

# An introduction to the X Window System

An introduction to the X Window System.....	1
Some History .....	2
What is X?.....	2
X Window System Client-Server Design .....	5
The X Server .....	6
X Clients .....	9
Toolkits .....	9
The Window Manager .....	10
The X Display Manager (xdm).....	10
Limitations .....	13
About Exceed.....	14
Exceed Installation steps.....	14

## ***Some History***

The X Window System, known simply as "X", is a portable, network-transparent window system which runs on many different computers. It is frequently used in conjunction with the UNIX operating system. Popular platforms include workstations from companies such as Sun Microsystems Inc, and Silicon Graphics Inc, and standard IBM-PCs running Linux, which is a freely available implementation of UNIX, often including XFree86, a freely available implementation of X. This tradition of the entire system, including numerous applications, being released without a profit being made, makes the latter combination a particularly popular choice, and is preferred by many to less cost-effective, and sometimes less flexible offerings from mass-market operating system companies.

There have been numerous versions of the X Window System, but it was not until the eleventh version, known simply as "X11", that it was widely released and began to gain the popularity it enjoys today. Since then there have been many further releases which added extra functionality while attempting to remain largely backwards compatible. The current release is the sixth one, and is known as "X11R6" or simply as "R6".

The X Protocol was developed in the mid 1980's amid the need to provide a network transparent graphical user interface primarily for the UNIX operating system. X provides for the display and management of graphical information, much in the same manner as Microsoft's Windows and IBM's Presentation Manager.

---

## ***What is X?***

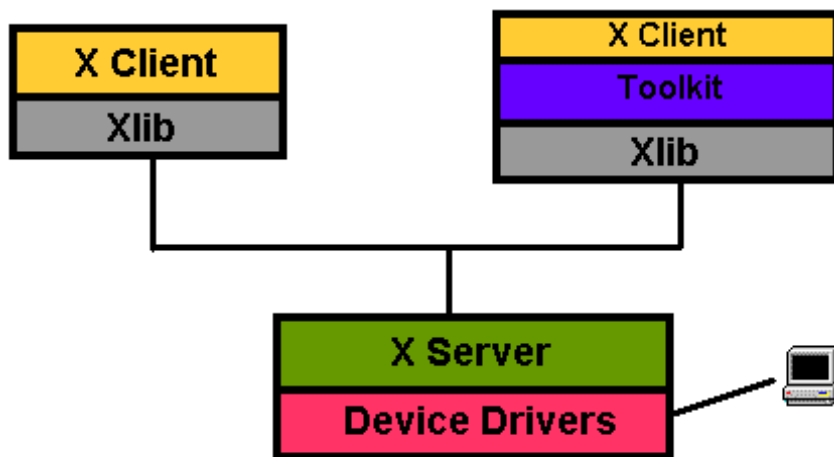
Most graphical user interfaces to computer systems are all-in-one systems. Often, with Windows and MacOS for example, they are integrated right into the operating system. This makes things nice and consistent for the user, but this consistency comes at the expense of choice. Linux, like most forms of Unix, takes a different approach.

Linux uses the X Window System. The X Window System (also called X11, or simply X) de-couples the graphics system from the OS, and splits it into two components: One component, called the **X server**, draws the dots and lines on your monitor; a second component, called the **X client**, tells the server what to draw and keeps track of what's going on within a window (or windows) on your desktop. In fact, these two components can even reside on different computers and communicate across a network. The X server has to run on your desktop, but it can be controlled by a client a continent away, if you so choose.

A standardized protocol, called **X protocol**, lets all this communication take place. The X protocol has been implemented on a wide variety of systems, including all flavors of Unix, Windows, and MacOS. It is important to note that in X terminology, the application talking to the X server is always called a client, even if this "client" is, in reality, a server application. Your company's billing system may run on a centralized server, but it's a client as far as X is concerned; it's a client to the X server on your desktop. This seems to be the reverse of normal client-server nomenclature, but it makes sense if you think about it in X terms.

The **X window manager** is another important part of the X Window System. The window manager is a client to the X server that runs alongside other clients, handling a lot of the management work of manipulating and displaying the windows on your desktop. You can run only one window manager at a time, and they enjoy special privileges. For example, when an application requests a new window on your screen, the X server won't create it until the window manager says where it should be placed.

X further provides a common windowing system by specifying both a device dependent and an independent layer, and basing the protocol on an asynchronous network protocol for communication between an X client and X server. In effect, the X Protocol hides the peculiarities of the operating system and the underlying hardware. This masking of architectural and engineering differences simplifies X client development and provides the springboard for the X Window System's high portability.



The advantages of this approach are many:

- Local and network based computing look and feel the same to both the user and the developer.
- The X server is highly portable allowing support for a variety of languages and operating systems.
- X clients also have a high degree of portability.
- X can support any byte stream oriented network protocol, local or remote.

Applications do not suffer a performance penalty.

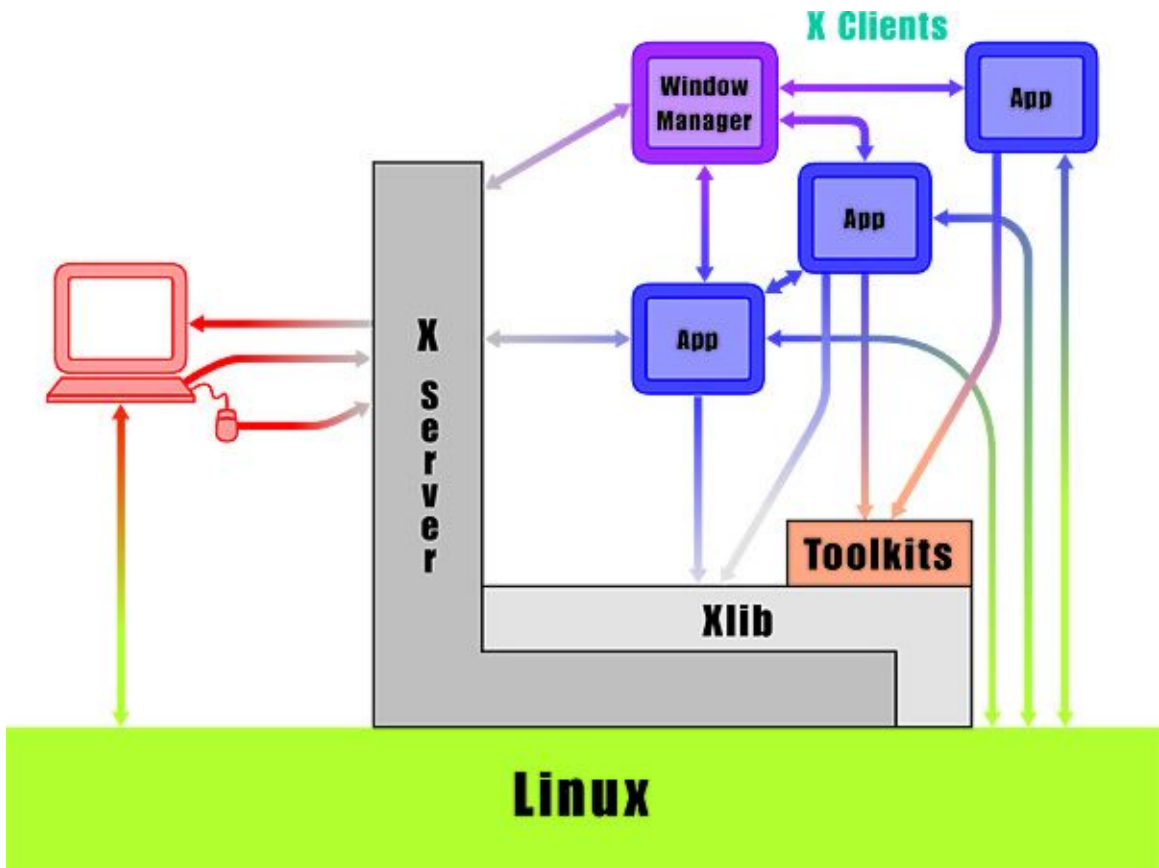


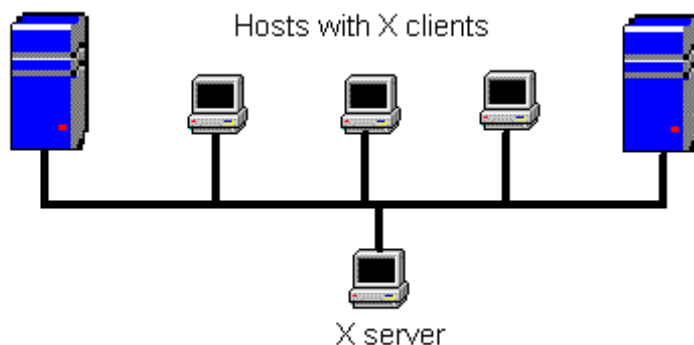
Figure 1 A conceptual overview of the X Window System components

## X Window System Client-Server Design

The design of the X Protocol specifies a client-server relationship between an application and its display. In X, the software that manages a single screen, keyboard and mouse is known as an X server. A client is an application that displays on the X server and is usually termed an X client or simply the application. The X client sends requests to the X server, for example a drawing or information request. The X server accepts requests from multiple clients and returns to the X client replies for information requests, user input and errors.

### The X Server

- Runs on the local machine.
- Accepts and demultiplexes network (or local IPC) based X client requests and acts upon them.



The X server has seamless access to distributed applications.

The X server therefore:

- displays drawing requests on the screen.
- replies to information requests.
- reports an error in a request.
- Manages the keyboard, mouse and display device.
  - Multiplexes keyboard and mouse input onto the network (or via local IPC) to the respective X clients. (X events)
- creates, maps and destroys windows.
  - writes and draws in windows.

### The X Client

Essentially an application written with the aid of libraries (i.e. Xlib, Xt) that take advantage of the X Protocol.

- sends requests to the server.

- receives events from server.
- receives errors from the server.

## Protocol Messages

### Requests

- X clients make requests to the X server for a certain action to take place. i.e.: Create Window
- To enhance performance, the X client normally does not expect nor wait for a response. The request is typically left to the reliable network layer to deliver.
- X requests are any multiple of 4 bytes.

### Replies

- The X server will respond to certain X client requests that require a reply. As noted, not all requests require a reply.
- X replies are any multiple of 4 bytes with a minimum of 32 bytes.

### Events

- The X server will forward to the X client an event that the application is expecting. This could include keyboard or mouse input. To minimize network traffic, only expected events are sent to X clients.
- X events are 32 bytes

### Errors

- The X server will report errors in requests to the X client. Errors are like an event but are handled differently.
- X errors are the same size as events to simplify their handling. They are sent to the error handling routine of the X client. (32 bytes)

## The X Server

The X server is what actually draws the graphics you see on your screen. It communicates with your video card, either directly or via the kernel's device drivers, depending on the X server and card you're using. It takes requests from applications and renders whatever text, windows, or images the client application asks it to display. The X server also lets these applications know about what-ever key-presses or mouse movements the person using the application may be making.

The X server is the only component of the X Window System that has to run on your local machine. Typically, a client application will ask the X server to draw a line or write a word on the screen. The client will say what font to use and where the drawing should take place. The X server then decides how to draw the line. It may hand this task off to your graphics accelerator, if the card is advanced enough. Otherwise the X server will do the work itself.

If it's being asked to render text, the X server will either parse the font file itself, or communicate with a font server to figure out how to render the font. In either case, the X server will then copy the rendered text to your video card's RAM.

The X Server mainly takes care of

- Placement and movement of windows.
- Resizing of windows.
- Iconification of windows e.g. how the window appears when the window is *minimised*.
- Starting and manipulation of windows.
- Control of input to windows.

By far, the most popular X server for Linux is the XFree86 server, but others are available from vendors like Metro Link and Xi Graphics. SuSE and Red Hat distribute the XFree86 server, but with extra drivers included.

You can usually fire up the X server by typing X at the shell prompt. When you do this, you should be rewarded with a screen full of a strange grayish pattern, (known as the "root weave") with an "X" at the center. The X will move around in response to your mouse, but otherwise nothing will happen. Your X server is waiting for a client.

## **X server architecture**

### *Device Dependent Layer*

- It is this layer that is responsible for localizing the X server to the native environment, be it Windows NT or Solaris.
- This layer swaps bytes of data from machines with differing byte ordering. Byte ordering (MSB and LSB) is noted in each X request.
- This layer hides the architectural differences in hardware and operating systems.
- Maintains device driver dependencies for keyboard, mouse and video.

### *Environment Architectures*

- **Single Threaded Architecture** - The X server is a single sequential process using the native time-slice architecture for scheduling demultiplexing requests and multiplexing replies, events and errors among X clients.
  - **Multithreaded Architecture** - The X server is a multithreaded process capable of exploiting the nature of the operating system by breaking jobs into multiple threads for the operating system and hardware to perform. True pre-emptive multitasking, multithreaded environments offer a high degree of power for the X server.
-

## X Clients

As I've already mentioned, X clients are applications that communicate with the X server. When any application that requires graphics starts up, it immediately establishes a connection with the X server and tells it what windows should be created to support it. It tells the X server what size and shape they should be, and what they should display. The client also informs the X server what events (mouse-clicks or keystrokes, for example) it wants to know about.

Though they are not common, there are some X clients that don't create any windows at all. For example, the program `xcpyinfo` simply provides a command-line printout of the capabilities of the X server it's talking to.

Following the client's instructions, the X server tells the client when a relevant event occurs. A wide variety of things can qualify as events. One can occur when a window gets moved, if a key is pressed, if the mouse gets moved, or if a window gets dismissed. It's up to the person who writes the client application to define which events the client cares about. Some clients -- for example a program that displays a clock on the screen -- may not care about keystrokes. But the clock program may well care to know when another window has been placed over top of it. That way it could suspend itself until it heard from the X server that it was visible again (another event).

---

## Toolkits

While every X client uses a library of functions known as Xlib to communicate with the X server, most don't directly call the functions in Xlib. To draw a simple 3D button using Xlib directly, a client must draw a couple of rectangular windows, shade them to give the appearance of beveled edges, monitor mouse events, be able change the button's appearance if it gets pressed, and so on.

To make this job easier, there are additional libraries of functions, known as toolkits, that include oft-used combinations of the Xlib functions. These combinations are called widgets. A developer can pluck a widget from one of these toolkits to, say, create a button that displays "Okay" when pressed. The toolkit takes care of all the little Xlib details.

In addition to simplifying the work of the programmer, toolkits give applications consistency. Applications made from the same toolkit simply look like they belong together. By tweaking the individual toolkit widgets, developers can change the look and feel of all the applications that use that particular toolkit. Some popular toolkits include Qt, GTK+, Motif, XForms, Tk, and XView.

Widget sets are not normally inter-changeable because they usually expose different interfaces to the programs that use them. However, sometimes it is possible to replace a widget set.

You can do this with Xaw, the Athena Widget Set. The original Xaw widget set has a simplistic 2D look, but its later developers have created new widget sets for Xaw that look different from the original. These usually retain its programming interface. So if you want, it's possible to put a replacement Xaw library on your system and still

have all your Xaw applications run. They'll look different, but with the same programming interfaces, everything will still work.

The first of these modified Xaw widget sets, Xaw3d, takes the standard Xaw look and makes it three dimensional. Other Xaw replacements attempt to simulate the look of other windowing systems. Xaw95 looks like the widgets in Windows 95 and neXtaw looks like those of NeXTSTEP.

---

## The Window Manager

X's window managers do a lot of work. The layout of windows on your desktop is controlled by the window manager. Frames around the windows are drawn by the window manager. The particular icon a window can take when shrunk depends on the window manager. Some window managers let you switch between multiple desktops. Some provide a virtual desktop. If you want to change the size of a window, you need to talk with the window manager. In short, the whole look and feel of your computer is completely influenced by your window manager.

So which window manager is best for you? Maybe you'd like one that feels like Windows 98 or the Macintosh? Perhaps you have an older system and want to use as little memory as possible? Maybe you're looking for a compact window manager for your notebook? Ease of configuration may be your top priority. Or perhaps you work with different systems from different vendors and would like a consistent look and feel when you switch between them? There are many window managers out there to satisfy every need.

---

## The X Display Manager (xdm)

The X Display Manager (xdm) **is an optional part of the X Window System** that is used for login session management. This is useful for several types of situations, including minimal "X Terminals", desktops, and large network display servers.

Since the X Window System is network independent, there are a wide variety of possible configurations for running X clients and servers on different machines connected by a network. XDM provides a graphical interface for choosing which display server to connect to, and entering authorization information such as a login and password combination.

Think of XDM as providing the same functionality to the user as the getty utility. That is, it performs system logins to the display being connected to and then runs a session manager on behalf of the user (usually an X window manager). XDM then waits for this program to exit, signaling that the user is done and should be logged out of the display. At this point, XDM can display the login and display chooser screens for the next user to login.

In the case of XDM, X Servers do not initially act passively as they actually have to act as clients to the XDM process while the user is logging in (using XDMCP, see below). After that X Servers sit and passively wait for x client requests.

So, the X Display Manager (xdm) keeps X running on displays and provides basic user session management. As part of this job, xdm provides an X-based login

function via the Xlogin widget, performing essentially the same services as the usual unix getty or login programs. It also provides client cleanup facilities for displays on which the server is no longer running. Finally, xdm automatically generates authorization information which can be used by the X display server to control which users on which hosts may access any given display.

Throughout the rest of this section, the following terms will be used as defined here unless otherwise specified:

- **Server** will designate an X server.
- **Display** will refer to either a workstation display or an X terminal.

## Whence it Came

xdm was developed by Keith Packard shortly after he joined the X Consortium, primarily as a result of his frustration with trying to manually control X from an ASCII login environment. It was released with X11R3, although there were copies floating about prior to that. Other companies such as Visual were developing their own display managers, but xdm quickly became the standard for most companies. The remaining major holdout is IBM, which ships xdm as an unsupported client and expects that their own *x\_st\_mgr* will be run instead (for X terminal support; no support is provided for managing workstation displays).

The **X Display Manager Control Protocol (XDMCP)** was introduced with X11R4. XDMCP requires that when a controlling client disconnects from a display server, the remaining clients associated with that session also be disconnected. **It also allows servers to talk to xdm without requiring xdm to have explicit, prior knowledge of each server.**

Also introduced with X11R4 was much better support for security. This support was implemented in the display server, in XDMCP and xdm, and Xlib. A new pseudo-client, *xauth*, was provided to aid in managing the new security files. At that time, only support for MIT-MAGIC-COOKIE-1 was provided by the Consortium. Three modes of session initiation were provided with R4: **BROADCAST**, **DIRECT** and **INDIRECT**. Of these, only the first two worked.

With X11R5, xdm added real support for the **INDIRECT** mode, so that users may now select from a list of hosts the host they wish to log into. As part of this, a **chooser** client is provided, which displays the list of available servers. Also provided with Release 5 is XDM-AUTHENTICATION-1, which allows an X terminal to authenticate xdm, **so that passwords are sent only to xdm, not just any program playing password collector.** This last feature is not supported by most X terminal vendors.

## Why Use It?

Besides the security features and convenience of XDMCP, xdm is the universally accepted method of managing X displays. Like all Consortium software, it is freely available in source form. Most major system and X station vendors support it. It is the only method of controlling X stations common to all vendors.

It provides easily-managed, centralized control over a site's X environment. It can provide rudimentary load balancing of the X environment.

## How It Works

xdm has a list of servers it is to manage. It also listens on the XDMCP port for others servers requesting management. **For each managed server, xdm spawns a copy of itself in which it provides an Xlogin widget.** When a user has been authenticated via the normal password techniques, xdm runs the session program (usually a script) to set up the user environment. This includes attempting to run a user-provided setup script or program. **Until a user is authenticated, xdm maintains tight control over the display.**

After a user has logged in, xdm writes a session key in the **.Xauthority** file if the **authorize** resource is set for that display in the **xdm-config** file. For local servers this key is passed through a file with the **-auth** switch on the display server command line. For remote servers (X terminals) this information is passed via XDMCP.

When a user terminates the last client started by the session program, or the session is otherwise terminated, xdm makes sure all the clients attached to that server are terminated.

When a session ends normally, xdm sends the server a **reset** request. This tells the server to drop all connections and reinitialize itself. At this point xdm provides another Xlogin window. If the server has terminated for any reason xdm will restart it if possible (i.e. it will be possible for local servers only). In normal operation this only applies if the **DisplayManager\*DISPLAY\*terminateServer** resource is set.

The xdm database (other than the authority files) is comprised of flat files. The first file read is xdm's config file, which contains the names of the other database files to read, along with a few high-level resources such as **authorize**.

Several signals are important to xdm. (All signals will be referenced by their POSIX names.) **SIGHUP** tells xdm to reread the configuration file and the servers file. **SIGINT** may be required to coerce xdm into rereading the resources file on some systems. This is very non-standard. **SIGTERM** tells xdm to shut down all managed servers and exit.

To force a display server reset, xdm sends the server a **SIGHUP**. xdm sends a **SIGTERM** signal to force a server to exit.

## Communication Modes

XDMCP allows servers not listed in the **servers** file to communicate with xdm via one of three modes, **DIRECT**, **INDIRECT**, or **BROADCAST**.

**DIRECT** mode is used when a server is to attempt to directly communicate to an xdm process on a given system. In **BROADCAST** mode a server places a general request for management on the network. The first xdm to respond as available becomes the server's manager. **INDIRECT** mode allows a server to communicate with an xdm process on a given system to find out where it should really be looking for management. This is especially handy in environments running multiple networks, and environments such as DECnet, where no broadcast function is available.

While XDMCP is usually thought of for X terminals, newer servers running on system displays also speak XDMCP. In theory these servers may be brought up as part of the system boot procedure in lieu of (or in addition to, in the case of IBM) a console **getty** process. This can be a problem if the system running xdm for your server is

down. In practical terms this is a terrible idea. About all it's useful for is debugging xdm configurations; local servers usually start much quicker than X terminals booting across networks.

## ***Limitations***

If xdm is run on a workstation console display, that display must run X. xdm does not work well in environments where multiple graphics systems attempt to share a display. As most vendors have merged X with their other graphics packages, this is not nearly the problem it formerly was.

Console output redirection was somewhat vendor specific prior to Release 5. Some vendors provided a solution to this problem. With other vendors you had to supply your own solution. A new client, ***xconsole***, was provided in R5 to solve this problem for X in general, not just for xdm. See the man page for xconsole for further information.

While some systems allow you to run a separate display server on each screen of a display (a nonstandard feature), xdm provides no support for this concept.

xdm creates one process per display for login purposes. After login, at least one process is usually consumed by the session script. As some login process and a login shell must normally be provided anyway, the number of processes created by xdm is not really excessive, but you should be aware of what's out there. The last client invoked in the ***session*** file may be invoked with the ***exec*** command found in all common shells, thus saving one process per server.

Running multiple display managers on multiple systems can provide some ad hoc load balancing, but if you need firm controls over system loading you will have to tweak the configuration files yourself. Some vendors provide load information beside each system available through the chooser client.

## About Exceed

Exceed basically installs in your PC the Exceed X Server. In the X Window environment, the Exceed X server is also referred to as an X window terminal or display server. Without Exceed X server type of software, X applications are accessible only via X terminals, UNIX, Linux, and VMS workstations.

Exceed works with your network transport software (TCP/IP, DECnet, or IPX/SPX) or your modem, to access X Windows applications on host computers running the X Window System. The host can be any operating system that is running the X Window environment.

A window manager interprets requests or commands entered on the PC and sends them to the Exceed X server. The X server sends the request to the X application, and the application sends instructions to the Exceed X server to display.

If you run an X window session without a window manager, you cannot perform window operations such as resizing, moving, and iconizing. Overlapping windows can make hidden parts of underlying windows inaccessible.

**Exceed uses two types of window managers: local and remote.** Local window managers run on your PC, while remote window managers run on a remote host. Running Exceed with a remote X window manager generally increases network traffic and may decrease overall system performance.

When you log into the UNIX host via the CDE (Common Desktop Environment) display manager using XDMCP, a handshake implemented by the X protocol is employed. CDE does not begin unless it detects a supported X server (Exceed is of course supported – meaning sufficiently X compliant).

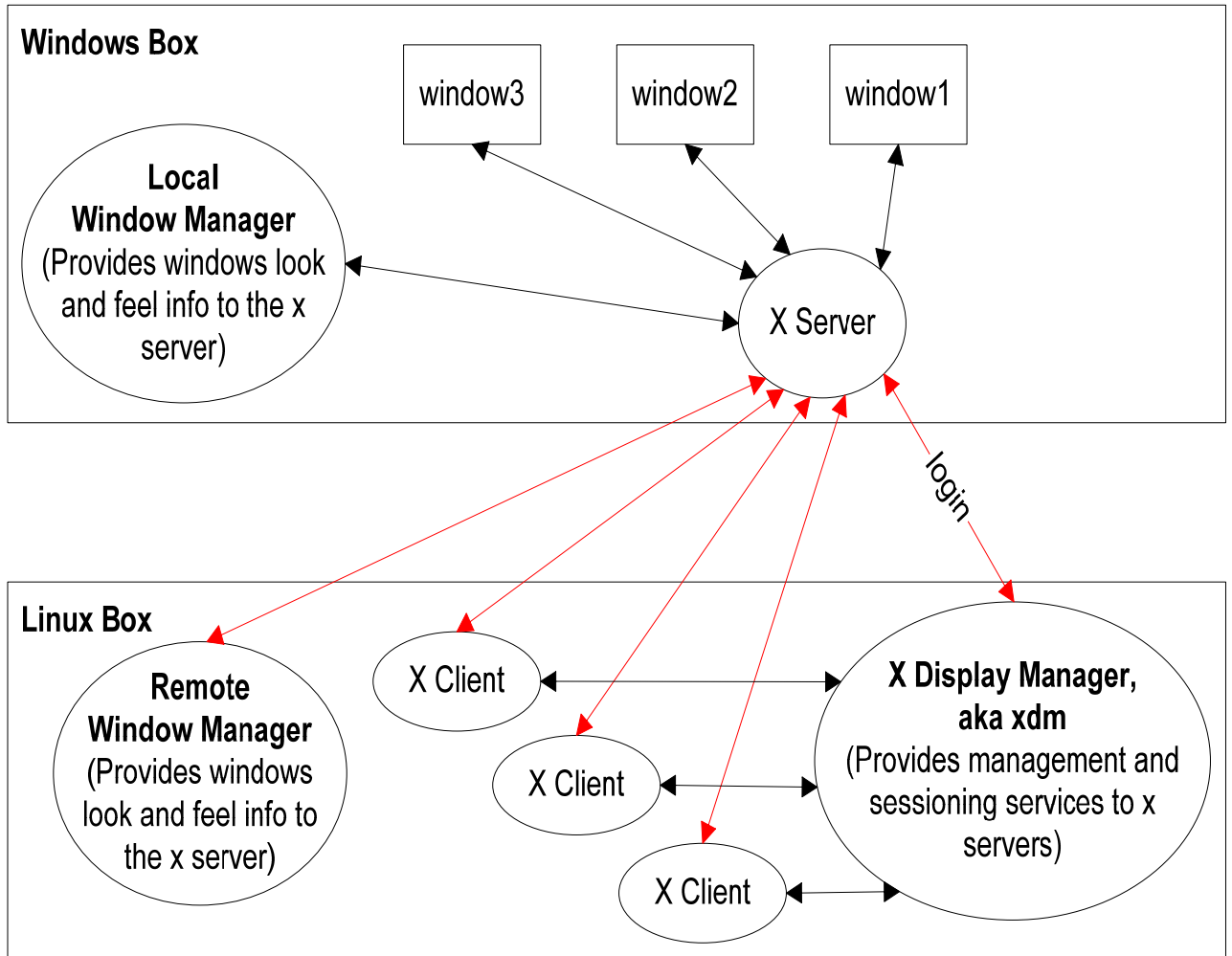
Exceed can run with a local or a remote window manager. Running a local (to the x server) window manager minimizes network traffic and makes the display more responsive.

## Exceed Installation steps

1. Install exceed on you PC
2. On your linux box(es)
  - edit your Xaccess file to include the PC's IP address
  - comment out the last line in the xdm-config file  
*DisplayManager.requestPort: 0*
  - start xdm

## Notes

- all xdm config files can be found /etc/X11/xdm
- you may also have to edit the xhost and xauth files (they are in a different directory)



**Figure 2** A diagram depicting interdependencies and interactions between various X window system entities during a Windows-Linux Exceed facilitated remote X windows session. Note that you cannot have both a local and a remote window manager. Local window managers are preferable as they minimize network overhead and lighten the processing load on the host where the X Clients are running