the frame window, then after change of gravity to the SouthEast reference point should be adjusted to point to the lower-right corner of the frame.

- 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) [2]
- XMoveWindow(w,x,y) behaviour depends on the window gravity. Upon receiving a request from client application the Window Manager calculates a new reference point, based on the client window's own size, border width and gravity. For given client window dimensions (width, height) and border width (bw), the reference point will be placed at:

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)

The Window manager will use the reference point as calculated above, until next XMoveWindow request. The Window Manager will place frame decorations in the following position, based on the window gravity :

StaticGravity:

window's left top corner will be placed at (ref_x,ref_y)

NorthWestGravity:

window frame's left top corner will be placed at (ref_x,ref_y)

NorthGravity:

window frame's top side's center will be placed at (ref_x,ref_y)

NorthEastGravity:

window frame's right top corner will be placed at (ref_x,ref_y)

EastGravity:

window frame's right side's center will be placed at (ref_x,ref_y)

SouthWestGravity:

window frame's left bottom corner will be placed at (ref_x,ref_y)

SouthGravity:

window frame's bottom side's center will be placed at (ref_x,ref_y)

SouthEastGravity:

window frame's right bottom corner will be placed at (ref_x,ref_y)

WestGravity:

window frame's left side's center will be placed at (ref_x,ref_y)

CenterGravity:

window frame's center will be placed at (ref_x,ref_y)

- Implementation Note for Application developers:

When client window is resized - its reference point does not move. So for example if window has SouthEastGravity and it is resized - the bottom-right corner of its frame will not move but instead top-left corner will be adjusted by the difference in size.

- Implementation Note for WM developers :

when calculating reference point at the time of initial placement - initial window's width should be taken into consideration, as if it was the frame for this window.

[1] ICCCM Version 2.0, §4.1.2.3 and §4.1.5

[2] ICCCM Version 2.0, §4.2.3

7.8. 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 the next revision of this specification.

7.9. 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(String) to attempt to kill a process.

8. References

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

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

9. Copyright

Copyright (C) 2000 See Contributors List

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

10. 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

11. Change history

11.1. 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.

11.2. Changes since 1.0pre5

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

11.3. 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.

11.4. 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 initial 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`.

11.5. 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.

11.6. Changes since 1.0pre1

- Removed implementation note concerning Gnome's (potential) file manager behaviour.
- The Window Movement section of the implementation notes has been revised.

11.7. 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.

11.8. 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`.

11.9. 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

11.10. 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`

11.11. 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:]]`