XVTnet User Manual

Published By

Providence Software Solutions, Inc.
201 Shannon Oaks Circle
Suite 200
Cary, NC 27511 USA

Revision History

First Printing ..... June 2009 ..... XVTnet Release 5.8

Contents

Introduction

Current Internet/Intranet solutions

Currently many business and personal computer users are using the Internet and Intranets to communicate, exchange information, and perform business in general. The current technologies for these solutions include such items as WEB thin clients, JAVA applications, and e-mail systems. WEB thin clients allow for information to be displayed in a relatively straightforward manner as well as allowing some interaction. JAVA applications or applets are generally used to assist in the interactivity of WEB pages. E-mail is used for user-to-user correspondence as well as file transmittal.

Because such efforts are producing specialized code for the Web technology, not all IT experts are blindly endorsing this new framework as the ultimate panacea for developing applications. In fact, a more traditional school of thought still prefers the traditional client/server model for several reasons. Traditional client/server applications can provide graphics interfaces far superior to stateless Web interfaces through higher performance, more complex visualization and far superior user experience. With the next generation of 64-bit processors, and the continuing decreasing costs of memory, it now becomes possible to store large data sets in memory, thereby taking advantage of computational models that are much faster than traditional relational database access. Given the increasing availability of communication bandwidth, it is becoming easier to develop balanced solutions where the tradeoff between computation and communication are adjusted at will. Given that deployment of client/server applications over the Web has become a seamless process, it is becoming easier to leverage previous investments to deploy applications over the Web.

One way this trend is seen is in the use of web services which are retrofitted to legacy applications to serve their processing ability remotely to clients across the network. Another way is to actually deploy the graphical user interface (GUI) client of an existing application across the web. We will discuss XVT's approach to this second method of Internet deployment. We will also describe the benefits that are offered by the XVT development framework which allows standalone applications to be run over the web simply by recompiling them; this enables the deployment of legacy applications over the Internet without the large investments usually associated with Web technology migration efforts.

The XVTnet Technology

The XVTnet technology provides a completely new capability for developing Internet and Intranet solutions. This technology provides a robust, flexible, and secure capability for providing complete large-scale applications of virtually any type across the Internet or Intranet. XVTnet is not information based, it is application based, and thus provides a true solution for Internet applications

instead of a combination of information transfer mechanisms. With XVTnet, the application resides on a host machine, yet it is executed and accessed from a remote site. The host machine can support multiple users simultaneously. The host and client are platform independent, thus any host can be accessed via any client and vice versa.

Differences from Other XVT Platforms

The XVTnet platform is somewhat different from other XVT platforms in that it is a combination of an existing XVT platform, an Internet-enabling component, and NetLink, a remote application viewer.

An explanation of XVTnet will provide you with a clearer understanding of what will and will not work with it. With XVTnet, the application that you compile and execute becomes a server or server application with little actual user interface. You create and interact with your user interface components on a separate computer, the client or remote system. All code that is not user interface-related occurs on the server side. This includes calculations, memory allocation, file access, etc.

Because of the server and client interactions, most platform-specific operations that are not user interface-based will work properly. However, any platform-specific code that deals with the user interface will not work. One example is creation of non-XVT windows and controls in your application. While an application using non-XVT windows will compile and possibly run, the results are not what are expected—the non-XVT windows created by your application will actually display on your server machine. Another example involves the function _beginthread which is specific to the Windows 32-bit platform, yet will work properly under the Win32 version of XVTNET.

We currently do not support attributes that must be used before xvt_app_create. These attributes include:

- ATTR_MEM_MANAGER
- ATTR_MULTIBYTE_AWARE
- ATTR_FONT_CACHE_SIZE
- ATTR_DEFAULT_PALETTE_TYPE

Non-portable attributes

On Windows platforms, there is a slight difference with how MDI mode is handled. If a window is to be of any other type than W_DOC, its parent must be set to TASK_WIN and not SCREEN_WIN.

The XVTnet platform has its own specific non-portable attributes, but may also support some of the non-portable attributes of its server and client platform.

Any non-portable attributes which the platform supports will be provided as a XVTnet platform-specific attribute with a name similar to that used by the platform as follows:

ATTR_NET_RWIN_PM_SPECIAL_1ST_DOC

To allow the application developer to properly determine the current platform, XVTnet provides three platform determination methodologies.

The first platform determination is provided as a platform-specific #define to denote the system as a XVTnet platform. As with any XVT platform, the XVTWS #define is used for the XVTnet server platform and will be equal to NETWS.

#if (XVTWS == NETWS)

The XVTnet platform also supplies proper #defines so that the developer may detect which server platform is being used. To test the server platform being used, the platform-appropriate XVTWS may be used with NET_ prepended as follows:

#if (XVTWS == NET_WIN32WS)

This lets existing platform switches based on XVTWS to continue to work.

The final platform determination to be made is the determination of the platform being used by NetLink, the remote application viewer. To determine this, the developer is provided with a XVTnet attribute for retrieving the client system's platform as follows:

if (xvt_vobj_get_attr(NULL_WIN, ATTR_NET_REMOTE_WS) == WIN32WS)

Note that this information is retrieved at run time and must be coded as such (not using #if, but if).

When using XVTnet attribute values that pertain to specific platforms, it is not an error to use an attribute value that is not supported by the platform. In this case, the attribute value will simply be ignored in the case of xvt_vobj_set_attr() and will return 0 in the case of xvt_vobj_get_attr().

Please see the section on XVTnet-specific attributes for a full list of supported attributes for the current platform.

WC_ICON controls aren't supported. This may change in future revisions, but the simple explanation is that on some platforms, the imagery data associated with an icon must be bound into the application at link time with the rest of the resources, which isn't possible when the universal thin client NetLink is what the

user is really running locally. Work-arounds for this would include using XVT_IMAGE images instead.

For the same reason, custom, application-specific cursors are not supported.

At this time, XVTnet is restricted to the use of the portable, bound help viewer for application help files.

XVTnet does not support conversion between PICTs and files due to platform dependencies. The problem is that to get picture data to and from a file requires the use of xvt_pict_lock(). xvt_pict_lock() returns a pointer to the platform specific data for the picture. This data is client-side specific, and thus it isn't always meaningful to the server. Future revisions may implement a platform-independent PICT, but for now this functionality is not implemented.

Comparisons with Existing Technology

HTML/CGI

HTML is the language used by WEB thin clients to display information and HTTP is a protocol for transferring HTML documents across the Internet. HTML stands for Hyper-Text Markup Language and HTTP stands for Hyper-Text Transfer Protocol. HTML is essentially a format for displaying text and graphics as well as allowing a user to select links (hypertext) to access other HTML documents. This is how a WEB page allows you to select link after link to traverse information. This is an effective way of displaying information, but is quite restrictive in user interaction capabilities.

CGI or Common Gateway Interface

CGI is a part of HTTP which allows users to enter information into a web page. This information can then be transferred to a host machine by pressing a special link on the page. This provides some added interaction with the host machine, as users can actually send information to the host, but is still limited in its capabilities. XVTnet allows full user interaction with the application. It provides for user interactions with several different types of controls as well as direct mouse and keyboard control of the application. It also provides for many different types of graphical display options, not simply text and images.

JAVA

JAVA is a platform independent object-oriented language which can be used to develop Internet/Intranet applications. The problem with JAVA is that the application still must execute on the client machine. This requires that the application be downloaded to the client, and then executed. This can result in

long start times for applications to download. There is also a possible security risk because the application is running on the client machine. This opens the possibility for the application to access to the client's file system, operating system, and anything else. Finally, because the application runs on the client, it uses the client's CPU for all of its processing. While this distributes CPU load, it also causes the application to run only as fast as the client CPU. Because XVTnet applications run on the host machine there is no need to download code before executing. This results in instantaneous application execution. This also results in complete client security. The host application only has access to host resources.

The toolkit is C/C++ based and thus can use any C or C++ based development tools to assist in the development process. Because XVTnet applications run on the host machine there is no need to download code before executing. This results in instantaneous application execution. This also results in complete client security.

Finally, because the application runs on the host, it uses the CPU of the host. This can be a benefit in that a large scale mainframe can be performing calculations and displaying them on a less powerful, remote desktop/laptop.

The X Window System

The X Window System is in many ways similar to the XVTnet technology, but with a few differences. The most significant difference is in bandwidth requirements, i.e. the required speed of data transmission for reasonable responsiveness. The X Window System requires a high speed LAN for reasonable interaction. This is most likely due to the software layering system used by X. The X Window System is built upon the X protocol, a low-level window and drawing protocol. Layered above the X protocol is the X Toolkit.

The X Toolkit defines widgets (controls) which the user can manipulate. Layered above the X Toolkit are widget sets and window managers such as Motif and Openlook. The X protocol does not understand widgets directly and therefore can-not optimizes widget interaction for low bandwidth situations such as the Internet. The X Consortium is currently working on a low bandwidth version of the X protocol called X Fast. This protocol may assist in some the bandwidth requirements, but X is still layered in software and as such may still have a high bandwidth compared to XVTnet.

The X Window System also defines its own "look and feel" for X applications. Microsoft Windows and Macintosh users may not be familiar with the X look and feel and thus may not be able to use the applications to their best ability. The XVTnet technology maintains the look and feel of the client's operating environment, independent of the host system OS.

Winframe

Winframe is another low-bandwidth GUI based terminal system. There are several distinct differences between Winframe and XVTNET. Winframe is based upon the Microsoft Windows GDI (Graphics Device Interface). The GDI was originally designed for writing graphics device drivers to be used by Microsoft Windows in rendering its graphics. GDI defines a set of graphics primitives with which to display the user interface. Winframe captures these GDI calls and transmits them across the internet for display on a client system. Although this is an excellent way to display existing Windows applications on a remote terminal, it is not the optimal method for developing Internet applications. XVTnet works at a higher level than GDI. It defines a set of GUI primitives instead of a set of graphics primitives. Because of this, XVTnet understands what an "Edit box" or a "List box" is, instead of simply a set of rectangles and text. This allows XVTnet applications to be more user responsive and interactive. This also allows the application developer to write the application to best take advantage of the features offered by the protocol. Yet another advantage to this "high level" protocol is the ability for XVTnet applications to have the same "look and feel" of the client system, NOT the host (Macintosh clients look like Macintosh applications, Windows applications look like Windows, etc.).

As an example, if the user is scrolling through a list box filled with items, XVTnet transmits those items to the client and requires no further interaction as the user is moving around within the list. Winframe must constantly send new display information as the user scrolls around in the list. This severely limits the speed with which the user can interact with the control. An example of how the developer can utilize the capabilities of the technology to his benefit is in bitmap/image display. The developer can transmit all images required by the program upon application startup. This then gives the user a one-time startup delay, with no further delays in interaction.

Another difference between XVTnet and Winframe is that the XVTnet technology is platform independent, Winframe is not. While Winframe clients can reside on different types of machines, the Winframe host must reside on a special Winframe version of Microsoft Windows. XVTnet is completely platform independent. The host requires no special operating system and versions are available for Windows, Macintosh, Linux, and UNIX.

The Toolkit

The XVTnet toolkit is what allows application developers to develop XVTnet server applications. The XVTnet toolkit is a set of C sub-routine calls which can be used to develop XVTnet applications. Because the toolkit is based on the C language, most existing C/C++ development tools can be used to develop an XVTnet application. All that is required is that the user interface be written with XVTnet routines. This also makes XVTnet a completely embeddable technology. The

XVTnet subroutine set is based on an industry standard API for developing cross-platform applications, XVT. XVT is already an industry leader in cross platform development and greatly simplifies the design and coding of the GUI.

The XVTnet toolkit has three basic components:

A. User Events

B. GUI Commands

C. Information Requests

User Events

Events are the internal representation of user actions. There are 21 events defined for XVTnet. These events include mouse actions, window sizing, window manipulations, control manipulations, keyboard actions, display updating, and timing events. The developer can receive these events and use them to design how his application will interact with the user. XVTnet is completely platform independent. The host server and the thin clients require no special operating system and versions are available for Windows, Macintosh, Linux, and UNIX.

The XVTnet toolkit has three basic components:
A. User Events
B. GUI Commands
C. Information Requests

Yet another advantage to this "high level" protocol is the ability for XVTnet applications to have the same "look and feel" of the client system, NOT the host (Macintosh clients look like Macintosh applications, Windows applications look like Windows, etc.). The developer can also choose which events need to be handled and which ones don't; thus, unnecessary events will not take up extra bandwidth.

GUI Commands

GUI Commands are used to create display objects or to display information to the user. There are hundreds of commands in the XVTnet toolkit. These commands include control creation and manipulation, clipboard handling, dialog and window creation and manipulation, drawing, font handling, image manipulation, printer handling and text editing.

Controls provided in the toolkit include all the standard GUI controls; push buttons, radio-buttons, check-boxes, edit-boxes, list-boxes, popups, etc. This gives the developer a full suite of user controls to work with. Clipboard support allows XVTnet developers to access the client's clipboard and retrieve as well as place data on that clipboard. Dialogs and windows can be created, interacted with

and destroyed as necessary by the developer. All controls and drawing commands are used within windows and dialogs. A full suite of drawing commands are available to the developer including arrows, arcs, icons, images, lines, ovals, complex pictures, pie-charts, polygons, polylines, rectangles, rounded-rectangles, and text. In addition the toolkit supports clipping, multiple fonts, scrolling and variable pen and fills parameters for all of the above. Font handling is also included in the toolkit. Any font installed on the client machine will be available to the host application. Full image manipulation is included which allows dynamic image manipulation by the host program with automatic optimized update to the client. Images can also be automatically transferred via JPEG compression to optimize image display speed. The thin client allows the user to choose how to display images (quality vs. Speed).

Printing capabilities are also included in the toolkit and allow the application developer to perform any of the listed drawing commands on the client's printer as well as the screen. A full set of text editing capabilities are also included. These capabilities are optimized such that most of the text editing manipulation is done on the client so text editing is smooth and fast.

Information Requests

Finally, information requests are used to gather information about the client to the host. These information requests include such things as listing the fonts available, retrieving printer information, retrieving clipboard data from the client, retrieving values from controls, retrieving user screen information etc. These information requests allow the developer to write his application so that it can perform optimally on all systems.

Security

XVTnet is an extremely secure technology for several reasons.

Host-side Application Execution

The application executes on the host machine, not the client. This restricts the application developer from accessing the client's file system, operating system, etc. The application developer only has access to the client's user interface.

Packet Encryption

All transmissions between machines can be encrypted using a public-private key encryption scheme. This prevents any outside hacker from "spying" on an XVTnet session.

User to Host Verification

Users can be verified to the host without passwords being transmitted. The remote system can request a password from the user, then use this password as a key to encrypt the data. The host must also know the key to decrypt the data, thus the password must be known by the host, but is never transmitted.

Host to User Verification

Hosts can be verified to the user via the methodology described above. If the host does not have the user's password, it cannot decrypt the user's data. Thus the host is verified to the user as well as the user verified to the host.

Bandwidth

What makes the XVTnet technology feasible for the Internet is its low bandwidth requirements. XVTnet achieves this low bandwidth capability because it uses a high-level event model combined with high level GUI commands. Because XVTnet understands such items as controls and windows at a high level, much of the work to implement the GUI is performed on the client. Also for any given event or command, only the most minimal of information is transferred. More lengthy operations have also been optimized with the use of compression techniques. The most obvious of these is the automatic use of JPEG compression for transferring images. This JPEG compression occurs even if the original image is not in JPEG format.

The following are a few examples of bandwidth requirements and times associated with them:

Creation of a dialog with 20 controls<TABLE FIXEDWIDTHS>

MACROBUTTON HtmlDirect *

Command	Unit	Qty	Total
Dialog Creation	8 bytes	1	8 bytes
Control Structures	28 bytes	20	560 bytes
Total			568 bytes

With a 14,400 baud connection this dialog would be displayed in 0.38 seconds and would be under client control for screen updates and simple interactions.

Spreadsheet 5 columns wide by 10 lines high with an average of 10 chars per cell.

Command	Unit	Qty	Total
Lines Horizontal	8 bytes	5	40 bytes
Lines Vertical	8 bytes	10	80 bytes
Text in cells	14 bytes	50	700 bytes

Total 820 bytes

With a 14,400 baud connection this spreadsheet view would take approx. 0.57 seconds to draw.

Internet Applications

What makes the XVTnet technology feasible for the Internet is its low bandwidth requirements:

Command Unit Qty Total
Dialog Creation 8 bytes 1 8 bytes
Control Structures 28 bytes 20 560 bytes
Total 568 bytes
Transit Time @ 56K ~.095 sec
Command Unit Qty Total
Lines Horizontal 8 bytes 5 40 bytes
Lines Vertical 8 bytes 10 80 bytes
Text in cells 14 bytes 50 700 bytes
Total 820 bytes
Transit Time @ 56K ~.143 sec

More Interactive WEB Pages

One use for XVTnet technology on the Internet is the ability for businesses to provide more interactive WEB pages. Business is currently restricted to simple HTTP/CGI interaction. This can make the implementation of even simple systems difficult. Take as an example an inventory check/pricing system. The company must create HTML documents listing their entire inventory as well as current pricing for each item. Because HTML documents are static, the company will have to make sure to "update" their documents periodically. A simple XVTnet application can be written which reads the inventory database the instant the user requests, display this information in a list box, and update the in- formation if it changes while the user is on-line.

On-line banking

On-line banking and investing is another example of an XVTnet application. Because of its built-in security, the bank or investment firm can guarantee the proper user is on-line. Once on-line the user can have instant access to all of his financial data in the form of spreadsheets. The user can click a few options and answer a few prompts to transfer money, make payments, etc. The on-line bank could even have a Quicken style interface to checkbook registers where the user can enter new transactions and the bank could clear them as they arrive. The possibilities are endless. XVTnet application can be written which reads the

application database the instant the user requests, display this information in a list box, and update the information if it changes while the user is on-line.

Intranet Applications

Client-Server

The XVTnet technology can be used in the Intranet community to replace the current use of client-server database applications. With client-server database applications the data necessary for an application is accessed across the network. This is generally expensive and depending on the application, may not be as efficient as XVTnet. With XVTnet, only the data which is relevant to the user is transferred across the network, if the user can't see the data it doesn't need to be sent. XVTnet may also be more secure because the actual data never leaves the host machine, only a view of the data. Finally, the XVTnet client can be distributed free as compared to $1000 to $1500 per client for ORACLE.

X Window System replacement

Currently many corporations are using The X Window System to develop corporate Intranet applications. XVTnet can be used as a replacement for this with the following advantages. Due to the lower band-width requirements, slower LAN speeds can be used for wide area Intranets and remote access to applications. If PC X-Servers are being used as the client for these applications, XVTnet can represent a significant cost savings (< $100 compared to >$500). The look and feel of XVTnet applications match the platform upon which the client resides, thus making applications more user friendly and reducing training costs.

On-line Gaming

On-line games and gaming are very popular, but currently to play an on-line game you must first purchase some software that allows you to connect with the game. With XVTnet the user would already have the software necessary to play the game. Of course 3D action games (such as DOOM etc.) would not be possible, but strategy games, card games, and board games are all excellent candidates for XVTnet. Because the application resides on a host machine, multi-player games can be developed very easily since all games are actually played on a single machine.

Educational Software

Educational software is another candidate for XVTnet technology. Large volumes of data can be stored on the server for use in the educational software, including assignments, testing and scoring, interactive learning systems, etc.

Subscription Software/Software as a Service (SaaS)

Subscription software is yet another possibility with the XVTnet technology. Because the developer is able to develop an entire application of virtually any type, software firms may be able to lease the use of these applications (over the Internet) to their subscribers. No longer would users be required to go to the store, purchase software, install the software onto their machine, and configure the software in order to use it. They could simply "subscribe" to a software provider who would maintain their data files as well as the applications. The provider would provide backup services, etc. so that the user doesn't have to worry about such things.

Multi-user Interactive Software

Any type of multi-user interaction is an excellent candidate for XVTnet technology.

Enhanced Terminal Software

Many corporations are still using terminal based systems with connections for central computer access. These terminal based applications can be replaced with user friendly GUI applications without a lot of hardware costs. The current connections can be used as PPP points to access XVTnet applications. Not only does this make applications more user friendly, it also allows each user to run multiple applications simultaneously on his/her display.

How XVTnet Works

XVTnet is a GUI (Graphical User Interface) Internet applications development technology. Because XVTnet is a GUI development system, it is important to know how GUIs are developed to understand how XVTnet works. GUIs or graphical User Interfaces generally follow the same model, the event-driven model. This model was developed by Xerox in the 1960s and has progressed to the Macintosh, PC, Linux, and UNIX environments. The event model works as follows. A user interacts with a workstation. These interactions are represented to the system as Events. Events are such things as mouse movements, menu selections, button presses, keyboard actions, etc. When the system receives an event, it executes code to handle the event. This code produces some (usually graphical) response to the user. The response may in-turn require more interaction with the system thus generating more events, which cause more responses, etc. This cycle of events and responses allows the user to perform any action necessary to complete his job. XVTnet transports GUIs across the net. The XVTnet technology uses this Event-response cycle to implement GUI applications that can be run across the Internet.

XVTnet consists of a server application and a thin client that runs native on the user's OS. The server application runs on a host machine on the network and is accessed by the user's thin client on a remote machine. The user's thin client receives user events and transmits them to the server application. The server application then generates responses which are transmitted back to the user's thin client for final display to the user. The server application is an application which is developed with the XVTnet toolkit. This technology provides a robust, flexible, and secure capability for providing complete large-scale applications of virtually any type across the Internet or Intranet.

This section addresses the issues specific to compiling and running XVT applications on the Internet. The information here assumes that you are familiar with developing XVT applications on your specific platform. If you are not familiar with developing and compiling applications for your platform, see the XVT Platform-Specific book for your platform.

Setting up the XVT/XVTnet Build Environment

In order for Design-generated makefiles to work, the environment variable XVT_XVTNET_DIR must be defined to be the full path to your XVT/XVTnet installation directory. Windows example:

```
C:\> set XVT_XVTNET_DIR=C:\XVTNET92\
```

Once this is set, you can build the examples and your applications.

Changes to Existing XVT Makefiles

Only a few changes need to be made to existing XVT makefiles in order to make them build with XVTnet instead of DSC:

- Instances of XVT_DSC_DIR should be changed to XVT_XVTnet_DIR.
- In the link line, library names should be changed to reflect the (different) names of the XVT/XVTnet libraries. Usually, this means adding an 'i' to the library name. See your lib directory for exact filenames for your platform.

Executing an XVT/XVTnet Application

When an application is compiled with XVTnet, it includes your application, plus an application server module. When you execute the application, it automatically runs the application server module before running your application. This module establishes a connection with the client computer, exchanges parameters, then calls your application's "main()" routine. The application server module built into your application can "serve" only your application and will only serve it up once. Once the remote user requests to exit, your application exits and cannot serve any more users. This mode is used primarily for debugging your application. (Note that UNIX users will need to run as root to do this, since port 508 is a root-restricted port under UNIX). When you are ready to put your application online, you will have to set it up with the XVT Application Server.

Setting up an XVT Application Server

The XVT Application Server is a special executable module that allows you to serve multiple users with multiple XVT Applications simultaneously. It also allows you to define exactly what applications can be executed on your server platform.

NOTE: If you are trying to make a XVTnet server visible to the Internet, and you are behind a firewall, you must work with your site administrators to find a way to make port 508 (the XVTnet protocol port) visible across that firewall. If you do not do this, users will not be able to connect to your application server using NetLink.

Windows NT Application Server

The Application Server is an executable called appserv.exe. This application is in the form of a Windows NT service. To install the service, execute the following command line:

Appserv -install

This will then enable the service and place an entry in the Control Panel/Administration Tools/Services list.

Start the service to enable it. To do this, simply run the Windows "services" applet from the Control Panel/Administrative Tools. In the list you will find "XVT Application Server Service" listed. Select that entry and press "Start." This will start-up the service and allows users to connect to your system. You must now configure which XVT Applications are allowed to be executed.

To define which applications are to be executed, you must create a file called trust.app (a sample can be found in the .\XVTNET\bin directory) in the Windows System32 directory. This is a text file with the following format:

<appname> <startup path> <commandline>

Each entry specifies an application that can be executed by a remote user. The <commandline> argument should point to an application developed with XVTnetT (although this is not enforced).

The first entry in this file should be the default application. It is always named "default" and is the application to be run if no other is specified. It is generally used to either display a message or to allow the user to select another application to run. Its entry has the following format:

default <startup path> <commandline>

After setting up your application's file, users should be able to connect with NetLink and run any of the applications you have specified. You can specify various parameters in this command line. Built-in switches include:

-d or -D	run as a daemon
-n or -N	show the name
-l or -L	enable logging
-os or -OS	use SSL
-c or -C	the SSL Certificate file name
-k or -K	the SSL Key file name
-u or -U	run as User

Unix application server

Usage:

```
appservd [-d] [-l] [-n]

-d Run as stand-alone daemon
-l  Give verbose logging information
-n Show connected names in log file
```

Under Unix the application server daemon is called "appservd". This daemon is

responsible for accepting connections from the NetLink client and executing the appropriate requested application in the trust.app file.

appservd can be run in two different modes, as a stand-alone daemon or from the inetd daemon. The difference between them is that in order for the daemon to be automatically started at reboot (in the final, production environment), it needs to be run from the inetd daemon, while stand alone mode is sufficient for the development process. To run appservd in stand-alone mode, execute it as follows:

appservd -d

NOTE: appservd must be run as root when using stand-alone mode.

To run the appservd from the inetd daemon, you must make an entry in your /etc/inetd.conf file as follows:

xvttp stream tcp nowait <user> <script_path>/<script_name> <script_name>

where:
<user> is the userid to run the applications as
<script_path> is the full path to a script to run which will run appservd
<script_name> is the name of the script to run

NOTE: An entry for xvttp must also be in the /etc/services file. The standard port number for xvttp is 508, but this may be changed as long as the NetLink clients are also pointed to that port.

The script must properly set up the environment in which to run the XVTnet applications. This includes setting environment variables for the DISPLAY as well as any others such as shared library paths. The following is a sample script:

```
#!/bin/csh
# change to the applications directory…
cd /xvtapps

# set up the environment. The application still uses X-windows, so
# the DISPLAY is still necessary.
setenv DISPLAY localhost:0.0
# LD_LIBRARY_PATH applies to Sparc platforms, DEC, and others. See your
# XVT Platform Specific Book for X/Motif for details.
setenv LD_LIBRARY_PATH /tools/X11R5/lib:/xvtlibs

# execute the appservd daemon— DO NOT REDIRECT OUTPUT.
```

./appservd

Once these three files have been modified, the inetd daemon must be restarted. Do so by sending a hang-up signal (HUP, or signal 1) to that process (or by rebooting the machine):

```
# ps -ef | grep inetd
root   382    1 0.0    Jun  13  ?? 0:00.87     /usr/sbin/inetd
root  32183992   0.0    13:17:52    ttyp0 0:00.01     grep inetd
#
```

Seeing that the inetd daemon is running (in this case) as process 382, we issue the following command (notice the very important '-' in front of the '1'!):

```
# kill -1 382
```

Your application should now be able to accept connections, and will continue to do so even after reboots.

To define which applications are to be executed, you must create a file called trust.app (a sample can be found in the .\XVTNET\bin directory) in the /etc directory. This is a text file with the following format:

<appname> <startup path> <commandline>

Each entry specifies an application that can be executed by a remote user. The <commandline> argument should point to an application developed with XVTnet (although this is not enforced).

The first entry in this file should be the default application. It is always named "default" and is the application to be run if no other is specified. It is generally used to either display a message or to allow the user to select another application to run. Its entry has the following format:

default <startup path> <commandline>

After setting up your application's file, users should be able to connect with NetLink and run any of the applications you have specified. You can specify various parameters in this command line. Built-in switches include:

-l	Give verbose logging information
-on	Enables no security (default)
-oa <password>	Enables single-application-password security
-ou	Enables user-password security
-op	Enables public/private key security.

The –oa, –ou and –op parameters require application-specific coding in the server and the NetLink customization library. See section 4.0 for more information.

Debugging

If you are having problems getting applications to connect to a machine, the following are some debugging tips.

Try getting the appservd to run in stand-alone mode first. When running, appservd will produce no output and request no input, this is normal. Just try connecting to the machine with a NetLink client.

If after executing "appservd -d" it returns immediately to the prompt, check the "appserv.log" file, it will give some information as to why it failed.

The following are some common error numbers:

13 - permission error, you must run appservd -d as root (the log file may not have been created, also due to permission problems).

48 - the port is already in use, either another appservd is running, or inetd is configured to run appservd.
49 - there is a problem with TCP/IP on this machine.
50 - the network is down.

Once the appservd is running in stand-alone mode, try using the inetd to run a script that runs appservd. If this fails, it is generally because the script is failing. This is difficult to test, because you cannot easily see what is happening in the script.

Make sure your company's firewall isn't interfering; try telneting to the xvttp port (usually port 508) on that machine, and view the output. You should NOT be able to see any recognizable text, but if you do it will be an error message from the script. The point of trying to connect to it via telnet is to make sure that the inetd daemon is connecting you to appservd properly.

If it is still failing, try executing the script while logged on as root. If no errors show up, the script is not the problem and you should look at the appserv.log file for assistance.

Try executing the appservd with a -l parameter to get more verbose logging information.

All output the application generates to the stdout and stderr FILEs will be logged to the appserv.log log file (be aware of this in a production environment— output should be minimized so as not to make the user's appserv.log file grow too rapidly.

NOTE: Always make sure that appservd is running as a user who has write permissions to the appserv.log file.

APP File Format

<site> [<application>]

XVTnet applications may be bookmarked or loaded via web pages. A XVTnet bookmark file ends with .app and contains one text line made up of two fields separated by spaces. The first is the name or IP address of the Application Server, and the second is the command line to execute. Example:

demo.xvt.com control

Linking to Your Application from the Web

In order for a .app file to be transmitted across the Internet, both the web server and the web browser need to know what the .app extension means and what MIME type to map to the extension. As long as the MIME types on both ends are the same, the actual values don't really matter. Because we are working to register the MIME type application/xvt, this is a safe value to use.

The configuration of your web server varies from platform to platform and brand to brand, but with most Unix Web servers, the config directory contains the file mime.types. The .app extension should be mapped to the application/xvt MIME type as follows:

application/xvt app

The browser also needs to know what to do with .app files. Both Netscape and Internet Explorer give you the ability to specify what happens when specific MIME-typed files come in. Configure your browser to spawn the NetLink application when it receives application/xvt files.

Once these associations are made, web pages can be constructed to make references to whatever application file you want. For example:

Click <A HREF=bankprog.app> HERE </A> to do your online banking!

When the user clicks on the word HERE, the server will recognize the .app extension and send the contents as MIME type application/xvt. The browser will see that MIME type and run NetLink with the appropriate information necessary to connect to the server/application you named in the .app file, and the user's session will begin.

XVTnet Resource Specifics

XVTnet supports the use of most XVT resources including windows, dialogs, menus, and icons. NetLink automatically downloads resources as necessary for remote display. This downloading of resources happens whenever the resource is used, with the exception of icons. Icon resources are downloaded once the first time they are used, and cached for future use. In general, the download time for resources is negligible. User-defined CURSOR resources are not supported in XVTNET.

XVTnet Optimization Issues

Almost all XVT applications will run surprisingly fast across the Internet using XVTnet. There may, however, be changes that can be made to optimize applications so that they run even faster. These changes will be portable, and may optimize even the non-Internet versions of your application.

Most XVTnet optimization issues will revolve around a very few XVT functions that require round-trip requests. A round trip request is one that requires a response be received from the client computer before processing can continue. The time it takes for data to be returned from the client computer is very long in comparison to the time it takes to simply send a command and move on.

XVTnet was developed using sophisticated caching techniques that minimize the effects of round trips in all but a few functions. Caching is performed on almost all objects in the system such that even if information must be retrieved from the client computer, it will only be acquired once and retrieved from cache thereafter.

A list of functions which are still heavily impacted by the round-trip effect are listed in Appendix B. If you are having optimization problems with your application, it is strongly recommended that you review your usage of functions that appear on this list.

The delays caused by the round-trip effect are based on the result of return error values. For example, when you create a window, the server system must wait for a response from the client to determine if the window creation was successful.

These error-checking round-trip operations are, by default, optimized away with the use of several XVTnet-specific attributes:

- ATTR_NET_OPTIMIZE_CTL_CREATES
- ATTR_NET_OPTIMIZE_FONT_CREATES
- ATTR_NET_OPTIMIZE_IMAGE_CREATES
- ATTR_NET_OPTIMIZE_LIST_ADD
- ATTR_NET_OPTIMIZE_NAV_CREATES

- ATTR_NET_OPTIMIZE_TXEDIT_CREATES
- ATTR_NET_OPTIMIZE_TX
- ATTR_NET_OPTIMIZE_WIN_CREATES

These optimization attributes are initially set to TRUE. If your application behaves strangely, it may be due to some unforeseen side effect of these attributes. Try turning them off if you observe problems. When they are turned on, the server application will have no way of determining if the object has not been created until an error message is generated. A symptom caused by this type of occurrence involves the server application appearing to have a valid object, but that object causing an invalid object error message when used. Installing an error handler that will catch such errors can easily squelch this. Usually, applications behave just fine with them on, however.

Another platform-specific attribute that will assist in optimizations is ATTR_NET_USE_COMMAND_QUEUE. This attribute determines whether or not the internal command queue will be used for command processing. The command queue causes all non-round-trip commands to be queued until either a round-trip occurs or the event finishes processing. Queuing the commands and sending them at one time significantly increases speed. Because this attribute makes debugging the application difficult, it should only be enabled for release versions.

Masking unused events can also significantly speed up XVTnet. In windows, E_MOUSE_* events are, by default, always sent. Turning them off with xvt_win_set_event_mask() will significantly reduce the number of packets (assuming you aren't using mouse events—note that they may be necessary if you use text edit controls in the window).

XVTnet also provides an optimization/debugging tool. Every XVTnet application can be executed with a '-L' parameter. This parameter causes the application to generate a logfile of every function that is called, the time (in microseconds) of when it was called, and whether or not a response from the client computer was required. This log is an invaluable tool in discovering where optimizations can be made.

The best course of action for optimizing your application for the Internet is to simply compile and run the application. In most cases no optimizations will be necessary. If there are certain places where optimizations are necessary, use the guidelines given in this section to find out what to do.

Image Processing

The XVTnet system allows for sophisticated image processing across the Internet. To accomplish this, XVTnet uses the standard XVT bitmap image manipulation functions to display bitmaps on the client computer. These functions can be used

exactly as documented in the XVT manual, but it will help the developer to know exactly how and when things happen.

XVT supports both platform-independent images and platform-specific pixmaps. Both of these are also supported under XVTnet, but it is important to understand the different ways in which they are handled. Images are platform-specific and can be directly accessed by the developer, thus they are server-side-resident. Pixmaps are platform-specific and all details about how they operate are hidden from the developer; they reside strictly on the client computer.

XVTnet is based in part on the work of the Independent JPEG Group.

Image handling

XVTnet maintains two copies of any image, one on the server and one on the client. The image on the client remains empty until it needs to be displayed. It is at this time that the image is transferred from the server to the client. The system will not transfer the image again unless it has changed. The developer may want to design his application such that the image is not re-drawn until all changes have been made.

XVTnet considers the following functions to be image-modifying functions:

- xvt_image_fill_rect
- xvt_image_get_scanline
- xvt_image_set_pixel
- xvt_image_transfer

Pixmap handling

Pixmaps always reside on the client computer. The developer can use pixmaps to determine exactly when images will be transferred. The developer can display a message stating "Downloading images…" while he loads images and draws them into pixmaps. (Notice that it is not necessary to display a dialog when downloading images as this information is displayed in the status bar.) Once the images are drawn into pixmaps, these pixmaps can be used instead of the original image. This technique will improve performance under native XVT applications, as well as under XVTnet.

Image caching

Caching techniques are used in XVTnet to increase the performance of image processing. Whenever an image must be downloaded, XVTnet checks in a special cache directory for a previously downloaded version of the image. If a previous version is found in cache, that image is automatically used and no transfer is required. The cache is managed by the remote user and thus need not be a concern of the application developer. A CRC calculation is used to detect a

previously cached image.

Image compression

XVTnet automatically compresses images using JPEG compression when it transfers them. Because JPEG is a compression technique, the remote user is allowed to select from varying degrees of quality vs. speed. This is handled automatically by XVTNET, and is therefore transparent to the programmer.

Error handling

XVTnet fully supports all of the XVT error handling facilities, including ATTR_ERRMSG_HANDLER & xvt_errmsg_push_handler/xvt_errmsg_pop_handler. It is recommended, however, that these facilities not be used when debugging your application. This is because error signals must be transferred from the client computer to the server system to be processed by your application. This signaling process may take some time and may not show up as expected when debugging the application. If all error message handling facilities are not used, the error message is generated and displayed on the client screen instantly, making debugging the application much simpler. For release versions of the software, the exact timing of the error signal is not as critical.

Printing

XVTnet supports the full range of XVT printing capabilities. All printing actions occur on the client computer, which is what the remote user would expect. The remote user will be able to configure his printers, etc. as with any XVT application. Because of their remote nature, PRINT_RCDs cannot be saved. Any attempt to load a PRINT_RCD and test with xvt_print_is_valid will fail. Because this should already be handled in cases such as a user removing a printer, this requires no change in your application.

Customizing the Client with Custom Messages

Sometimes it is necessary to perform special, custom operations on the client side that aren't GUI-related. XVTnet allows the application to send arbitrary data in both directions, from the server to the client and from the client to the server. The application must supply a function to receive the custom message. The prototype of the function is as follows:

```
void xvt_net_receive_packet( XVT_PACKET *packet, short code );
```

The function must reside in the XIMUL550.DLL library (on Unix systems, the libxvtxmul550.so library) (that library must be shipped with the client as well). The XVTnet application-side library must be shipped as well, since this contains the communications code.

Sending a Message

In order to account for the varying binary representations of information from platform to platform, the xvt_net_* functions provide a means of transmitting arbitrary packets back and forth. The XVT_PACKET data type represents the message to be sent. It must be allocated using:

```
XVT_PACKET *xvt_net_packet_create( short code );
```

"code" is an application-defined number that defines the kind of packet being created. This is useful when more than one kind of packet can be sent across. It is the application programmer's responsibility to make sure that packets are written (first in, first out) and read the same way (the same data types in the same order).

Once an XVT_PACKET has been created, it can be loaded with any arbitrary amount of data (up to 64K per packet total) using the following functions:

```
void xvt_net_add_long_param( XVT_PACKET *packet, long value );
void xvt_net_add_short_param( XVT_PACKET *packet, short value );
void xvt_net_add_string_param( XVT_PACKET *packet, const char *str );
```

All of these functions automatically compensate for big-endian/little-endian binary representation differences. One function, useful for sending whole binary files back and forth, is:

```
void xvt_net_add_void_param( XVT_PACKET *packet, void *data, size_t len );
```

xvt_net_add_void_param() does not translate any part of the data buffer; it puts the memory block into the packet exactly as-is.

Once all application-specific data has been added to the packet, it can be sent to the other end:

```
void xvt_net_send_packet(XVT_PACKET *packet);
```

xvt_net_send_packet() returns as soon as the message is sent; it does not wait for processing to finish on the other end. The message is sent with a higher priority than GUI packets-- which will queue up and wait for the custom packet to be sent.

> NOTE: Custom packets are limited to less than 64Kbytes. If larger data elements need to be sent, they should be parted into smaller segments across multiple messages. Files should be sent using the file

transmission API (See section 5.0)

Once a packet is sent, the system automatically frees the memory—do NOT refer to the XVT_PACKET pointer thereafter.

Receiving a Message

When the application's xvt_net_receive_packet() function is called because an in-bound message was received, that message packet is passed as a parameter. This packet has to be disassembled in the same order it was assembled using the following functions:

```
long xvt_net_get_long_param( XVT_PACKET *packet );
param( XVT_PACKET *packet );
char *xvt_net_get_string_param( XVT_PACKET *packet,
                    char *buffer, size_t len );
void *xvt_net_get_void_param( XVT_PACKET *packet, size_t *len );
```

No facility is provided for "rewinding" the packet buffer, so save the values as they are read out.

Once the application's callback function has returned, the XVT_PACKET is automatically freed.

XVTnet Encryption

XVTnet does not by default use encryption, but it does provide all the overrideable function callback mechanisms necessary to allow the application programmer to use the encryption mechanism of choice. XVTnet offers a number of security options:

- *No encryption/security*
- *A password is necessary to access the application*
- *Each user logs in with a username and password, optionally with DES encryption keyed on the password*
- *Public/private key encryption*

At startup, before the main() function is called, xvt_net_crypt_init() is called on both the client and the server.

```
void xvt_net_crypt_init( const char *key, XVT_KEY_TYPE type );
```

At shutdown, after the TASK window's E_DESTROY event is triggered, the xvt_net_crypt_terminate() callback function is called on both the client and the server. This function is expected to clean up any resources allocated to the encryption/decryption of XVTnet data packets.

void xvt_net_crypt_terminate(void);
Whenever the client or the server is ready to send a buffer of data, the xvt_net_crypt_calc_encrypted_size() callback function is called. This application-supplied function returns number of bytes that must be allocated for a message of size len to be encrypted. XVTnet allows the encrypted size to be larger in case there is extra information necessary to decrypt the buffer, or in case there are any extra data the application wants to send along with the message.

size_t xvt_net_crypt_calc_encrypted_size(size_t len);

Once XVTnet knows that size, it allocates a buffer large enough, and places the data to be encrypted at the beginning of that buffer, and it calls the application-supplied xvt_net_crypt_encrypt() callback function.

void xvt_net_crypt_encrypt(void *buffer, size_t len);

xvt_net_crypt_encrypt() then encrypts (in place) the data buffer. If a temporary buffer is needed for the encryption process, it is up to the application to supply it.

When data is received, the application-supplied function xvt_net_crypt_decrypt() is called. It must decrypt the buffer in place.

void xvt_net_crypt_decrypt(void *buffer, size_t len);

When initiating encryption using user validation, the server can call xvt_net_crypt_get_password(), an application-supplied function that looks up a user record by username and returns (in clear text) the password associated with that account. This is then passed to xvt_net_crypt_init() as an XVT_GENERAL_KEY, useful for instance in setting up DES keys:

char *xvt_net_crypt_get_password(char *username);

xvt_net_crypt_get_public_key() and xvt_net_crypt_get_private_key() are (client-side) application-supplied callback functions called by the client and are asked to provide the client's public and private keys when negotiating public/private key exchanges.

char *xvt_net_crypt_get_public_key(void);
char *xvt_net_crypt_get_private_key(void);

xvt_net_crypt_generate_key() is the application-supplied callback function called by the server when it asked to send the server's general key for public/private encryption.

char *xvt_net_crypt_generate_key(void);

The function must reside in the XIMUL458.DLL library (on Unix systems, the libxvtxmul458.so library) (that library must be shipped with the client as well). The XVTnet application-side library must be shipped as well, since this contains the communications code.

Various log-in scenarios

Logon – No user validation, no encryption
No negotiation is necessary. The stub libraries provided by XVT do this automatically; no application-customization is necessary.

Logon – Application validation
The application has a single password protecting its usage, and the application is encrypted (possibly using DES).

- This option is initiated by placing a "-oa<password>" parameter on the server command line.
- The application's name is sent to client.
- The client displays the application's name and prompts the user for the application's password.
- The server and the client call the application-provided function. xvt_net_crypt_init() with the password as an XVT_GENERAL_KEY.
- The server and the client exchange passwords (encrypted) and verify.

Logon – User validation
Users each have their own password to the application and the application is encrypted (possibly using DES).

- This option is initiated by placing a "-ou" parameter on the server command line.
- The client displays the username/password dialog.
- The username is sent to the server.
- The server calls the application-provided function xvt_net_crypt_get_password(), passing it the username.
- xvt_net_crypt_get_password() returns the clear-text password for that user.
- The server calls xvt_net_crypt_init() with the password as an XVT_GENERAL_KEY.
- The client calls xvt_net_crypt_init() with the password as an XVT_GENERAL_KEY.
- The server and the client exchange passwords (encrypted) and verify.

Logon – Public/Private Key encryption
The server and the client use public/private key encryption to maintain secure communications.

- This option is initiated by placing a "-op" parameter on the server command line.
- The client retrieves public/private key with xvt_net_crypt_get_public_key() and xvt_net_crypt_get_private_key().
- The client sends its public key to the server.
- The server calls xvt_net_crypt_init() with the public key as an XVT_PUBLIC_KEY.
- The client calls xvt_net_crypt_init() with the private key as an XVT_PRIVATE_KEY.
- The server generates a general key by calling xvt_net_crypt_generate_key ().
- The server sends the general encryption key (generated by server) to the client.
- The server and the client call xvt_net_crypt_init() with the general key as an XVT_GENERAL_KEY.
- The server and the client exchange general keys (encrypted) and verify.

XVTnet File transfer functions

XVTnet provides two functions for transferring entire data files back and forth between the client and the server. These functions move the files back and forth with no binary data translation or ASCII end-of-line translation—it is a complete binary copy.

Sending a file to the client

BOOLEAN xvt_net_send_file_to_client(const char *fname,
BOOLEAN wait)

This function will attempt to transfer the specified file to the client. If the wait parameter is true, this function will not return until the file has actually arrived at the client site. If the wait parameter is false, this function will return immediately, even though the file is continuing to transfer. The transfer will occur in the background allowing the user to continue interacting with the application. If the wait parameter is set to TRUE, the transfer may be faster (depending on network bandwidth and usage).

Return Value:
The return value specifies whether or not the file transfer was initiated successfully. A return value of FALSE indicates either a catastrophic failure (unlikely) or that the client user refused to accept the file.

Fetching a file from the client

```
char *xvt_net_get_file_from_client( char *fname,
    size_t max_fname, BOOLEAN wait )
```

This function will attempt to retrieve the specified file from the client. The client user may elect to send a different file instead, which will cause the frame buffer to be filled with the actual name selected by the client.

If the wait parameter is TRUE, this function will not return until the file has arrived successfully at the host site. If the wait parameter is FALSE, this function will return immediately, even thought the file has not yet arrived, the file will be transferred in the background allowing user interaction to continue.

Return Value:

The return value specifies whether or not the file transfer was initiated successfully. A return value of NULL indicates either a catastrophic failure (unlikely) or that the client user refused to accept the file, otherwise a pointer to the fname buffer is returned.

Command Line Arguments

Though not necessary for most applications, Netlink supports the following command line arguments in any order and combination:

-l or L	Enables logging to the local file "netlink.log".
-s or S socket_id	Spawns the Netlink for the socket 'socket_id'.
-r or R server_id:port_id	Enables a proxy connection through the server 'server_id' using the port 'port_id'. 'server_id' can be in the form of a qualified DNS name or static IP address.
-w or W server_id:port_id	Enables a web service connection through the server 'server_id' using the port 'port_id'. 'server_id' can be in the form of a qualified DNS name or static IP address.
-p or P	Specifies Netlink to use the port 'port_id' instead The default of the default port.
-application.app	Specifies Netlink to use the APP 'application' for default connection information. If not specified, Netlink will reference the APP file 'default.app' if available for connection information otherwise a dialog will be displayed upon launch asking for server and application.

netlink.cfg

Netlink uses the netlink.cfg to store preferences. Currently three pieces of information are stored in netlink.cfg: Graphics Quality, Cache maximum and Default IP address. Even though netlink.cfg is a text file, it is best not to edit by hand. Instead, use the Options command under the Netlink menu.

Bookmark Files and the APP File Format

XVTnet applications may be bookmarked or loaded via web pages. Bookmark files are stored in the APPS directory local to the Netlink application and allow Netlink to connect to applications specified by a name. The bookmark name(s) will appear in Netlink's Applications menu.

A XVTnet bookmark file ends with .app and contains one text line made up of two fields separated by spaces. Example:

<site> [<application>]

The first is the name or IP address of the Application Server, and the second is the command line to execute. Example:

demo.xvt.com control

Linking to Your Application from the Web

In order for a .app file to be transmitted across the Internet, both the web server and the web browser need to know what the .app extension means and what MIME type to map to the extension. As long as the MIME types on both ends are the same, the actual values don't really matter. Because we are working to register the MIME type application/xvt, this is a safe value to use.

The configuration of your web server varies from platform to platform and brand to brand, but with most Unix Web servers, the config directory contains the file mime.types. The .app extension should be mapped to the application/xvt MIME type as follows:

application/xvt app

The browser also needs to know what to do with .app files. Both Netscape and Internet Explorer give you the ability to specify what happens when specific MIME-typed files come in. Configure your browser to spawn the NetLink application when it receives application/xvt files.

Once these associations are made, web pages can be constructed to make references to whatever application file you want. For example:

Click <A HREF=bankprog.app> HERE </A> to do your online banking!

When the user clicks on the word HERE, the server will recognize the .app extension and send the contents as MIME type application/xvt. The browser will see that MIME type and run NetLink with the appropriate information necessary to connect to the server/application you named in the .app file, and the user's session will begin.

Conclusion

As can be seen in this document, the Internet as well as corporate Intranets can benefit greatly from the XVTnet technology. It adds a new tool for network developers, a tool which can work with, and complement the capabilities of the existing tools. Access to XVTnet applications can be seamlessly integrated into the net and accessed with little or no knowledge of its internal workings by users. This access will be secure and reliable as well as fast and efficient. Providence Software Solutions, Inc. provides a Cross Platform C/C++ IDE and libraries, XVT. With a long-standing reputation for addressing application portability needs with a comprehensive cross-platform application development solution that spans 12 operating systems, Providence offers both the products and services to extend your market reach and maximize your software investment. The XVTnet technology provides a completely new capability for developing Internet and Intranet solutions. The XVTnet technology provides a robust, flexible, and secure capability for providing complete large- scale applications of virtually any type across the Internet or Intranet.

A

APPENDIX A: **Server Environment**

Default root

MOTIFHOME=/usr/X11R6
XVT_DSC_DIR=/home/builder/xvtdsc580
XVT_XVTNET_DIR=/home/builder/xvtdsc580
LD_LIBRARY_PATH=/usr/lib:/usr/local/lib:/home/builder/xvtdsc580
PATH=/usr/X11R6/bin:/home/builder/xvtdsc580/bin
UIDPATH=./%U:/home/builder/xvtdsc580/bin/%U

B

APPENDIX B: **SSL certificates**

```
# create the folders to hold the certificate and the key
cd /home/builder
mkdir certs
mkdir keys

# copy the pki localhost to appserv.crt in the same location as the
localhost.crt
cp -f /etc/pki/tls/certs/localhost.crt
/etc/pki/tls/certs/appserv.crt

# copy the pki localhost to appserv.key in the same location as the
localhost.key
cp -f /etc/pki/tls/private/localhost.key
/etc/pki/tls/private/appserv.key

# go to the local certs folder and create a symbolic link to the
appserv.crt
cd /home/builder/certs
ln -s /etc/pki/tls/certs/appserv.crt appserv.crt

# go to the local keys folder and create a symbolic link to the
appserv.key
cd /home/builder/keys
ln -s /etc/pki/tls/private/appserv.key appserv.key

# restart xinetd service to link up the appserv configuration with
the new cert/key location and file names
service xinitd restart
```

C

APPENDIX C: What is xinetd?

The xinetd daemon is a TCP wrapped *super service* which controls access to a subset of popular network services including FTP, IMAP, and Telnet. It also provides service-specific configuration options for access control, enhanced logging, binding, redirection, and resource utilization control.

When a client host attempts to connect to a network service controlled by xinetd, the super service receives the request and checks for any TCP wrappers access control rules. If access is allowed, xinetd verifies that the connection is allowed under its own access rules for that service and that the service is not consuming more than its alloted amount of resources or is in breach of any defined rules. It then starts an instance of the requested service and passes control of the connection to it. Once the connection is established, xinetd does not interfere further with communication between the client host and the server.

Red Hat Enterprise Linux 4: Reference Guide

D

APPENDIX D: **Typical xinetd configuration**

```
# default: on
# description: Vend XVT DSC/DSC++ Net applications as requested from XVT Netlink client.
service xvttp
{
    # required entries for service
      disable = no
      socket_type = stream
      protocol = tcp
      wait = no
      user = root

    # select the port if different than defined in services
    port = 443

    # new environment to be passed to appserv
      env += DISPLAY=:0.0
      env += XVT_DSC_DIR=/home/builder/xvtdsc580
      env += XVT_DSI_DIR=/home/builder/xvtdsc580
      env += XVTPATH=/home/builder/xvtdsc580/print
      env += MOTIFHOME=/usr/X11R6
      env += LD_LIBRARY_PATH=/home/builder/xvtdsc580/lib
      env += UIDPATH=./%U:/home/builder/%U:/home/builder/xvtdsc580/bin/%U

    # existing environment to be passed to appserv
      passenv += LANG
      passenv += PATH

    # server and arguements
      server = /home/builder/xvtdsc580/bin/appserv
      server_args = -l -os -k /home/builder/keys/appserv.key -c /home/builder/certs/appserv.crt
}
```

E

APPENDIX E: Java Web Start Setup

The directory hierarchy required to launch a XVTnet application with Java Web Start is detailed below. The naming of the directories is important as they are referenced by different components in the process. The case of the names is also important.
The www/ folder below refer to a directory that needs to be created on the web server and made visible to the internet. It can be called whatever is appropriate for the visible folder.
The files required to successfully launch an XVTnet based application developed with XVTnet 5.8 are listed below in a directory hierarchy:

```
www/
    XVTnet.html
    xvtnet.jnlp
    images/
        XVTnet_logo.jpg
    lib/
        xvtnet.jar
        linux/
            netlinks
        win32
            netlink.exe
```

Web Server Requirements

Java Web Start was first introduced in Java version 1.3 and has been included in each subsequent release. The web server that is going to serve the HTML page must have Java 1.3 or later installed and configured. Even if the web server does not require Java for anything else, Java must be installed and put in the path.
Also the server must be configured to use the MIME type JNLP. Most server operating systems released in the last 2 or 3 years probably already has this MIME type define, but if not, it must be defined for Java Web Start to work.

1. xvtnet.html

The xvtnet.html file is the mechanism that starts the entire launch process. This file has been kept to the bare minimum to be viewed in a browser and do the necessary steps to launch a .jnlp file. Although the exact contents of the .html file can be altered and modified, the sections written in script need to remain somewhat in tack. The most important part however is the link to the .jnlp file.

This is really the central purpose of the existence of the html page.

2. xvtnet.jnlp

The xvtnet.jnlp is the configuration file for the launching of the XVTnet application via Java Web Start. There are several places that will need to be changed to match the server it is being loaded on and optional changes that can be made to alter the parameters passed to the Netlink application. An .app file is created at runtime and placed in the folder on the client. This .app file uses the information from the xvtnet.jnlp file to configure Netlink to launch without user input.
The entire xvtnet.jnlp file is included below to highlight the sections that need to be changed to fit the server environment it is being run from.

```
<?xml version="1.0" encoding="utf-8"?>
<!-- JNLP File for XVTnet Sample Application -->
<jnlp
 spec="1.0+"
 codebase="http://www.providencesoftware.com/XVTnet/"
 href="xvtnet.jnlp">
 <information>
 <title>XVTnet Sample Application</title>
 <vendor>Providence Software Solutions, Inc.</vendor>
 <homepage href="XVTnet.html"/>
 <description>XVTnet Sample Application</description>
 <description kind="short">XVTnet Sample Application</description>
 <icon href="images/xvt_logo.jpg"/>
 <icon kind="splash" href="images/xvt_logo.jpg"/>
 </information>
 <security>
   <all-permissions/>
 </security>
 <resources>
   <!--The minimum version of Java JRE required -->
   <j2se version="1.4.2"/>
   <j2se version="1.4+"/>

   <!-- The full url of the jar file location -->
   <property name="urlString" value="http://www.providencesoftware.com/XVTnet/lib/"/>

   <!-- Name of the app file for XVTnet to read. This should not need to be changed -->
   <property name="appFile" value="xvtnet.app"/>

   <!-- The fully qualified name or ip address of the machine that is running appserv -->
   <property name="appServer" value="omar.nrlmry.navy.mil"/>

   <!-- The default port number that appserv is listening on -->
   <property name="defaultPort" value="443"/>

   <!-- The default ip address of the machine that is running appserv -->
   <property name="defaultIP" value="omar.nrlmry.navy.mil"/>

   <!-- The logging level for Netlink to use. The values are "true" or "false". -->
   <property name="logging" value="true"/>

   <!-- The requirement of using SSL. Values are "true" and "false". -->
   <property name="requireSSL" value="true"/>

   <!-- The name that appserv uses from trust.app to reference this application -->
   <property name="appName" value="XVTnet_ssl"/>

   <!-- This tells app whether or not to use a proxy server. Values are "true" or "false" -->
   <property name="useProxy" value="false"/>

   <!-- This tells app the name of the proxy server. Values are "" or the proxy name -->
   <property name="proxyServer" value=""/>

   <!-- This tells app the proxy server port. Values are "" or a port number -->
   <property name="proxyPort" value=""/>

   <!-- This tells app whether or not the proxy server uses HTTP 1.1. Values are "true" or "false" -->
   <property name="proxyHTTP11" value="true"/>

   <!-- This tells app whether or not to use a web service. Values are "true" or "false" -->
   <property name="useProxyWS" value="false"/>
```

```
   <!-- This tells app the name of the web service server. Values are "" or the web service name -->
   <property name="proxyServerWS" value=""/>

   <!-- This tells app the web service port. Values are "" or a port number -->
   <property name="proxyPortWS" value=""/>

   <!-- This tells app whether or not the web service uses HTTP 1.1. Values are "true" or "false" -->
   <property name="proxyHTTP11WS" value="false"/>

   <!-- The level of graphics quality. Values are 1 = exact, 2 = accurate, 3 = fast, 4 = fastest, 1000
= grayscale and is added to the other values-->
   <property name="graphicsQuality" value="2"/>

   <!-- The size of the cache maximum in megabytes -->
   <property name="cacheMaximum" value="5"/>

   <!-- This property tells the app to keep it's dll's on the local machine to speed up application
startup. Values are "true' or "false" -->
   <property name="keepLocalFiles" value="true"/>

   <!-- This property tells the app to verify files that are sent back and forth. Values are "true' or
"false" -->
   <property name="verifyFiles" value="true"/>

   <!-- This property tells the app to keep create output.txt on the local machine to verify or debug
the execution command line. Values are "true' or "false" -->
   <property name="createOutput" value="true"/>

   <!-- This property tells the Netlink to start in minimized or regular mode. Values are "true' or
"false" -->
   <property name="startMinimized" value="true"/>

   <!-- The location of the jar file. If package is unzipped properly, this should not need to be edit-
ted -->
   <jar href="lib/xvtnet.jar"/>
 </resources>
 <application-desc main-class="com.pss.dsw.XVTnet"/>
</jnlp>
```

3. xvtnet.jar

The xvtnet.jar file contains the compiled Java binaries to respond to when the .jnlp processing requests it. The Java programs are generic in the sense that they will launch XVTnet Netlink in either Windows 32 bit or Linux 32 bit format. It refers to and uses the files in the /lib folder called /win32 and /linux. Both of the folders contain the executable version of Netlink version 5.8 from XVTNET. In order to update the binaries, the new copy simply needs to be copied into the appropriate folder in the /lib folder.

4. linux/

The linux folder contains the Linux version of XVTnet Netlink version 5.8 and the supporting shared libraries. These files can be updated with new builds simply by copying them into the linux folder.

5. win32/

The win32 folder contains the Windows 32 bit version of XVTnet Netlink version 5.8 and the supporting DLL files. These files can be updated with new builds simply by copying them into the win32 folder.

6. Summary

The application should be build with the same XVTnet version as the Netlink and supporting files loaded onto the web server. Once this directory structure is in place, the HTML file should be modified to the requirements and likings to post on

the network. When this page is brought up in a browser, the user simply clicks on the Application link to launch the Java Web Start process. Java Web Start reads the xvtnet.jnlp file and calls the XVTnet constructor function of the compiled XVTnet Java class. The .jar file classes copy over to the client the files listed in the appropriate platform folder and then executes Netlink with the parameters specified in the .jnlp file. If the xvtnet.app file is specified, Netlink will connect to the XVTnet Appserver and execute the XVTnet application. Performance is based on the speed of the internet at the time of use.

F

APPENDIX F: **Platform-Specific Attributes**

ATTR_NET_SPECIAL_TXEDIT

Enables or disables special handling for text edits. This causes the xvt_tx_process_events function to automatically occur on the client computer for events which match the given event mask. This increases text edit performance, but does not allow event filtering by your application for the events matching the mask.

Uses win argument	Yes
xvt_vobj_get_attr returns	Previously set value
xvt_vobj_set_attr effect	Enables or Disables special processing
xvt_app_create use:	Can use either before or after
Default value	EM_NONE
Argument type	EVENT_MASK

ATTR_NET_SYNC_MODE

Enables or disables synchronous event mode for the given window. In synchronous event mode the return value for each event is waited for by the client computer, thus causing each event to become a round-trip. This will slow down the application, but may help debug timing problems. The following events are always asynchronous: E_UPDATE, E_MOUSE_MOVE, and E_TIMER. The E_DESTROY event is always synchronous.

Uses win argument	Yes
xvt_vobj_get_attr returns	Previously set value
xvt_vobj_set_attr effect	Enables or Disables special processing
xvt_app_create use:	Can use either before or after
Default value	FALSE
Argument type	XVTNET_SYNC_MODE

ATTR_NET_REMOTE_EXEC

Executes a command line on the client computer. Note that the client computer must be able to understand and execute the command line, or it will display an error message to the user. The developer may use the ATTR_NET_REMOTE_OS

attribute to determine the client operating system when formatting the command line.

When the user executes this command, he will see a dialog which reads "About to execute the following command line:" A checkbox reading "Don't ask again about this app" can be checked when the user trusts the application, and the dialog will no longer appear.

Uses win argument	No
xvt_vobj_get_attr returns	Illegal
xvt_vobj_set_attr effect	Causes execution of command line
xvt_app_create use:	Can use either before or after
Default value	NONE
Argument type	STR

ATTR_NET_OPTIMIZE_CTL_CREATES
ATTR_NET_OPTIMIZE_FONT_CREATES
ATTR_NET_OPTIMIZE_IMAGE_CREATES
ATTR_NET_OPTIMIZE_LIST_ADD
ATTR_NET_OPTIMIZE_NAV_CREATES
ATTR_NET_OPTIMIZE_TXEDIT_CREATES
ATTR_NET_OPTIMIZE_TX
ATTR_NET_OPTIMIZE_WIN_CREATES

These attributes enable or disable the optimization of error checking return values on creates statements. If optimization is enabled (TRUE), return values will not be checked for errors. See section 2.4 for more information.

Uses win argument	No
xvt_vobj_get_attr returns	Previously set value
xvt_vobj_set_attr effect	Enables or Disables special processing
xvt_app_create use:	Can use either before or after
Default value	TRUE
Argument type	BOOLEAN

ATTR_NET_REMOTE_OS

Retrieves the XVT_OS macro value for the client computer. This allows the programmer to specially design his application for a given client OS.

Uses win argument	No
xvt_vobj_get_attr returns	XVT_OS macro value
xvt_vobj_set_attr effect	Illegal
xvt_app_create use:	Can use either before or after
Default value	NONE

Argument type	XVT_OS value

ATTR_NET_REMOTE_WS

Retrieves the XVTWS macro value for the client computer. This allows the programmer to specially design his application for a given e windowing system. This also determines which e platform-specific attributes will work properly.

Uses win argument	No
xvt_vobj_get_attr returns	XVTWS macro value
xvt_vobj_set_attr effect	Illegal
xvt_app_create use:	Can use either before or after
Default value	NONE
Argument type	XVTWS value

ATTR_NET_REMOTE_VERSION

Retrieves the version number information of the remote NetLink client software. This attribute can be used to enable or disable usage of features supported by newer clients that are not supported by older clients. The version number information is encoded into the long word return value as follows:
First 8 bits Major version #
Second 8 bits Minor version #
Final 16 bits Revision #

Uses win argument	No
xvt_vobj_get_attr returns	Client version number
xvt_vobj_set_attr effect	Illegal
xvt_app_create use:	Can use either before or after
Default value	NONE
Argument type	long

ATTR_NET_STATUS_STATUS_SET

Sets the status value in the status bar on the client computer. The status value is generally used by XVTnet and NetLink to display status information as processing occurs. This attribute can be used to allow the developer to display his own messages. The value for this string can be a maximum of 10 characters; any other characters will be truncated. This is the second, smaller field in the statusbar.

Uses win argument	No
xvt_vobj_get_attr returns	Illegal

xvt_vobj_set_attr effect	Sets value into status bar
xvt_app_create use:	Can use either before or after
Default value	NONE
Argument type	STR

ATTR_NET_STATUS_TEXT_SET

Sets the text value in the statusbar on the client computer. The text value is generally used by XVTnet and NetLink to display general information as processing occurs. This attribute can be used to allow the developer to display his own messages. The value for this string can be any length, but not all of the string may be viewed depending on the client screen resolution and current window size. This is the field in the statusbar.

Uses win argument	No
xvt_vobj_get_attr returns	Illegal
xvt_vobj_set_attr effect	Sets value into statusbar
xvt_app_create use:	Can use either before or after
Default value	NONE
Argument type	

ATTR_NET_CLIENT_SYNC

Causes the server and client applications to synchronize once. All command and event queues will be flushed and any errors that may occur will have occurred. This attribute allows the developer to gain some control over application timing when queuing is involved.

Uses win argument	No
xvt_vobj_get_attr returns	Illegal
xvt_vobj_set_attr effect	Causes synchronization
xvt_app_create use:	Can use either before or after
Default value	NONE
Argument type	ignored

ATTR_NET_USE_LOCAL_TIMERS

Causes XVTnet to use local timers to generate timer events instead of using the client timer. By default, the client computer will generate events based on timers created. Due to network delays, however, these events may not arrive in a controllable interval. Using local timers allows for controllable intervals, but may cause problems with queue flooding if the interval is too small. For this reason, local timers should not be used to generate graphical commands.

Uses win argument	No
xvt_vobj_get_attr returns	Previously set value
xvt_vobj_set_attr effect	Enables or Disables special processing
xvt_app_create use:	Can use either before or after
Default value	FALSE
Argument type	BOOLEAN

ATTR_NET_USE_COMMAND_QUEUE

Causes XVTnet to queue commands before sending them to the client system. This can significantly speed complex drawing operations, but makes debugging difficult. It is recommended that this attribute be enabled for release versions of the software only.

Uses win argument	No
xvt_vobj_get_attr returns	Previously set value
xvt_vobj_set_attr effect	Enables or Disables special processing
xvt_app_create use:	Can use either before or after
Default value	FALSE
Argument type	

XVTnet Client platform-specific attributes

The following attributes are specific to the client platform. The attributes will not generate errors when the client system does not support them, they will simply be ignored in the case of set_attr and return 0 in the case of get_attr. Each of these attributes is prefixed with the text ATTR_NET_R, which is then followed by the platform-specific attribute text. See the appropriate platform-specific book for more detailed information regarding these attributes.

Windows/Windows 95/Presentation Manager (OS/2)/Windows NT:
ATTR_NET_RWIN_DELAY_FOCUS_EVENTS
ATTR_NET_RWIN_MENU_CACHE_COUNT_MAX
ATTR_NET_RWIN_PM_SPECIAL_1ST_DOC
ATTR_NET_RWIN_R3_DIALOG_PLACEMENT
ATTR_NET_RWIN_USE_PCL_RECTS

F

APPENDIX F: Round-Trip Commands and Attributes

The following is a list of round-trip commands. If you are experiencing optimization issues with your XVTnet application, you should look at your usage of these commands.

Always round-trip commands

xvt_app_escape
xvt_app_get_file
xvt_app_get_files_count
xvt_app_process_pending_events
xvt_cb_close
xvt_cb_get_data
xvt_cb_has_format
xvt_cb_open
xvt_ctl_get_colors
xvt_ctl_get_native_colors
xvt_ctl_get_text_sel
xvt_dm_post_ask
xvt_dm_post_error
xvt_dm_post_fatal_exit
xvt_dm_post_font_sel
xvt_dm_post_message
xvt_dm_post_note
xvt_dm_post_page_setup
xvt_dm_post_string_prompt
xvt_dm_post_warning
xvt_dwin_get_font_app_data
xvt_dwin_get_font_family
xvt_dwin_get_font_family_mapped
xvt_dwin_get_font_native_desc
xvt_dwin_get_font_size
xvt_dwin_get_font_size_mapped
xvt_dwin_get_font_style
xvt_dwin_get_font_style_mapped
xvt_errmsg_pop_handler
xvt_fmap_get_families
xvt_fmap_get_family_sizes
xvt_fmap_get_family_styles
xvt_fmap_get_familysize_styles
xvt_fmap_get_familystyle_sizes
xvt_font_deserialize
xvt_font_get_family_mapped
xvt_font_get_size
xvt_font_get_size_mapped
xvt_font_get_style
xvt_font_get_style_mapped
xvt_font_get_win
xvt_font_has_native_desc
xvt_font_is_mapped
xvt_font_is_print
xvt_font_is_scalable
xvt_font_is_valid
xvt_font_map
xvt_font_serialize
xvt_list_count_all
xvt_list_count_sel
xvt_list_get_all
xvt_list_get_elt
xvt_list_get_first_sel
xvt_list_get_sel
xvt_list_get_sel_index
xvt_list_is_sel
xvt_list_rem
xvt_list_set_sel
xvt_menu_get_tree
xvt_palet_add_colors
xvt_palet_add_colors_from_image
xvt_palet_create
xvt_palet_default
xvt_palet_get_colors
xvt_palet_get_ncolors
xvt_palet_get_size
xvt_palet_get_tolerance
xvt_palet_get_type

xvt_pict_create
xvt_print_close_page
xvt_print_create
xvt_print_create_win
xvt_print_get_next_band
xvt_print_is_valid
xvt_print_open
xvt_print_open_page
xvt_print_start_thread
xvt_scr_get_focus_topwin
xvt_scr_get_focus_vobj
xvt_scr_list_wins
xvt_tx_destroy
xvt_tx_get_active
xvt_tx_get_attr
xvt_tx_get_limit
xvt_tx_get_line (A_GET only)
xvt_tx_get_margin
xvt_tx_get_next_tx
xvt_tx_get_num_chars
xvt_tx_get_num_lines
xvt_tx_get_num_par_lines
xvt_tx_get_num_pars
xvt_tx_get_origin
xvt_tx_get_sel
xvt_tx_get_tabstop
xvt_tx_get_view
xvt_tx_is_scroll_update
xvt_tx_rem_par
xvt_tx_set_par
xvt_vobj_get_attr (depends on attr)
xvt_vobj_get_flags
xvt_vobj_get_outer_rect
xvt_vobj_get_title
xvt_vobj_is_focusable
xvt_vobj_is_valid
xvt_vobj_translate_points
xvt_win_get_ctl_colors
xvt_win_get_cursor
xvt_win_get_event_mask
xvt_win_has_menu
xvt_win_list_wins
xvt_win_process_modal
XVTNET_win_set_handler_external
XVTNET_remote_sync

Round-trip when creation optimization is disabled
xvt_ctl_get_font
xvt_dlg_create_def
xvt_dlg_create_res
xvt_event_get_font
xvt_font_create
xvt_image_create
xvt_menu_get_font_sel
xvt_nav_create
xvt_pmap_create
xvt_tx_create
xvt_tx_create_def
xvt_tx_get_font
xvt_win_create
xvt_win_create_def
xvt_win_create_res
xvt_win_get_ctl
xvt_dwin_get_font
xvt_win_get_tx

Round-trip once, cached afterwards
xvt_ctl_get_id
xvt_dwin_draw_image
xvt_dwin_get_draw_ctools
xvt_dwin_get_font
xvt_dwin_get_font_metrics
xvt_dwin_get_text_width
xvt_font_get_family
xvt_font_get_metrics
xvt_font_get_native_desc
xvt_nav_list_wins
xvt_tx_get_rect
xvt_tx_get_win
xvt_vobj_get_client_rect
xvt_vobj_get_data
xvt_vobj_get_palet
xvt_vobj_get_parent_win
xvt_vobj_get_type

One-time round-trip
xvt_app_get_default_ctools

Round-trip unless ATTR_NET_USE_LOCAL_TIMERS == TRUE
xvt_timer_create

Round-trip unless ATTR_NET_OPTIMIZE_TX == TRUE
xvt_tx_add_par xvt_tx_append
xvt_tx_clear

Round-trip unless ATTR_NET_OPTIMIZE_LIST_ADD == TRUE
xvt_list

