XVTnet Professional

Administrator's Guide

xvtnet

Version 11.1

Table of Contents

Introduction

Administrator's Guide

Welcome to XVTnet, XVT's development solutions for the Internet. This guide allows the user to navigate through the XVTnet application. It is to be used in addition to existing XVT manuals for your XVT Development Platform.

XVTnet allows the user to compile your XVT applications so that they execute across the Internet with the same ease of use that is featured on other XVT platform ports. It will assist in setting up and navigating the XVTnet package, along with re-compiling your application for the Internet or your intranet.

Server Environment

Unix/Linux
Default root

MOTIFHOME=/usr/X11R6
XVT_DSC_DIR=/home/builder/xvtdsc111
XVT_XVTNET_DIR=/home/builder/xvtdsc111
LD_LIBRARY_PATH=/usr/lib:/usr/local/lib:/home/builder/xvtdsc111
PATH=/usr/X11R6/bin:/home/builder/xvtdsc111/bin
UIDPATH=./%U:/home/builder/xvtdsc111/bin/%U

Application Setup

Executing an XVT/XVTnet Application
When an application is compiled with XVTnet, it includes your application along with an application server module. When the application is executed, it automatically runs the application server module before running your application. This server module:

- Establishes a connection with the client computer.
- Exchanges parameters.
- Calls your application's "main()" routine.

The application server module built into your application can S*erve* 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 an XVTnet server visible to the Internet, and you are behind a firewall, you must work with your site administrators to establish a port that matches the configuration pot. If a specific port is not configured, port 508 (the XVTnet protocol port) serves as the default. If you do not do this, users will not be able to connect to your application server using NetLink.

Windows Application Server

The Application Server is an executable called appserv.exe. This application is in the form of a Windows 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 S*ervices* applet from the Control Panel/Administrative Tools. In the list you will find *XVT Application Server Service* listed. Select that entry and press S*tart*. 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 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:

-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
-p or -P	port to use

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 xinetd daemon. The difference between the two is in order for the daemon to automatically start at reboot (in the final, production environment), it needs to be run from the xinetd 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.

What is xinetd?

The xinetd daemon is a TCP wrapped S*uper 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 allotted 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.

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.

xinetd

```
# 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/xvtdsc111
        env += XVT_DSI_DIR=/home/builder/xvtdsc111
        env += XVTPATH=/home/builder/xvtdsc111/print
        env += MOTIFHOME=/usr/X11R6
        env += LD_LIBRARY_PATH=/home/builder/xvtdsc111/lib
        env += UIDPATH=./%U:/home/builder/%U:/home/builder/xvtdsc111/bin/%U

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

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

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

```
# ps -ef | grep xinetd
root       382       1       0.0       Jun  13   ??          0:00.87             /usr/sbin/xinetd
root     32183     992       0.0       13:17:52             0:00.01             grep xinetd
#
```

Seeing that the xinetd daemon is running(in this case 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.

The xinetd service has the ability to be stopped, started and restarted, through:

Service xinetd stop
Service xinetd start
Service xinetd restart

NetLink 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.

Java Web Start Setup

The directory hierarchy required to launch an 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 11.1 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 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 few years already have this MIME type define, but if not, it must be defined for Java Webstart to work.

xvtnet.html
The xvtnet.html file is the mechanism that starts the entire launch process. This file has been pared down for portability so as to be viewed in a browser and to execute 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.

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 listed below to highlight the sections that need to be changed to fit the server environment.

```
<?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://[Server Name]/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=[Server Name]"/>

  <!-- 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 val     ues-->
  <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 edited -->
  <jar href="lib/xvtnet.jar"/>
  </resources>
  <application-desc main-class="com.pss.dsw.XVTnet"/>
  </jnlp>
```

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 11.1 from XVTNET. In order to update the binaries, the new copy simply needs to be copied into the appropriate folder in the /lib folder.

linux/

The linux folder contains the Linux version of XVTnet NetLink version 11.1 and the supporting shared libraries. These files can be updated with new builds simply by copying them into the linux folder.

win32/

The win32 folder contains the Windows 32 bit version of XVTnet NetLink version 11.1 and the supporting DLL files. These files can be updated with new builds simply by copying them into the win32 folder.

Applications

APP File Format

<site> [<application>]

XVTnet applications may be bookmarked or loaded via Web pages. An 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. Most browsers 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.

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 xinetd restart
```

Various Log-in Scenarios

Log-on – *No User Validation, No Encryption*

No negotiation is necessary. The stub libraries provided by XVT do this automatically; no application-customization is necessary.

Log-on – *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.

Log-on – *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.

Log-on – *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.