ProvideX

VERSION 7

Client-Server Reference

WindX, JavX, Application Server, AutoUpdater

sage software

8920 Woodbine Ave. Suite 400, Markham, Ontario, Canada L3R 9W9

The capabilities, system requirements and/or compatibility with third-party products described herein are subject to change without notice. Refer to the Sage Software Canada Ltd. website www.pvx.com for current information.

Publication Release: V7

Contents

Preface

The ***ProvideX Client-Server Reference*** provides details on the installation, configuration, and operation of ProvideX thin-clients and hosting facilities. Although this volume is intended for application developers, it does not try to explain how to design or create a client-server program – instead, it documents the tools that are available for building client-server implementations in ProvideX.

Refer to the *ProvideX Language Reference* for descriptions of ProvideX keywords and concepts: directives, functions, system variables, mnemonics, parameters, specialty files, reserved words and system limitations. Rather than reproduce existing material, references to other publications may be supplied where applicable.

Using this Documentation

This documentation is designed for both viewing and printing via **Acrobat® Reader**. Click **Help > Reader Guide** on the menu bar to learn how to display, copy, search, and print PDF documentation. While there are several ways to navigate the contents of a PDF-based document, the following methods are highly effective, and are consistent with other documentation distributed by ProvideX:

Bookmarks

The list of bookmarks, displayed on the *left side* of the Acrobat window, serves as a hyperlinked *table of contents*. Bookmarks are displayed in a hierarchy where subordinate headings appear indented below main headings. When subordinates are hidden or *collapsed*, a plus sign (in Windows) or triangle (in Mac OS) will appear next to the main heading. Simply click on the *plus sign* or *triangle* to display all collapsed headings.

Cross-References

Blue hyperlinks appear throughout this document wherever one section cross-references another. They also appear in the form of hyperlists; such as the *Table of Contents*, the *Index*, and the linked tables placed at the beginning of some chapters.

The mouse pointer looks like an index finger when it is positioned over a linked cross-reference — simply click to activate the link. For example, "**Using this Documentation**" is hyperlinked back to the beginning of this section.

PDF Navigation Tips: The *chapter name* at the top left corner of the page head can be used as a hyperlink to the beginning of the chapter. Use the page-up/down/back/forward buttons ▼ ▲ ◀ ▶ to move one page at a time. Use the ***Back*** button ◀◀ to jump to the *previous view.*

Conventions

The following syntax items are used to illustrate what is needed in the format of a program statement in ProvideX.

...	Dots indicate the continuation of a list of elements.
[ ]	Square brackets enclose optional elements in the format. For example, in **ABS**(*num*[,**ERR=***stmtref*]) you can omit the **ERR=***stmtref* portion of the statement as in **ABS**(**X-Y**). (Exceptions are noted for individual commands where the brackets are "real"; i.e., part of the syntax.)
{ }	Curly brackets enclose a list of elements in syntax formats where it is mandatory to select one item. For example, with {**YES** | **NO**}, you must select either **YES** or **NO**. In descriptions in this manual, they denote {bitmap / icon} buttons. (Exceptions are noted for individual commands where the brackets are "real"; i.e., part of the syntax.)
|	Vertical bars (pipes) separate a choices; e.g., {**YES** | **NO**}.
chan	Channel or logical file number. It must be an integer between 0 and 127. This identifies the channel to which your directive applies; e.g., `CLOSE (14)`. • Channel zero (0) is the console. If you omit the channel, the system defaults to 0 (the console). • Channels 1 to 63 are commonly used for local files. • Channels 64 to 127 are used for global files. • ***Exception***: In extended file mode (**'XF'** system parameter) the channels range from 0–32767 for local files, and 32768–65000 for global files.
col,ln, wth,ht	Position/coordinates. Numeric expressions. Column and line coordinates for top left corner, width in number of columns, and height in number of lines.
ctrlopt, fileopt	Optional syntax elements — three-character codes followed by an equals sign and argument (`DOM=3250`). For a list of typical options, see *Input/Output and Control Options* in the *Language Reference*.
stmtref	Statement reference. This can be either the line label or line number of a statement in the current program. Line numbers must be in the range of 0 to 64999.

If your given line number does not exist, ProvideX goes to the statement with the next higher line number. For example, if line 1000 doesn't exist and 1010 is the next line number, then for `GOTO 1000` ProvideX will go to 1010 and proceed with execution from there. See also *Labels/Logical Statement References* in the *Language Reference*.

Exception: ProvideX verifies the existence of an IOList and *stmtref* for **IOL=***stmtref*. It does not proceed to the next higher statement number.

varlist List of comma-separated variables. Typically, a mix of string and/or numeric variables is acceptable; e.g., `DEPT,ITEM,DESC$...` (See individual directives for restrictions.)

Numeric Expressions and Variables

When a syntax format in this manual includes a ***numeric*** variable like *chan*, *index* or *num* (lowercase), you can normally substitute a ***numeric expression*** consisting of variables, literals, functions, and operators. For instance, your value could be something like `HFN` or `4` or `NUM(A1$)*3-2`. (`NUM` in upper case is the function.) When numeric variables are used in numeric expressions, subscripts are allowed; e.g., `COST[4]`.

Note: Exceptions and valid values are stated when there are restrictions on the use of numeric or string expressions in a format (e.g., where only variable names are allowed).

String Expressions and Variables

When a syntax format in this manual includes a ***string*** variable like *prog_name$* or *title$*, you can normally substitute a ***string expression*** consisting of variables, literals, functions, and operators; e.g., `PRINT "Printing "+REPORT$`. When string variables are used in string expressions, subscripts and substrings are allowed; e.g., `CUSTOMER$(15,4)`.

Chapter Outlines

Chapter 1. **Introduction,** *p.9*. Provides an overview of ProvideX client-server functionality explains differences between available thin-client products.

Chapter 2. **Client-Server Functionality,** *p.17*. Discusses general client-server concepts and explains standard behaviour that applies to all ProvideX thin-clients.

Chapter 3. **WindX - Windows Thin-Client,** *p.31*. Documents the installation, configuration and operation of the ProvideX Windows thin-client.

Chapter 4. **JavX - Java-Based Thin-Client,** *p.43*. Documents the installation, configuration and operation of the different Java-based thin-client products for PC platforms and for portable devices.

Chapter 5. **Application Server,** *p.81*. Documents the installation, configuration and operation of the ProvideX Application Server facility.

Chapter 6. **AutoUpdater,** *p.133*. Documents the installation, configuration and operation of the ProvideX utility for automatically updating client applications.

1 Introduction

ProvideX client-server technologies deliver multilateral solutions for displaying and interacting with your ProvideX applications on a server.

Thin-Clients

WindX (Windows-based) and **JavX** (Java-based) thin-clients can be used to optimize software development and to minimize the resources required to install and/or service a large user application. With a few changes to your source code, you can leverage the ProvideX thin-client architecture to maintain heavy processing and data storage on a secure central server while delivering a custom user interface to a wide range of client platforms, from web browsers to mobile/handheld devices.

The thin-client functionality is fully integrated into ProvideX and the server handles the heavy processing. The net result is that:

- Applications can be easily adapted for true distributed processing capability.
- Less software needs to be installed and configured.
- Bandwidth is optimized because only the application GUI travels across the connection.
- More concurrent users can be served because of the reduced network traffic.
- Data integrity is maintained by keeping data processing on the server.

Application Server

Install the ProvideX **Application Server** to extend the usability and security of your server-based applications with secure access control, user and application launch control, central administration, and access tracking features.

AutoUpdater

The **AutoUpdater** utility is included with ProvideX to provide a means for automatically updating client installations of ProvideX and WindX. It can be configured to check for and install critical patches/upgrades on all client workstations whenever they are connected to the server.

Thin-Client Products

Historically, programmers have had two thin-clients to choose from in the ProvideX suite of development tools: ***WindX***, which is written in ProvideX, and ***JavX***, which is written in Java.

Fundamentally, ***WindX*** is designed to provide a Windows graphical environment to an otherwise non-graphical (non-Windows) application that is based on a remote computer. Along with a standard Windows user environment, WindX provides for local file access and processing, and it can be configured to include a client-side auto-update capability.

When ***JavX*** was first introduced, it was designed to be the Java version of WindX. JavX provides a ***flexible alternative*** to the Windows-only thin-client because it takes full advantage of Java's portability and platform independence. However, it is not possible to recreate the entire Microsoft Windows environment within the Java framework, so some WindX functionality is not available under JavX.

JavX uses Java 2 GUI components to replicate the complex graphical features available in ProvideX. But unlike WindX, a Java-based thin-client is able to run on any machine that has the appropriate **Java Runtime Environment** (JRE). Even if the client machine does not have a Java JRE, it is a free download that can be installed just prior to running JavX.

For specific differences between the two ProvideX thin-clients, refer to the section **JavX vs. WindX**, *p.26*.

Choosing the Right Solution

With the introduction of the first Windows-based thin-client, WindX delivered the ability to harness the power of ProvideX at both ends of a client-server implementation. When JavX was initially released, it promised similar functionality, but without the dependence on a local copy of ProvideX. This trusted and familiar thin-client technology is now experiencing a dramatic evolution beyond the simple desktop platform.

There are currently many millions of Java-enabled mobile phones, PDAs, and other portable devices currently in use all over the world. Consequently, the ProvideX thin-client architecture has since been adapted to tap into this exploding marketplace. Depending on the target platform, developers now have four ProvideX thin-clients to choose from: **WindX**, **JavX SE**, **JavX AE**, and **JavX LE**.

But with more choices available, developers need to be sure that they are building the appropriate client-server implementation to serve their user's needs. The differences between the various thin-client products are outlined below, and are illustrated in the **Thin-Client Requirements Table**, *p.11*.

Accessibility vs. Expressiveness

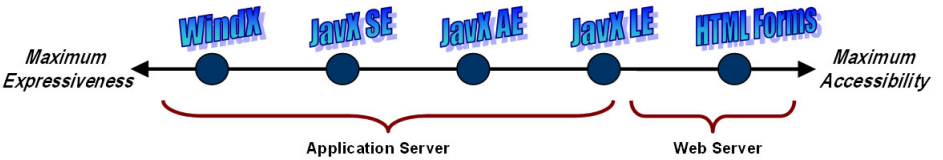
Each client type has its own set of advantages in functionality and flexibility. So, which is the right solution for your client-server application? Choosing the appropriate thin-client requires an analysis of the trade-offs between ***accessibility*** and ***expressiveness***. The specific level of functionality offered by each client is illustrated in the **Thin-Client Requirements Table** below.

The richest full-featured GUI applications require the closest ties to a specific platform, which places strict limits on their accessibility. However, when applications are designed to run on a wider variety of platforms, GUI functionality has to be abstracted away from the client operating system, which sacrifices expressiveness.

Therefore, the best ProvideX solution for an expressive GUI might be to employ WindX to deliver a powerful, but Windows-specific, user environment. Conversely, you could employ the ProvideX WebServer and use standard HTML forms (completely outside of WindX or JavX) to deliver a universal, but rudimentary, user environment.

Thin-Client Requirements Table

The full range of ProvideX client-server options can be illustrated as follows:



WindX Full-featured thin-client that takes advantage of the local OS to deliver a rich graphical environment (along with auto-update capability) from any remote ProvideX host system to any MS Windows client.

A WindX client is most effective when:

- end-users are accessing the application from a conventional office workstation
- long complex tasks are to be performed on a regular basis
- application developers determine the client platform (not end-users).

This is fully documented under **WindX - Windows Thin-Client**, *p.31*.

JavX SE The *JavX Swing Edition* is a Java-based thin-client with similar functionality to WindX that enables ProvideX applications to run on any number of client platforms (Windows, Linux, Apple) that support the Java 2 Standard Edition (J2SE) runtime. With JavX SE, the web browser is promoted to a universal ProvideX client—users can navigate to a JavX SE-enabled web page which uses a Java applet to interface with the server application. However, JavX SE implementations offer slightly less functionality than WindX and have limited access to the local machine.

A JavX SE client is most effective when:

- end-users are accessing the application from outside the office
- long complex tasks are to be performed occasionally
- end-users require more choice in platforms.

JavX SE is fully documented in the section **JavX for PC Platforms**, *p.60*.

JavX AE The *JavX AWT Edition* provides a simpler GUI designed specifically for handheld devices that support the Java 2 Micro Edition (J2ME) runtime.

A JavX AE client is most effective when:

- end-users are accessing the application from outside the office
- more general tasks are to be performed occasionally
- typical platform is a WinCE PDA.

JavX AE functionality is upwardly-compatible with JavX SE and is fully documented in the section **JavX for Portable Devices**, *p.73*.

JavX LE The *JavX Light Edition* represents a limited non-GUI version of JavX that is intended primarily for fixed-purpose industrial or consumer products.

A JavX LE client is most effective when:

- end-users may be performing a few simple tasks using an interactive device/appliance.

JavX LE functionality is upwardly-compatible with JavX AE/SE and is fully documented in the section **JavX for PC Platforms**, *p.60*.

HTML Forms A ProvideX WebServer/HTML implementation allows for a basic web user environment that has no access to the local machine.

An HTML Forms implementation is an effective client when:

- end-users require brief access to fill in a few form fields that are presented using a standard web page.

Refer to the ProvideX WebServer documentation for information on using HTML with ProvideX to create a web application.

Important: It is therefore clear that the appropriate configuration depends entirely on the needs of your end-users. Avoid trading off a fully-functional GUI in order to gain universal accessibility that your clients won't need ... *and vice versa.*

Background and Prerequisites

Client-server implementations can be as complex (or as simple) as the applications they are supporting; however, a few basic requirements must be met in order to establish a successful client-server connection in ProvideX:

- WindX or JavX thin-client installed/configured on a client machine or device.
- ProvideX application running on a host system that is set up for remote multi-user access via ***NTHost/*NTSlave** or the **Application Server**.
- A network connection between the two.

The sections below provide an brief overview of the prerequisites necessary for setting up and running a client-server application using WindX or JavX.

Licensing ProvideX Client-Server Facilities

Both WindX and JavX are available as ***freely-distributable plug-ins***; however, each client installation must be connected to ProvideX on a server that maintains a multi-user Professional or eCommerce license. If the server is not licensed for plug-in access, server file/directory permissions are incorrect, or a ProvideX session cannot be established within 2 minutes of startup, then the plug-in will terminate automatically.

The thin-client connection initially uses one user slot from the server-side ProvideX user license. Subsequent sessions that are spawned programmatically from the same workstation will not require additional user license slots.

While the plug-ins are free to use and distribute, both WindX and JavX are subject to Sage Software license agreements once they are downloaded from **www.pvx.com**.

WindX Standalone

WindX may also be installed as a ***standalone*** product that is capable of interacting with any ProvideX application on any server. This implementation requires an individual license (*serial number, user count, expiry date* and *activation key*) per installed client.

Note: WindX Standalone and Plug-in downloads are for installation on Windows 95/98/ME/NT4/2000/XP client machines only. JavX may be installed on any OS platform that supports the Java 2 Standard Edition (J2SE) runtime.

JavX LE by Request

Unlike other JavX editions, the *JavX Light Edition* is designed for a wide variety of fixed-purpose/embedded devices that *may or may not* have a standardized method for installing software. Because of this, the JavX LE installation **JAR** archive file is currently not available for download. Developers who are interested in running JavX for a specific device implementation are welcome to contact Sage Software Canada Ltd. to request a copy of JavX LE.

Application Server

The ProvideX Application Server can be purchased as a separate add-on, but it is also distributed as part the eCommerce product bundle.

AutoUpdater

This facility is included with the base system activation; however, the AutoUpdater is only available with ProvideX Version 7 (or higher), and currently it can only be deployed as part of a ProvideX client-server implementation that includes WindX (Version 4.11 or higher).

Connection Requirements

In a generic implementation of client-server architecture, one program or computer acts as a *server*, providing services to other programs or computers, called *clients*. While some computers can be linked directly via fixed connection, most client-server systems communicate over a *network*, using technologies that connect through and between many different computers.

WindX allows a variety of communication methods including serial (RS232), Telnet, and **TCP/IP** connections; see **WindX - Windows Thin-Client**, *p.31*. However, current ProvideX thin-client implementations (involving WindX and JavX) are more likely to be handled via *direct* TCP/IP using one of the following:

- *NTHost/*NTSlave facilities supplied with the ProvideX base system.
- *ProvideX Application Server*, full-featured client-server connection system.

For an overview of the procedures required to configure a client-server environment in ProvideX, see **Hosting Facilities**, *p.17*.

TCP/IP Overview

Transmission Control Protocol/Internet Protocol (TCP/IP) is the primary communication language for governing the services that everyone uses over the *Internet*, including file transfer, electronic mail, and remote logon. It is also the communications protocol for most private (intranet or extranet) networks. Several other protocols are used by Internet applications in conjunction with TCP/IP, including Hypertext Transfer Protocol (HTTP), File Transfer Protocol (FTP), and Simple Mail Transfer Protocol (SMTP).

However, among the suite of Internet protocols, it is the TCP and IP components (or layers) that provide the transport and network address functions for all Internet communication.

Transmission Control Protocol (TCP) is the layer responsible for successful delivery of data between client and server. It converts messages/files into small ***packets*** that are transmitted, then reassembled into the original data at the destination computer. It detects errors or lost packets and will trigger retransmission until all the data is received intact. TCP support is built into ProvideX.

Internet Protocol (IP) is the layer responsible for identifying the computer or device at each end of the link. It uses a four-byte destination address (IP number) that is usually stated as *nnn.nnn.nnn.nnn*, where the value of *nnn* is between 0 and 255.

When a TCP connection is established between client and server, the client requests a connection to a specific server by giving its IP address and a service identifier (port/socket number). On the host computer, an application establishes itself by identifying its service number which typically ranges from 1 to 65535. Typically, numbers below 2000 are reserved for specific Internet applications, while numbers above are available for general use.

Network Implementation

Today, almost all commercial operating systems include and install the TCP/IP layers by default. TCP/IP is included in most UNIX and Linux distributions as well as with Mac OS X and Microsoft Windows and Windows Server. In many cases the network settings are configured automatically or via a simple setup procedure once the user identifies and connects to the network.

Sockets and Ports

As described earlier, each computer on a TCP/IP-based network is identified by a single IP address. At each IP address, multiple simultaneous connections are a made possible through the use of software mechanisms called ***sockets***.

When a program needs to share (read/write) data with another program over a network (local or internet), it gets the operating system to create a socket to establish the connection between them. Depending on the implementation, the socket may be either ***active*** or ***passive***:

- If it is active, the socket is connected to a remote active socket via an open connection and there is a constant exchange of information. If the connection is closed, the sockets at both endpoints immediately cease to exist.
- If it is passive, the socket is not connected but is in a state where it is always waiting for an incoming connection. If a connection is made, the passive socket spawns a new active socket.

All sockets are identified by a ***port*** number. In a typical TCP/IP implementation, there would be multiple sockets using the same port. Each port number may represent a single ***passive socket*** (listening for any incoming connections) as well as multiple ***active sockets*** (corresponding to many open connections) on the port.

An apartment building is often used as an analogy for describing how sockets maintain connections between two machines in TCP/IP addressing. The IP address is like the apartment building's street address. The port number is like the apartment number within the building.

2
Client-Server Functionality

Client-server functionality has been integrated into ProvideX to create a seamless environment in which to develop and run applications. On the ***server side***, a ProvideX host program (either *NTHost or the Application Server) launches a copy of ProvideX and monitors a **TCP/IP** socket, "listening" for incoming requests from clients.

Note: While the Windows thin-client, WindX, allows connections via **Telnet/Serial Configuration**, direct TCP/IP is more common in conventional client-server architecture.

On the ***client side,*** JavX or WindX opens the client end of the same socket the host is listening to, and passes requests through to the server.

This chapter provides an overview of general client-server concepts in ProvideX, discussing server-side implementation (hosting facilities), standard client-side functionality, and the primary differences between the two ProvideX thin-client products: *WindX and JavX*.

Hosting Facilities

In a direct TCP/IP client-server environment, the JavX and WindX thin-clients must be set up for access on the server-side using either *NTHost/*NTSlave or the *ProvideX Application Server*.

*NTHost/*NTSlave are the default programs supplied with the ProvideX base system for establishing a simple TCP/IP connection; however, we do not recommend this hosting choice if you require more control over access and security. The *ProvideX Application Server* provides an enhanced configurable solution for creating and maintaining your client-server implementations. More importantly, this add-on includes a suite of administration tools that allow you to protect your sensitive data from exposure to hostile network environments; i.e., *the Internet*.

This section describes how to set up the basic client-server processes. If you need a safer environment for your TCP /IP connections, see *Chapter 5.* **Application Server** for complete configuration details.

Using *NTHost/*NTSlave

The default client-server configuration in ProvideX requires two distinct processes. On the host computer, the program *NTHost is run to monitor incoming requests from client PCs and to initiate new processes to service these requests. On the client computer system, the program *NTSlave begins the initial connection to the host by requesting a new session to be started.

If the sessions can be started, then *NTHost sends back the TCP/IP port number of the new process. *NTSlave uses this number to connect to this process and passes control to JavX or WindX to handle all of the terminal and GUI interactions.

In order for the initial connection between *NTHost and *NTSlave to be established, *NTSlave must know the port number that *NTHost is monitoring. By default, this is port number 10000 but it can be changed in the command line used to start both programs. As *NTHost initiates new processes, it assigns new port numbers in increments starting with the port number one higher than the *NTHost port assignment. By default, up to a maximum of 1000 sequential port numbers can be assigned.

*NTHost *on a Windows Platform*

To run *NTHost in Windows, set up a shortcut with the following command line:

`c:\pvx\pvxwin32.exe *nthost`(assuming ProvideX is installed in `c:\pvx`)

There are three optional arguments that can be supplied: the port number that *NTHost is to monitor (and the clients to connect to), the maximum port number to use for connections, and the ability to request using the TCP/IP ***Keepalive*** functionality. These can be specified in the command line as program arguments; e.g.,

```
c:\pvx\pvxwin32.exe *nthost -arg 10000 11999 -k
```

These arguments direct *NTHost to monitor socket 10000 and assign spawned sessions to port numbers 10001 through 11999. If no arguments are specified, then *NTHost monitors port 10000 and assigns port numbers 10001 through 10999. Each session employs the TCP/IP ***Keepalive*** functionality to periodically send commands to a connection to ensure that it is still active.

To run *NTHost, simply launch this shortcut or place it in the Windows Startup folder to have it run each time the machine is restarted.

*NTHost can also be run as ***Windows Service***, which is a special process designed to start and run automatically under Windows (NT/2000/XP) directly from start up. Different methods for setting up service entry points to ProvideX are outlined under *Windows Services* in the *ProvideX Installation Manual*.

*NTHost *on a UNIX/Linux Platform*

Setting up *NTHost on a UNIX (or Linux) platform is a little more complicated and involves changing the system process `'inittab'`.

Important: The following procedure should only be performed by developers who are knowledgeable in UNIX.

To run *NTHost as a UNIX service, add a line using the following syntax to the system `'inittab'` file (usually in the directory `/etc/`):

`pvx1:234:respawn:/pvx/pvx \*nthost -arg` *port uid umask*

Where:

`pvx1:234:respawn`	`'inittab'` parameters that cause UNIX to spawn *NTHost at all normal execution levels and keep it running.
`/pvx/pvx \*nthost`	Location of ProvideX executable, *NTHost program name.
port	Port number that *NTHost is going to monitor.
uid	UNIX user ID used by the spawned process.
umask	umask setting for all spawned tasks used when creating files.

There are three optional parameters that may be specified at the end of the command line. They are the maximum port number to assign to the spawned sessions, `-v` which indicates that debugging information is to be sent to the system console, and `-k` which activates the TCP/IP ***Keepalive*** functionality; e.g.,

```
pvx1:234:respawn:/pvx/pvx \*nthost -arg 10000 11999 elvis -v -k
```

Note: The \ in front of `*nthost` causes UNIX to process the `*` *asterisk* literally rather than as a *wildcard* character.

When *NTHost launches new sessions, it does so via an `'su'` command. This ensures that applications do not run as root (which maintains system security). Since the host program will be spawned from the `'inittab'`, it will be running with full *root* privileges. This means that the host is capable of running any application, and can utilize `'su'` to launch an application under another user ID.

Special care must be taken when using `'su'`. The user ID chosen must be valid and the password must not have expired. If applications refuse to launch, then it is likely that the password for the user ID has expired, or there is an administrative lock on the account. The user account should not launch anything automatically via its profile since the new session picks up the characteristics of the user id being used.

When running under IBM AIX, characteristics such as umask in the user's account setup may need to be modified as they will override the command line arguments.

Another potential problem is the number of processes a single user may run. Since all applications are spawned as this user, the kernel may need to be re-tuned to increase the number of processes per user.

*NTSlave *for Launching Thin-Clients*

The *NTSlave process runs on the client side and is used to request a new session, establish a connection to the server, and then launch WindX. The following *NTSlave client command line syntax applies to WindX only and is fully documented in the section **Launching WindX**, *p.37*:

```
c:\pvx\pvxwin32.exe *ntslave -arg server prog port
```

Unlike WindX, JavX is not dependant on a local copy of ProvideX. Therefore the *NTSlave functionality is built directly into JavX. The JavX connection from the client to *NTHost on the server is explained in the section JavX **Installation and Configuration**, *p.44*.

Secure Socket Layer (SSL)

The Secure Sockets Layer (SSL) is a commonly-used protocol for managing the security of a message transmission on the Internet. Support for TCP/IP-level SSL-encryption is available for both WindX and JavX when configured for use in the *ProvideX Application Server*. SSL is not supported under *NTHost.

Encryption of communications between a JavX client and the ProvideX host was not possible until Sun added support for SSL to version 1.4 of the Java plug-in. For JavX to have SSL support, the client machine must have the Java 1.4 (or later) plug-in installed.

Standard Thin-Client Behaviour

Once one of the **Hosting Facilities** is configured and running on the server, an installed/configured thin-client should be able to establish communication with the server-based ProvideX system. As with ProvideX itself, the launch procedures (command line syntax) for **WindX** and **JavX** can be customized to define system settings, operating environment, and various application requirements. For more information on the thin-client startup, refer to **Launching WindX**, *p.37* or **Launching JavX**, *p.46*.

ProvideX Session Interface

The default syntax for launching the thin-client environment produces a typical ProvideX session window for entering and executing programming instructions. This interface opens with the standard sequence of serial number and copyright notices followed by a command prompt.

Note: ProvideX prompts differ slightly when they appear within a thin-client environment. The main prompt changes from a **>** to a **}** and the un-saved program prompt**:** *colon* changes to a **;** *semi-colon*.

At this point, the server-based ProvideX interposes itself between the user and the (client) operating system and permits all work to be controlled via the remote (server) operating system.

Detecting a Thin-Client

Once ProvideX on the server recognizes the terminal as a ProvideX client station, it changes internal settings that allow graphical requests to be routed to the client automatically. All graphical directives and functions invoked by the application are tokenized to be sent to the client.

WindX and JavX are recognized by ProvideX on the server when the terminal type is set to either `winterm` or `ansi`. These are the only two terminal types that are recognized as potential client stations.

The type `winterm` uniquely identifies the terminal as a ProvideX client. However, most UNIX systems will not recognize this terminal type, which means that no other UNIX application can use it. To resolve this issue, the ProvideX `ansi` device driver sends a unique escape sequence to the terminal during its initialization process. If the terminal is a ProvideX client, then a special response will be generated. However, if the request times out, then ProvideX carries on under the assumption that the terminal is really an `ansi` device.

Client-Side Functionality

The client-side functionality in ProvideX is designed to minimize network traffic and improve system performance:

- Standard **Mnemonics** are transmitted "as is".
- **Terminal Input/Function keys** are sent unchanged.
- **Graphical Control Requests** are tokenized.
- **Turbo Mode** allows the client to receive and process requests locally without the need to acknowledge each transmission from the host.

Mnemonics

WindX and JavX respond directly to the internal form of all mnemonics. Therefore, unlike conventional terminals, no translation table is required. Mnemonics, such as **'CS'**, are transmitted as `$1B$+"CS"` and screen position commands, such as **@(1,2)**, are sent as `$1B$+"@2"+CHR(1)+CHR(2)`. Long-form mnemonics, such as **'WINDOW'** and **'DROP'**, are sent in their native form as well.

Local Processing

A lot of the functionality regarding screen refreshing is processed locally. This means that less data needs to be sent to the client station to handle functions such as window manipulation.

On a normal terminal when a window is created, the characters that make up the window border are transmitted from the host to the terminal. Additionally, when a Window is removed, then all of the data that was hidden behind the window must be redrawn. These two functions are processed locally which reduces the traffic on the line and improves system performance.

In JavX, the **'PICTURE'** mnemonic takes either a URL or path on the local machine (e.g., `www.pvx.com/some.jpg` or `c:\ some.jpg`). If JavX is running as an applet, it cannot access local files. All image sources (button, picture etc.) must be URLs. If not, JavX adds the source of the page on which it is embedded to the beginning of the path; e.g., `some.jpg` becomes `www.thesource/some.jpg`.

Terminal Input/Function keys

All input entered at the ProvideX client keyboard is sent directly to the host. Function keys and CTL events generated by graphical controls are included in the data stream as well.

Graphical Control Requests

ProvideX tokenizes all graphical directives and references, then forwards them to the client for processing. These tokenized commands are then passed to the local copy of ProvideX for processing. Additionally, access to the control attributes (e.g., `BackColour$, Height, Enabled`) is tokenized as well and forwarded to the client for processing.

Note: In many instances, it is better to use directives rather than attributes when interfacing with controls under JavX. Each reference to an attribute involves a packet being sent to the PC. For example, setting grid values would involve setting three (3) attributes but only one directive.

Turbo Mode

During normal operation, each tokenized message sent by the host to the client requires an acknowledgment. While this process guarantees that the application and client are synchronized fully, it can slow down overall transmission speeds.

Turbo Mode allows JavX and WindX to receive and process many requests locally without the need to acknowledge each transmission from the host. To enable Turbo Mode, set the system parameter '**TU**' on the host system.

While in Turbo mode, acknowledgments are not sent by WindX for directives and functions that do not return a value, for example, a write command for a graphical control. If an error occurs, it is reported locally or can be ignored depending on the configuration option.

Note: While Turbo mode does improve performance, it may cause some unexpected results from code that relies on error detection; e.g., relying on an error branch to detect a bad value when issuing a **WRITE** command to a control will not work. To avoid this situation, either change the application logic or turn off the **'TU'** system parameter.

Language Enhancements for Thin-Clients

For the most part, thin-client functionality is handled seamlessly within ProvideX. However, there are some enhancements to the language that are specific to thin-clients. These are listed as follows:

MSE *System Variable*

Byte `22` returns the version number of the thin-client (WindX or JavX). A value of `$FF$` means that the client is not connected via a thin-client. Byte `32` returns either `W` for WindX or `J` for JavX or `$00$` for no thin-client. See determining WindX or JavX and use of [WDX].

TCB(88) *System Function*

This function returns the version code of the thin-client (WindX or JavX), or zero for no thin-client connection.

FIN( ) *System Function*

The **FIN( )** function can be used, when specifying channel 0 (zero), to retrieve information specific to JavX.

`FIN(0,"IsApplet")`	Returns: `"0"`=Application or no thin-client connected `"1"`=Un-Signed applet `"2"`=Signed applet.
`FIN(0,"GetClientOS")`	Returns a string of client OS and OS type; e.g., `"Mac OS X" "Windows 98".`
`FIN(0,"GetLookAndFeel")`	Returns the current look-and-feel GUI used on the client. Possible values are `"windows"`, `"mac"`, `"motif"`, and `"metal"`.
`FIN(0,"GetParam` *parm_name*`")`	This form of the function may be used when JavX is run as an applet. It returns the values of parameters that are within the tags on the HTML page that JavX was launched with; e.g., `webserver$=FIN(0,"GetParam server")` This returns the data associated with the parameter named `server` within the tags on the page that launched JavX. If the parameter name specified does not exist on the web page, then `Error #65: Window element does not exist or already exists` is returned. For details, see **HTML Reference**, *p.63*.
`FIN(0,"GetVMVendor")`	Returns the JRE vendor.
`FIN(0,"GetVMVersion")`	Returns the JRE version number 1.4.1.
`FIN(0,"GetVMName")`	Returns the JRE name.

'FONT' *Mnemonic.*

The **'FONT'** mnemonic has been enhanced (for JavX) to allow the client's look-and-feel to be change programmatically:

PRINT 'FONT'("*operator look&feel***"),**

Where:

operator	A ~ *tilde* sets the look-and-feel for any new windows or dialogues subsequently created. It does not set the look-and-feel for the current Window or Dialogue. (This mode is recommended.) A ^ *caret* sets the look-and-feel for previous, current, or subsequently created windows or dialogues.

(This mode is not recommended. Different look-and-feel's have different drawing characteristics that can cause improper updates when changing the look-and-feel of windows, dialogues, and objects within after they have been drawn.)

look&feel Possible values are `metal`, `windows`, `mac` and `motif`; e.g.,

```
PRINT 'FONT'("~metal")
```

This would change the look-and-feel to `metal` for any subsequent windows, dialogues, and objects.

The typefaces available to the **'FONT'** mnemonic are limited to only those fonts that are directly supported by Java. We recommend that you stay within the standard Java font list, otherwise you may get unpredictable results. The following platform-independent font names are supported by all Java applets: ***Serif***, ***SansSerif***, ***Monospaced***, ***Dialog*** and ***DialogInput***.

Also, note that Java does not necessarily display the font names as specified. Instead, Java maps the logical font names to a physical font that exists on the client workstation. For example, *SansSerif* is mapped to the proportional font *Arial* on a Windows system. *Monospaced* maps to the fixed-width font *Courier* on a Windows system.

Note: You cannot programmatically control exactly which font is used on the client workstation. Therefore, since fonts vary from system to system, you should test the look-and-feel of your ProvideX program on various operating systems. This will help ensure that you provide a consistent look-and-feel to your GUI screens.

'OPTION' *Mnemonic.*

The **'OPTION'** mnemonic has been enhanced (specifically for JavX) to allow the following:

- Set the client's look-and-feel to be change programmatically:
  ```
  PRINT 'OPTION("LookAndFeel","os default"),
  ```
- Set the `mapFonts` flag:
  ```
  PRINT 'OPTION("mapFont","true"),
  ```

  If `mapFonts=true`, then JavX will map physical font names to Java logical font names. For example, the physical font name "MS Sans Serif" would be mapped to the logical Java font name `"SansSerif"`. Logical font names map automatically to the appropriate physical font on any platform where the JAR is installed. Therefore, `"SansSerif"` would then be mapped back to "MS Sans Serif" if it were installed on a Windows platform.
- Set the `graphicfontname`:
  ```
  PRINT 'OPTION("graphicfontname","Courier"),
  ```

  WindX (ProvideX) retrieves the default graphical font from the Windows OS. JavX has no way of retrieving the OS default graphical font, so historically, JavX has

used the text plane font as the default graphical font. This font can be changed programmatically for each window using the **'GF'** mnemonic. Applications that do not use **'GF'**, but instead rely on an INI file or the OS default, may look slightly different in JavX. The `graphicfontname` parameter allows developers to specify the default graphical font for all windows.

- Set the `graphicfontsize`

```
PRINT 'OPTION("graphicfontsize","14"),
```

`"graphicfontsize"` used in conjunction with `"graphicfontname"`, allows developers to specify a default graphical font name and size. The default graphical font size is 13 points.

JavX vs. WindX

WindX runs in conjunction with a copy of ProvideX for Windows, so WindX can issue virtually any command of which ProvideX is capable. On the other hand, JavX uses a set of Java equivalents to match ProvideX language features – and while JavX permits ProvideX GUI applications to be run on a wider range of platforms, it cannot replicate some Windows-specific functionality. However, JavX is deployed without the need for a local copy of ProvideX.

While most ProvideX applications designed for WindX will work with JavX interchangeably, there are a few differences to be noted. Each client type (WindX, JavX SE, JavX AE, or JavX LE) has additional restrictions and/or enhancements. Refer to *Chapter 3.* **WindX - Windows Thin-Client** and *Chapter 4.* **JavX - Java-Based Thin-Client** for functionality that is specific to each of the client platforms.

The following sections provide an overview of the primary differences.

Available in WindX, but not JavX ...

There are no JavX equivalents for the following features that are currently available under WindX:

1. ***Calling a ProvideX program across a JavX connection.*** (**CALL "[WDX]...**). This functionality is unique to WindX and is fully documented under **[WDX] Remote Capabilities**, *p.40*. For the most part, the **[WDX]** tag is ignored by JavX. However, there are some exceptions that have been specially programmed into JavX, see **Thin-Client Utility**, *p.28*.
2. ***Local File I/O***. JavX only supports local Serial file access; JavX cannot access local Keyed or Index files.
3. ***Printing.*** It is not possible to open ***WINPRT*** or ***WINDEV*** to a JavX client. See **Printer Support**, *p.52*, for displaying and printing reports on a JavX client.

4. ***Parallel or Serial Ports.*** It is not possible to access a printer port (LPT) or serial port (COM) on the workstation at this time.
5. ***Bitmaps.*** JavX supports JPG and GIF formats, but ***not BMPs***. For more information, see **Image Support**, *p.53*.
6. Format specifications within multi-lines (**FMT=**) are not yet available.
7. Cross-line and crosshatch pattern fills using the **'FILL'** mnemonic are not supported; however they have been substituted with a gradient fill (explained in the next section).

If your ProvideX application requires any specific attributes or features that are currently not available in JavX, please send an email to our support department, **support@pvx.com** and inform us of your situation. This will help us prioritize the enhancements for future releases of JavX.

Note: Specific JavX features are listed and described under **Common Functionality and Limitations**, *p.48*. When the ProvideX server attempts to use an unsupported feature, it generates an `Error #98: Feature not yet implemented`.

Available in JavX, but not WindX ...

Along with the advantages of *platform independence* and the *flexibility* to run from within or through a *web browser*, the JavX design offers several features that are not available with WindX including:

1. **SYSTEM_HELP** directive. As an ***application***, JavX executes the **SYSTEM_HELP** command by passing the command line to the OS for execution, just as ProvideX itself would. When JavX is running as an ***applet*** there is no command line to pass the **SYSTEM_HELP** command to. This means that you cannot start applications, (i.e., *notepad* or *calc*) automatically via the **SYSTEM_HELP** directive.

   However, if the **SYSTEM_HELP** command begins with `HTTP://` , `HTTPS://`, or `FTP://`, then JavX traps the command internally and passes the command to the browser for execution.

   The command may have a suffix to control the "target", which is how the browser should deal with the URL reference. Suffixes are specified by a ~ (tilde) followed by either an HTML target reference or a named frame reference if working within a frames page. If no suffix is specified, then `_blank` is automatically appended and used. Valid values for suffixes are:

`_blank`	Browser is to open the URL in a new browser window.
`_parent`	Browser is to load the document in the parent frame or parent window.
`_self`	Browser is to load the new document in the same frame or window that JavX is in.
`_top`	Browser is to load the new document in the current browser window, thereby removing all frames.

A specific frame name targets a named frame window within a frameset.

Example:

```
SYSTEM_HELP "http://www.pvx.com/~_blank"
```

This tells JavX to have the browser start a new browser window, and have that window link to the ProvideX website.

The **SYSTEM_HELP** command can be used to create complex web pages or dynamic updates to frames within a web page. It may also be used to automatically begin a download of a file to the user's desktop via FTP://, and display the SaveAs or Open selections for the user.

Note: The most common printing method for JavX applications utilizes **SYSTEM_HELP** to launch a new browser session where the URL points to a PDF that is generated on the web server. See **Printer Support**, *p.52*.

2. '**PICTURE**' mnemonic can take a reference to a URL in place of the image file name.

Example:

```
'PICTURE'(0,0,400,100,"http://www.louvre.fr/img/charte/collec/peint/joconde.jpg")
```

Thin-Client Utility

The ***windx.utl** utility program provides several functions that help simplify the development of applications using WindX and JavX thin-clients; e.g.,

```
CALL "[WDX]*windx.utl;Get_LWD",Station_dir$
CALL "*windx.utl;Spawn",cmdline$,inifile$,appfid$
```

The functions supplied by this utility are listed and described below:

`CALL "[WDX]*WindX.utl;Get_Addr",x$`

Returns the IP address of the client system.

`CALL "[WDX]*WindX.utl;Get_ARG",x,x$`

WindX only. Returns the command line argument number specified by X in X$.

`CALL "[WDX]*WindX.utl;Get_LPG",x$`

WindX only. Returns the LPG (Lead Program Name) system value for the WindX session.

`CALL "[WDX]*WindX.utl;Get_LWD",x$`

WindX only. Returns the local current disk directory (Last Working Directory) for WindX session.

`CALL "*WindX.utl;Spawn",x$,c$,f$`

No** [WDX] **prefix required. Spawns a new session of ProvideX on the host and an associated session on the client system. See **Launching Multiple Sessions**, *p.29*.

`CALL "*WindX.utl;Spawn_Nohup",x$,c$,f$`

No `[WDX]` ***prefix required.*** Same as `Spawn` but will detach the session from the main user task. See **Launching Multiple Sessions**, *p.29*.

`CALL "[WDX]*WindX.utl;Get_WindX",x$`

WindX only. Returns the absolute pathname of the WindX program.

`CALL "[WDX]*WindX.utl;Get_NewPort",x`

Returns the port number of an unused TCP/IP port on the WindX station.

`CALL "[WDX]*WindX.utl;Get_TCB",x`

Returns the value of the TCB function task specified by `X` in `X`; i.e., `X=TCB(X)`.

`CALL "[WDX]*WindX.utl;Get_Val",x$,y$`

Evaluates/returns value of string expression *x$* in *y$*; i.e., *y$*`=EVS(`*x$*`)`.

`CALL "[WDX]*WindX.utl;Get_Num",x$,Y`

Evaluates/returns value of numeric expression *x$* in *y* (i.e., *y*`=EVN(`*x$*`)`.

`CALL "[WDX]*Windx.utl;get_file_box",path$,dir$`

Emulates a local call to **GET_FILE_BOX** directive.

`CALL "[WDX]*Windx.utl;get_file_box_read",path$,dir$`

Emulates a local call to **GET_FILE_BOX READ** directive.

`CALL "[WDX]*Windx.utl;get_file_box_write",path$,dir$`

Emulates a local call to **GET_FILE_BOX WRITE** directive.

Launching Multiple Sessions

The Thin-Client Utility can also be used to spawn a new session of ProvideX on the host, and an associated session on the client. Because this syntax is performed locally, calls to `*Windx.utl;spawn` and `*Windx.utl;Spawn_Nohup` do not require the `[WDX]` prefix. The syntax and functionality are slightly different when used with WindX or JavX. The differences are outlined below:

WindX Syntax

`CALL "*WindX.utl;Spawn",x$,c$,f$`

Where:

x$	command line parameters to be used on the host.
c$	pathname of INI file to be used on the client PC.
f$	specifies the value of FID(0) for the session.

The following syntax is similar to `;spawn` but it detaches the session from the main user task so that if the main task terminates, the spawned task continues executing: `CALL "*WindX.utl;spawn_nohup"`*,x$,c$,f$*.

JavX Syntax

```
CALL "*WindX.utl;Spawn",x$,c$,f$
```

Where:

x$	specifies the program to run on the host
c$	specifies the window location
f$	specifies the value of FID(0) for the session.

By default, if the main session terminates, the spawned session terminates. When *c$* is `"#embed"`, the new JavX session appears ***in*** the web browser; otherwise, JavX creates a new window ***on top*** of the browser. When *c$* is `"#float"`, the new session will float in a new dialogue above the existing browser window.

When spawning a new session of JavX in this way, the new session from the same client workstation does not use any additional ProvideX user slots from the server's ProvideX license. For more information, see `onapplet=true` described under **HTML Parameters**, *p.65*.

The following syntax is similar to `;spawn` but it detaches the session from the main user task so that if the main task terminates, the spawned task continues executing: `CALL "*WindX.utl;spawn_nohup",x$,c$,f$`.

3

WindX - Windows Thin-Client

The WindX thin-client makes it possible to distribute a feature-rich, graphical user interface to a Windows client from any server-based ProvideX application, even if the host system does not support that type of interface. It can be deployed as a stand-alone product or as a freely distributable Plug-In running on Windows 9x-2000/Me/NT4/Server 2003/XP/Vista.

For most implementations, WindX would be launched within a start sequence using a Windows application shortcut – from that point, the session is completely transparent to the user. Since WindX is actually a ProvideX program that runs on the client system, some GUI functionality and file access can be configured for local handling. Less (graphical) data needs to be carried between the WindX station and the server, which means faster processing for handling many of the functions required for window manipulation.

WindX is the most full-featured product in the ProvideX thin-client suite. It can issue virtually any command of which ProvideX is capable. For a discussion on the full range of client-server options in ProvideX, refer to the section **Choosing the Right Solution**, *p.10*.

This chapter documents the installation, configuration and functionality of the WindX thin-client.

Topics

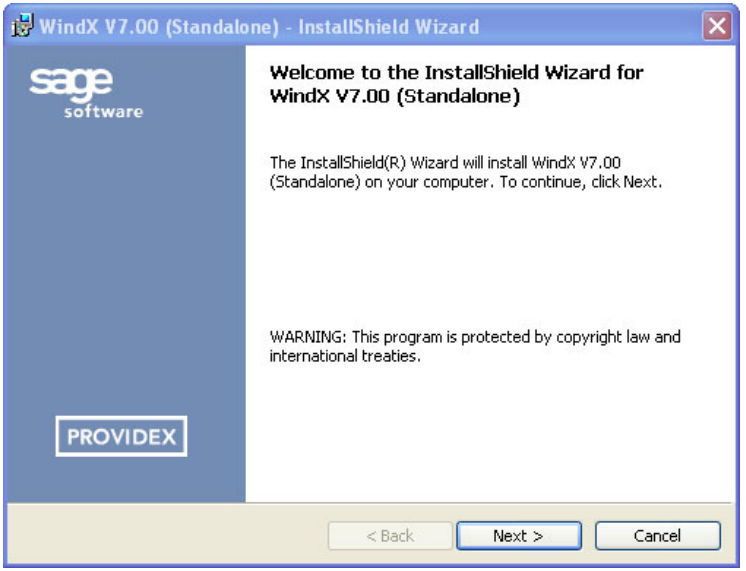
Installation and Configuration

Both the standalone and plug-in versions of WindX can be obtained from your dealer/distributor or downloaded from the ProvideX website, **www.pvx.com**. Once you download or copy the installation program to your client computer, follow these steps to install and activate WindX:

1. If possible, remain connected to the Internet. The installation process may include some options to download additional components.
2. Double-click on the installation program that was downloaded to your computer to begin the installation process. The installation program launches an ***InstallShield Wizard*** and immediately checks for any existing versions of WindX.

   If an older version of WindX is detected, you will be given the option to upgrade/overwrite your existing version, or to install the new version in a different location. If your current version of WindX is identical to the new install, you will be given the option to modify, repair, or remove the existing components.

   The wizard takes you through a series of dialogue boxes:



3. Follow the wizard instructions and click **Next >** to complete each step. The final step installs the software onto your hard disk and displays a progress bar to indicate the current installation status. *This process may take several minutes.*
4. The ***InstallShield Wizard*** will be completed when all components are copied to disk. If you downloaded the ***Plug-in***, WindX is fully installed at this point and will *skip further steps* in the installation process.

However, if you downloaded a ***Standalone*** version of WindX, the ***InstallShield Wizard*** automatically invokes the ProvideX Activation utility:

- If this is a ***new*** installation, a Demo Mode activation is applied automatically.
- If this is a ***maintenance update*** for an existing copy of WindX, the activation is transferred automatically from the original version.
- If this a ***purchased*** upgrade, you must establish a ***new*** activation for WindX.

The activation for WindX Standalone is performed in the same manner as the ProvideX Windows Activation. The necessary activation information will be issued to you when you purchase a product package from Sage Software Canada Ltd. or your authorized dealer/distributor.

For detailed activation instructions refer to procedures explained in the *ProvideX Installation Manual*.

Connection Methods

WindX may be configured to communicate with ProvideX running on Windows, UNIX, Linux, or Apple Mac OS X servers using either direct **TCP/IP**, Telnet, or serial port connections. Once the host system (UNIX or Windows) is set up for network access, no special software, apart from ProvideX on the host, is required to use the WindX client. Connections may comprise any combination of:

- Local Area Network (LAN)
- Wide Area Network (WAN)
- Serial, Point-to-Point Protocol (PPP), Virtual Private Networking (VPN)
- Communications security using industry-standard SSL encryption.

Most conventional ProvideX client-server implementations are handled via direct **TCP/IP**, using ProvideX's *Nthost or the Application Server on the host system.

Direct TCP/IP Connections

The *ProvideX Application Server* (or *NTHost/*NTSlave) **Hosting Facilities** can be used to provide direct TCP/IP communications for WindX. The client processes for a WindX direct TCP/IP configuration include:

***CLIENT** Client-side software used to establish a connection to the ProvideX Application Server running on the host machine. See *Chapter 5.* **Application Server** for complete documentation.

***NTSlave** Basic process for connecting WindX to the host system. The default configuration is explained under **Using *NTHost/*NTSlave**, *p.18*.

As mentioned earlier, WindX is launched automatically if ***CLIENT** or ***NTSlave** are used to connect with the host. The configuration is defined within the command line (Target) syntax. For more information, see **Launching WindX**, *p.37*.

Telnet/Serial Configuration

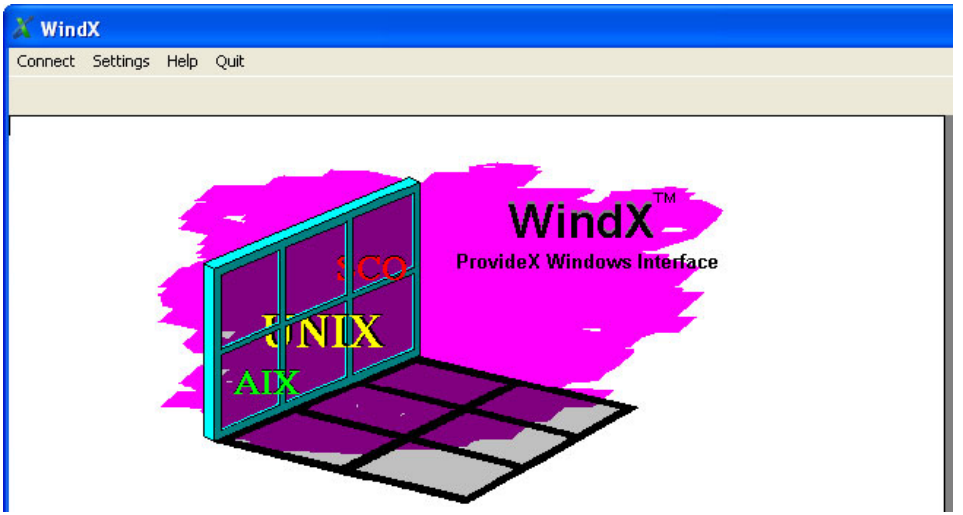
A default console is available for setting up and accessing Telnet and Serial COM port connections. The console can be invoked from within ProvideX on the client; e.g.,

```
-> run "windx/windx
```

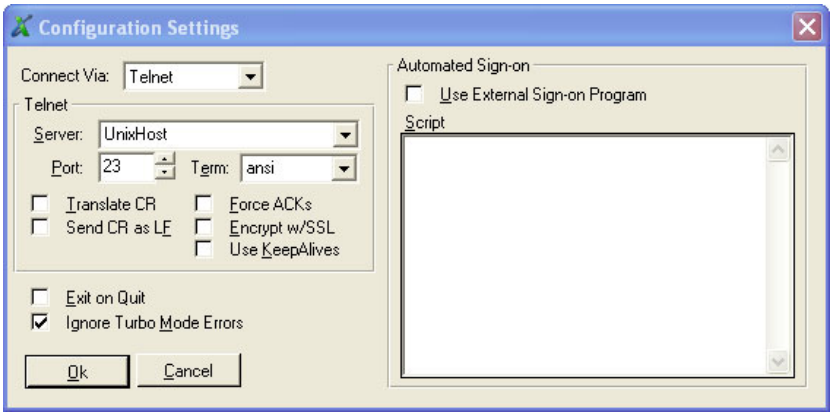
The WindX install provides a Windows shortcut with command line (Target) syntax, similar to the following:

```
"c:\pvx\pvxwin32.exe" windx
```

The splash screen for the WindX Telnet/Serial console appears as follows:



Click on Settings in the menubar at the top of the console to open the configuration screen. The configuration settings allow you to choose the connection type (Serial COM port or Telnet) and set different operational parameters:



Options are saved in the default file `windx.cfg` or in the file specified in **ARG(1)**, if set. This is a ProvideX Serial file containing one record with a **SEP** between each field. Different configuration files can be used as required. The following syntax saves the configuration options to the file `my.cfg`:

```
"C:\pvx\pvxwin32.exe" windx -arg my.cfg
```

When selecting a COM port, the user must specify the port speed and enable the hardware flow control, if necessary.

Connecting Via Telnet

If you choose to connect via Telnet, then you must enter the name and/or IP address of the targeted host computer. Several other configuration options are available:

Port	Normally, port 23 is used with Telnet communication; however, this can be changed if necessary.
Terminal Type	The ability to specify a Terminal Type when connecting to a UNIX server was introduced in version 5.01.
Translate CR / Send CR as LF	The Translate CR and Send CR as LF options handle minor variations in the protocol that have been found to exist between vendor implementations.
Force ACKs	The option to Force ACKs (Forced Communications Acknowledgments) may help improve the performance of WindX when in Telnet mode. It periodically adds a few bytes of data to the packets sent by WindX in an attempt to force the Telnet daemon (`telnetd`) on the server to communicate faster if it is using the Nagle algorithm (which produces delays in transmission). Most `telnetd` servers are configured to use this algorithm automatically. By enabling this feature, WindX sends a few bytes that force an immediate response from the server, which triggers any pending data on the server to be sent immediately. Not all Operating System Telnet Daemons support this feature. Whether this option will have a positive or negative impact on performance can only be determined through testing.
Encrypt w/SSL	The option to Encrypt with SSL allows WindX to communicate with Telnet daemons which have SSL support, such as those using an SSL wrapper on the server. The Standalone WindX must be activated for SSL before this feature can be used. The WindX Plug-In does not require any special activation for SSL.

Use Keepalives	The TCP Keepalive option informs ProvideX whether or not a client is still communicating with the server.
Exit on Quit / Ignore Turbo Mode Errors	This option causes WindX to auto-terminate once the connection with ProvideX is severed. Local reporting of Turbo mode errors can be suppressed as well.

Automated Sign-on

There is also a built-in script processor within WindX which can be used to dial a phone and/or generate a host sign-on sequence. The script text consists of a series of lines which starts with a single character code followed by a colon.

The script consists of a series of lines each of which contains a single letter identifying the command, a colon, and the command parameter:

`R:`*xxxxx*	Wait to receive *xxxxx* in the Data flow or Timeout
`S:`*xxxxx*	Send *xxxxx* to host followed by a carriage return (The Enter key)
`W:`*nnn*	Wait *nnn* milliseconds
`T:`*nnn*	Change reception timeout value to *nnn* seconds.
`M:`*xxxx*	Display *xxxx* on the status bar to advise user of state

The `T:` command can be used to change the default timeout value (5 seconds) used for a `R:` (receive) function. `W:` causes the script to wait the number of milliseconds specified. If the first command of a WindX script is `M:`, then all terminal output is suppressed allowing for a 'hidden' sign-on. Terminal output resumes automatically at the end of script.

For example, the following sign-on script for UNIX suppresses the login so the user would not see the user ID and password:

```
M:Starting host connect
R:login
S:Stitch
R:Password
S:Lilo
R:$
S:/pvx/pvx MYMENU
```

Connect Script

While the script processor is sufficient for many situations, an external sign-on program can also be used to alter the script sequence dynamically. If the option Use External Sign-on is enabled on the configuration screen, then WindX will issue a **CALL** to the program named `windx.sgn` just prior to processing the sign-on script. This program can alter the script as desired.

For example, if a NOMADS screen (`Signon` in `Mylib`) requests a user ID and password, the `windx.sgn` program can be used to alter these fields in a script:

```
0010 enter s$
0020 process "Signon", "Mylib.en",u$,p$
0030 u=pos("$user$"=s$)
0040 if u<>0 s$=s$(1,u-1)+u$+s$(u+6)
0050 p=pos("$pswd$"=s$)
0060 if p<>0 s$=s$(1,p-1)+p$+s$(p+6)
9000 exit
```

This example assumes that the value `$user$` and `$pswd$` exist as placeholders in the script and will be replaced with the true user ID and password.

Launching WindX

Once WindX is installed on the client system and is configured to access the host, it can be launched to establish a connection with the server-based ProvideX system. The WindX launch procedure differs slightly depending on the connection type, the hosting facility, and the intended application.

The command line syntax for launching WindX can be customized in several ways for various purposes. For example, if WindX was configured for a **Telnet/Serial Configuration**, it might be launched as follows:

```
"C:\WindX V7.00 Standalone\pvxwin32.exe" windx -arg my.cfg auto
```

This automatically launches a WindX session and applies the saved configuration file, `my.cfg`. The command line syntax for launching WindX using a direct TCP/IP connection is explained in the sections **Application Server *CLIENT Program** and **ProvideX *NTSlave Program** below.

INI Files and Shortcuts

As with any ProvideX application, the WindX launch sequence and various application properties can be customized and packaged into initialization (INI) files for controlling system resources and for defining how the application will be presented to the user. WindX may be set up to be launched from a Windows shortcut by placing the command string in the Target field; e.g.,

```
"c:\pvx\pvxwin32.exe" c:\myapp\myapp.ini *ntslave -ARG ...
```

This automatically launches WindX for use with ***NTSlave** on the client system.

Application Server *CLIENT Program

This software is used on the client workstation to establish a connection to the ProvideX Application Server running on the host machine. It is also used by the spawning process to initiate new sessions on the client workstation. ***CLIENT** can also be customized to provide a user interface for connecting to the application server. *Chapter 5.* **Application Server** includes a **Sample Setup Procedure** that illustrates use of ***CLIENT** launching WindX from a shortcut.

The command line (Target) syntax for launching ***CLIENT** from a Windows shortcut is described as follows:

PVXpath$ **[***state***] [***ini***]** ***CLIENT** **–ARG** *ServName* [*Socket*] ["*App*"] [*parms*]

Where

PVXpath	Path to ProvideX
state	Option setting for governing the initial WindX window. Either null for normal window, **-mn** to start minimized, **–hd** to start hidden.
ini	Optional user-defined INI file. The initial session will use this INI for its client-side ProvideX characteristics. Spawned sessions will re-use the same INI, unless specifically overridden during the spawn.
–ARG	Keyword marking the start of the argument list.
ServName	IP Address or DNS resolvable name of the server to connect to.
Socket	TCP/IP port (socket) that the server daemon is listening to. The default is 10000.
"*App*"	Optional lead program or configured application name that the server is to run. Quotes are required if it contains spaces. If Null or not given, then the request is for a ProvideX Console Mode Session.
parms	Optional dashed parameters: **-FID=***xxx* **-DIR=***xxx* **-CMD=***xxx* **-ARG=***xxx* **-SSL -LOGIN -KA -LANG=***xx* **-USR1=***xxx* **-USR2=***xxx* **-USR3=***xxx* **-USR4=***xxx*. For full details, see ***CLIENT Dashed Parameters**, *p.108*.

Example

The following command line uses ***CLIENT** to connect to the ProvideX Application Server running on IP address 10.100.22.243 (at socket 15000) and then runs the configured application on the server called `"myapp"`.

```
"C:\PVX\pvxwin32.exe" *client -ARG 10.100.22.243 15000 "myapp"
```

ProvideX *NTSlave Program

*NTSlave provides a more basic connection for WindX to communicate with the host system. This process connects to any server running *NTHost, whether it is Windows, UNIX, or Linux. This makes it possible to have many shortcuts connecting to different servers, all running at the same time. Refer to the default client-server configuration explained under **Using *NTHost/*NTSlave**, *p.18*.

The command line (Target) syntax for launching ***NTSlave** from a Windows shortcut is described as follows:

PVXpath$ ***NTSlave** **–ARG** *server prog port*

Where:

PVXpath$	Path to ProvideX
–ARG	Keyword marking the start of the argument list.
server	Either the IP address of the host system where *NTHost is running or the host name of the server.
prog	Name of program to run. If `""` *empty quotes* are specified (no program name) then the session starts from the default ProvideX console.
port	TCP/IP socket number that *NTHost is monitoring.

Example

The following command line uses ***NTSlave** to connect to NTHost running on IP address 10.100.22.243 (at socket 15000) and then runs the ProvideX program called `"myapp"` on the server.

```
"C:\Program Files\Sage Software\WindX V7.00
     Standalone\pvxwin32.exe" *ntslave -ARG 10.100.22.243
     "myapp" 15000
```

WindX Thin-Client Functionality

Once ProvideX determines that it is dealing with a WindX implementation, it automatically routes all graphical directives and functions from the server program to the client. Little or no code changes are required to move ***GUI processing*** over to the WindX client, but the server application does need to recognize the existence of the WindX session.

The methods for detecting WindX (and JavX) involve checking the **MSE** variable and checking the value returned in **TCB(88)**. For Telnet connections, ProvideX uses terminal type (`TERM=`) to detect the WindX session.

The methods for determining the existence of an active thin-client are explained in the section **Standard Thin-Client Behaviour**, *p.21*.

[WDX] Remote Capabilities

Because WindX is itself a ProvideX program, it has full access to local ProvideX functionality along with the ability to maximize resources on either side of the client-server connection. WindX implementations are able to take full advantage of the latest ProvideX features with a simple upgrade of the local copy of ProvideX.

Through WindX, the host application is able to access files, execute code, and launch applications installed on the client system. This allows programs and files to be distributed in such a way that processing logic is executed on the same system as the data, which reduces processing time, accelerates throughput, and increases overall system performance.

Access to the client system is invoked by using the **[WDX]** prefix in statements on the server-side application. However, remote processing is only possible if a given command works on both client and server systems, and if it supports use of the **[WDX]** special command tag. The **[WDX]** tag enables the following functionality across a WindX connection:

- Initiate remote commands via **EXECUTE** and **INVOKE**.
- **CALL** a subprogram that exists and runs remotely on the client
- Open remote serial ports and forward I/O directives for processing
- Direct printing to the client or server system via ***WINPRT*** and ***WINDEV***.
- Create and manipulate OOP/COM objects.
- Invoke thin-client utilities via **WindX.utl**.

Note: The syntax for the above remote actions is fully documented under the heading **[WDX]** Direct Action to Client Machine in the *ProvideX Language Reference*.

When programming for WindX functionality, it might be advisable to create a global string variable with the value **[WDX]** to be implemented when WindX is detected. This global string variable can then be used to prefix any commands that you want to be executed locally; e.g., `invoke %wdx$+"notepad.exe"`.

Print Management

The ***WINPRT*** and ***WINDEV*** special device files are specific to the Windows operating system and will be opened on any Windows system (server) that issues the command. However, from a UNIX or Linux server, ProvideX automatically directs printing to the WindX client; e.g.,

```
OPEN (14)"*winprt*" !For the PC client from UNIX host.
```

To direct print jobs and dialogues to the client PC, use **[WDX]** with **OPEN** [**INPUT**] directives for the two special device files. The client will in turn use its Windows print subsystem API to deal with the jobs and send them to the given printer; e.g.,

```
OPEN (7,ERR=1500)"[WDX]*WINPRT*"
```

With a WindX client and an NT Server, if you ***do not*** use **[WDX]** in your **OPEN** directive, then the printer selection dialogue will appear on the server console, and any print queue you name directly must exist on the Windows server in the Control Panel printers folder.

Note: Refer to syntax descriptions of ***WINDEV*** Raw Print Mode and ***WINPRT*** Windows Printing in the *ProvideX Language Reference*.

Local File Access

To access files on the client workstation, the pathname specified on an **OPEN** or file creation/ deletion directive simply has to include the **[WDX]** tag; e.g.,

```
OPEN LOCK (1) "[WDX]A:\PAYROLL.DAT"
READ RECORD (1,END=END_DATA) R$
...
END_DATA: CLOSE (1)
```

By specifying `[WDX}A:\PAYROLL.DAT`, the host application will actually be given access to the file `A:\PAYROLL.DAT` on the workstation. To create a file on the workstation, simply issue a command such as:

```
SERIAL "[WDX]C:\MyData\TESTDATA"
```

This creates the serial file `C:\MyData\TESTDATA` on the workstation.

Local Command Processing

In addition to remote file access, the **INVOKE**, **SYSTEM_HELP** and **EXECUTE** directives support the ability to be processed locally on the workstation. If the string parameter passed to either of these directives starts with the sequence **[WDX]**, then the rest of the command is passed on to WindX to execute locally.

Example:

```
INVOKE "[WDX]EXCEL C:\MyData\Budget2006.XLS"
SYSTEM_HELP "http://www.pvx.com"
```

The most common uses for issuing a remote **EXECUTE** include:

- Changing local directory
- Changing system parameters
- Changing the Prefix

- File creation

Note: When using **EXECUTE** remotely, it is possible that the client system is running ProvideX with different syntax tables, which could result in the directives being incorrect.

Remote Procedure Call

One of the more powerful features of WindX is its ability to execute ProvideX subprograms remotely. Since WindX itself is a ProvideX program, it is capable of calling and passing parameters to a local sub-program. To **CALL** a program to be run on the WindX workstation simply prefix the program name with **[WDX]**. This indicates that the specified sub-program is to be run on the local PC, not the host system. Remember that the program being called must exist on, or be accessible from, the client PC.

Any arguments specified in the **CALL** are sent to the WindX PC and any changes are passed back as per normal **CALL** processing. There is a limit of twenty (20) arguments that can be specified in a remote **CALL**.

It is important to remember that the called program is actually running on the client workstation, not on the host system that initiated the **CALL**. Only the variables that have been passed specifically on the **CALL** argument list are accessible. Global variables and files are not accessible to called programs.

***windx.utl** Utility Program

WindX (and JavX) include a general purpose utility program called ***windx.utl** that supplies a number of commonly needed functions, such as getting the current directory or workstation address. It also has an entry point that allows you to spawn a new session of ProvideX on the host and an associated WindX session on the client system.

A complete list of the available ***windx.utl** functions and syntax descriptions is provided under **Thin-Client Utility**, *p.28*.

4

JavX - Java-Based Thin-Client

JavX offers a platform-independent thin-client for displaying and interacting with your server-based ProvideX applications. With JavX, your applications can be run via any web browser anywhere, as well as on an increasing number of J2ME-enabled mobile/handheld devices.

Three editions of JavX are currently available to best serve the requirements of your target platform (PC/workstations, mobiles/handhelds, and embedded devices):

- **JavX SE** (*Swing Edition*) designed for desktop systems that run the Java 2 Standard Edition (J2SE) runtime environment — this includes Windows, Linux and UNIX X-Windows, and Apple Mac OS X systems. See **JavX for PC Platforms**, *p.60*.
- **JavX AE** (*AWT Edition*) designed for small devices that run the Java 2 Micro Edition (J2ME) Constrained Device Context (CDC) Personal Profile — this includes a variety of personal digital assistants (PDAs). See **JavX for Portable Devices**, *p.73*.
- **JavX LE** (*Light Edition*) designed for task-specific devices that run the J2ME CDC Foundation Profile — this includes a range of consumer products, automotive and other interactive components. See **JavX for Portable Devices**, *p.73*.

Note: JavX LE is currently not available for direct download; however, developers are welcome to contact Sage Software Canada Ltd. to request a copy.

For a discussion on the full range of client-server options, refer to the section **Choosing the Right Solution**, *p.10*.

This chapter starts with information that applies to all JavX editions and is then divided into sections that discuss the functionality that is specific to each of the client formats.

Java Runtime Environment

All JavX editions (***JavX SE/AE/LE***) run within a Java Runtime Environment (JRE) that has been created for each of the target PC and device platforms. A JRE contains the Java Virtual Machine, Java core classes, and any supporting files needed to run Java applications.

The ***JavX SE*** thin-client requires a Java 2 Standard Edition (J2SE) JRE. Many operating systems, hardware packages, and web browsers come with Java 2 fully installed and may not require any further downloads. Automated tools can be used to install the required runtime environments along with the JavX **JAR**. When **Launching JavX as an Applet**, then HTML code and Java script can be used within a web page to automatically download/install the JRE (if it is required). These runtime requirements are further described in the section **JavX for PC Platforms**, *p.60*.

The ***JavX AE*** and ***JavX LE*** thin-clients require Java 2 Micro Edition (**J2ME**) JREs. J2ME delivers a reduced set of the Java core classes and it has been adapted for use in constrained/embedded devices. These runtime requirements are further described in the section **JavX for Portable Devices**, *p.73*.

Note: Visit www.java.com or www.java.sun.com to learn more about the Java concepts (and terminology) being used in this documentation.

Installation and Configuration

JavX components and utilities are downloadable for direct installation from the ProvideX website at **www.pvx.com**.

As mentioned earlier, no additional host software is necessary, apart from ProvideX– but some preparation is required to ensure that JavX is installed and deployed successfully for communication between the host and client sides of your applications. Depending on the implementation, the client requirements, and the available **Java Runtime Environment** (JRE), JavX may be deployed either as a Java application (permanently installed) or as a Java applet (downloaded as needed).

Developers who are new to the Java environment should refer to the **JavX Developer's Kit**, before they attempt to install and/or launch any version of JavX.

JavX Distribution Format (JAR Files)

All JavX installations are distributed in JAR (Java ARchive) file format, which is a compressed archive that may be opened using any ZIP-compatible software. In fact, the only file used to install any edition of JavX is the JAR itself, either `JavXSE.jar`, `JavXAE.jar`, `JavXLE.jar`.

Each JAR contains all of the compiled classes and files that constitute that edition of the JavX application. At runtime, the application can be deployed either as a Java applet, where the JAR is loaded temporarily into the client browser's cache, or as an installed application, where the JAR is kept permanently on the client machine.

JavX does not require that the JAR be named `JavX[SE/AE/LE].jar`. Use a name that is relevant to your application. You can also add your own items to the distributed JAR; e.g., custom GIF or JPG files can be added to the JAR to be used as internal images in your application.

Note: When you add items to the distributed JAR, it is probably best that you rename the changed file to avoid any possibility of it being overwritten by a subsequent download. Also, ensure that web page references point to the new file name.

JavX Developer's Kit

The JavX Developer's Kit is designed to help you get started developing applications for JavX. It contains the following items:

- A shortcut to launch JavX[SE/AE/LE] as an application with a connection to ProvideX's *Nthost facility (running on the local machine on socket 10000).
- A shortcut to launch JavX[SE/AE/LE] as an application and connect to the *ProvideX Application Server* (running on the local machine on socket 10000).
- A folder containing the Web Page Configuration program (`Javxpagegen.pvs`) and its documentation. This is an easy-to-use program for building web pages that will run JavX as an applet. For more information refer to the **Web Page Generator**, *p.70*.
- A folder containing all JavX documentation, including this document and documentation on using the Web Page Configuration program.
- The three different JAR files for JavX SE, JavX AE, and JavX LE.
- Optional Java 2 Runtime Environment (JRE). All three versions of JavX will run in the J2SE JRE. We recommend that you select the Java VM/JRE option when:
  - you are not sure that your system has the appropriate JRE
  - you are installing JavX for the first time.

Note: The JavX Developer's Kit is freely downloadable from www.pvx.com. Versions of the kit are available for almost every platform. JavX licensing requirements apply. See **Licensing ProvideX Client-Server Facilities**, *p.13*.

Launching JavX

This section provides general information and procedures for **Launching JavX as an Application** and **Launching JavX as an Applet**. Procedures that are specific to JavX SE are provided in the section **JavX for PC Platforms**, *p.60*. Procedures for launching JavX AE and JavX LE on a device are provided in the section **JavX for Portable Devices**, *p.73*.

Launching JavX as an Application

When JavX is run as an application, the distributed **JAR** will be installed directly on the workstation or device, and JavX has local access to the client system with no imposed security restrictions. Utilities such as *InstallShield* or *InstallAnywhere* can be used to create a custom routine for automating the installation procedure. The following syntax launches JavX as an application from the command line:

JavaLaunch **-Jar Javx.jar** *ArgString$*

Where:

JavaLaunch The appropriate Java program to run a jar file. Most commonly named either `java` or `javaw` (on some windows platforms).

ArgString$ Variables to be passed from the command line in one string delimited by ';' (i.e., `"server=pvx.com; port=10000;"`). The ***required*** and ***optional*** arguments are described under **HTML Parameters**, *p.65*.

The following are different examples of JavX being launched via the command line:

```
java -jar JavXAE.jar "server=127.0.0.1; port=10000;"
```

This launches an instance of the JRE. It tells JavX to connect to the local machine that is listening on port 10000 and to sit at a ProvideX command line prompt rather than run a program.

```
java -jar JavX.jar "server=www.pvx.com; port=11000; program=*nomads;"
```

This launches the JRE, connects to *NTHost, then runs NOMADS after connecting to port 11000 on the server located at **www.pvx.com**.

```
java -jar JavXSE.jar "server=www.pvx.com; port=11000; program=*nomads;
      applicationserver=true; ssl=true;"
```

This launches the Java Run-Time, connects to the ProvideX Application Server on a secure socket, then runs NOMADS after connecting to port 11000 on the server located at **www.pvx.com**.

```
java -jar JavX.jar "server=66.46.24.226; port=31000;
      ConnectString=V2|mydir/myprog -ARG value1|T0|pvxuser|;"
```

This connects JavX to the server at `66.46.24.226` (ProvideX WebServer) on socket 31000. The program to run is set to null since the program is specified in the `ConnectString`. All pipe symbols in the `ConnectString` are substituted with `$8A$`, then the entire string is sent to the server. JavX waits for a response that consists of the socket number on which the server will run the requested program.

Launching JavX as an Applet

Currently this option is only available under **JavX for PC Platforms**, *p.60*. When JavX SE is deployed as an applet, the **JAR** is not installed permanently on the client workstation, but is delivered (as needed) via web server to the client's web browser.

The `JavXSE.jar` file must be made accessible from a website and the web server should have the correct mime types for Java files: `.jar` (application/octet-stream) and `.class` (application/x-java-applet).

The JavX applet occupies a rectangular region on the web page. These regions are defined by the `<APPLET>`, `<OBJECT>`, or `<EMBED>` settings within the HTML. When a browser encounters these settings, it identifies the applet and its location then loads and runs the applet. If a copy of the applet is not cached on the local system then it automatically downloads it from the website.

Launching JavX Clients without Arguments

JavX can be run without passing arguments at startup by including a Java property file called `JavX.properties` to the JavX **JAR**. This is simply a text file with each argument specified on a separate line. If no arguments are passed to JavX, it automatically attempts to open and read `JavX.properties`. For example, if `JavX.properties` contained the following, it would cause JavX to connect to an instance of *NTHost listening on Socket 10000 on the local machine:

```
# This File will pass the following "args" string to JavX:
# "server=localhost; port=10000;"
server=localhost
port=10000
```

In the above JavX.properties file, the lines starting with a # are comments and the arguments are not delimited by semi colons. When passing arguments from a command line or on an HTML page the arguments are delimited by semi colons.

Common Functionality and Limitations

Fundamentally, JavX can be considered a Java version of WindX. While WindX is written in ProvideX (which is written in C), JavX is written entirely in Java. However, because it is Java-based, JavX offers a platform-independent solution for displaying and interacting with ProvideX host applications. With JavX, applications can be run via any web browser anywhere, as well as on an increasing number of J2ME-enabled mobile/handheld devices.

Note: If you are programming with JavX for the first time, we recommend that you download and install the **JavX Developer's Kit**, *p.45*.

While JavX may not possess every feature available under WindX and ProvideX, its Java-based design offers several ***clear advantages***:

- Simpler World Wide Web application.
- Platform independence.
- *Look-and-Feel* flexibility.
- Runs without local ProvideX requisite.
- High speed communications.

JavX is not a ProvideX program so it can be deployed on any system without the need for a local copy of ProvideX. When programming for JavX as an applet, nothing is required on the client machine but a web browser and a Java2 JRE. The user simply navigates to the web page that contains JavX and the applet is sent to the browser. If the browser is closed, then the session is terminated.

ProvideX Features not Supported in JavX

JavX architecture has the following limitations in addition to those described in the section **JavX vs. WindX**, *p.26*. These features are not supported by any of the JavX client formats. For functionality that is specific to each of the client formats, refer to the sections covering **JavX SE** or **JavX AE/LE**.

Buttons. The following limitations apply to use of the **BUTTON**, **RADIO_BUTTON**, **CHECK_BOX**, **TRISTATE_BOX** directives:

1. The property **'BitmapPosition** is supported, but not completely. This property can be set to **1** (*left of text*) or **2** (*right of text*); however, **3** (*above text*) and **4** (*below text*) are not supported in JavX. For more control over images on buttons, use HTML for the text. For more information on using HTML on buttons, see **Language Enhancements for Thin-Clients**, *p.23*.

2. Button text is not wrapped automatically when it is too long; however, the text will be wrapped using HTML. For example,

   ```
   BUTTON 10,@(5,5,5,5)="This Text is Too Long"
   ```

   This would be truncated as "This…" depending on the font size specified.

```
BUTTON 10,@(5,5,5,5)="<html>This Text is Too Long</html>"
```

This wraps the text as expected.

3. The **OPT=** options **"O"** (*Steal Focus*) and **"s"** (*Scroll*) are not supported.

List boxes. The following limitations apply to use of the **LIST_BOX** directive:

1. When creating a ***Listview*** list box (`OPT="l"`), JavX does not support the format definition, sorting, and load-on-demand options. If the formatting (**FMT=**) option is specified, then JavX will create a vertically-scrolling list box (and sort will be supported).
2. The **OPT=** options **"E"** (*Edit Mode*) and **"V"** (*Full Row highlight*) are not supported.

Multi-lines. The following limitations apply to use of the **MULTI_LINE** directive:

1. Formatted multi-lines are not supported; e.g., a format mask of "###" specified by `FMT="###"`, is ignored in JavX and alphanumeric characters could be entered by an end user. Applications requiring format masks on multi-lines must apply the format on the server side.
2. The following **OPT=** options are not supported:

"**#**" (*Implied decimal point*)	"**!**" (*Support for Arabic characters*)
"**C**" (*Centre the input*)	"**F**" (*Full*)
"**H**" (*Hide*)	"**i**" (*Suppress implied decimal*)
"**I**" (*Activate implied decimal*)	"**R**" (*Right Justify*)
"**s**" (*Scroll*)	

Grids. The **GRID** directive has ***limited functionality*** under JavX. It is effective as a view (in JavX SE) but does not respond to input exactly as it does in ProvideX. Applications that are designed to use grids for data input may not respond as expected and should be adapted for use in JavX.

The following limitations apply to use of the **GRID** directive:

1. Some **GRID** properties are not supported: **AutoSequence**, **ImpliedDecimal**, **SepLoad**, **LockColumns**, **Len**, **CellTag**, **FillColour**, **CellImpliedDecimal**, **CellFormat**, **InsDelEnabled.**

   Use the following flag to ignore (not report) non-essential grid errors:

   ```
   PRINT 'OPTION'("REPORTGRIDERRORS","FALSE")
   ```

   This causes the grid to not report an error when setting any of the unsupported attributes listed above.

2. The following ***Cell Types*** appear as a simple check box:

`"CheckBoxRaised"`	3D check box that looks raised
`"CheckBoxRecessed"`	3D check box that looks recessed.
`"CheckMark"`	Check box that uses a check mark.
`"CheckMarkRaised"`	Raised check box that uses a check mark.
`"CheckMarkRecessed"`	Recessed check box that uses a check mark.

The following ***Cell Types*** are not supported: `"Query"`, `"QueryHideBtn"`, `"EllipsisDrop"`, `"VarDropBox"`, `"VarDropBoxHideBtn"`, `"UseTextNormal"`, `"UseTextSingleLine"`, `"UseTextEllipsis"`

Chart. JavX does not support the **CHART** directive at this time.

Local File I/O. Sequential serial files (without `ISZ=` specified). In ProvideX, the **KEC( )**, **KEF( )**, and **KEL( )** functions are valid on a sequential file, but not in binary file access. In JavX, these return an `Error 98: Feature not yet implemented`.

Menus. Images cannot be added to menu items (**MENU_BAR** or **POPUP_MENU** items). A menu item with an image will function but the image will not be visible.

Mnemonics. The following **OPT=** attributes are not supported for the **'DIALOGUE'** and **'WINDOW'** mnemonics:

& *Ampersand* - creates a window that logically attaches to the current window (i.e., leaves the current window active and shares controls)

***** *Asterisk* - creates a resizable window with automatic scrollbars for text plane; e.g., `PRINT 'DIALOGUE'(1,1,60,20,"Title",OPT="*")`

^ *Caret* - window is always on top (not applicable to the **'WINDOW'** mnemonic); e.g., `PRINT 'DIALOGUE'(1,2,30,3,"My Top Dog",OPT="^")`.

C or **X** Disable [X] close button.

The **'4D'** mnemonic works in JavX to draw XP-style frames; however, because JavX is designed to run on a wide variety of platforms, the appearance of GUI components is not controlled by the **'4D'** mnemonic but by the current ***Look and Feel***. The application will only show XP-style controls and frames if the **'4D'** mnemonic is set, and the current Look and Feel is set to "Windows" or "OS default". The **FRAME** mnemonic will have an XP Look and Feel (blue coloured rectangle with rounded corners) on any platform, but GUI components (e.g., buttons) will only look like XP if JavX is running on a Windows XP platform; i.e., JavX will not have an XP Look and Feel on an Apple OS X system.

For an in-depth discussion on this topic, refer to article entitled *Java Look and Feel Design Guidelines* on Sun's Java website: http://java.sun.com/products/jlf.

[WDX] Tag Support

As previously mentioned, because it is written in Java rather than ProvideX, JavX is not able to match every feature available to WindX. JavX does not run under ProvideX on the local machine, and the **[WDX]** tag is generally ignored. However, for compatibility purposes, it can *emulate* some **[WDX]** functionality.

To emulate a local ProvideX program call, simply add the name of the program to be called plus a return value to the `JavX.properties` file or to the argument string that is to be passed to JavX at startup.

Examples:
A ProvideX program running on the server executes the following:

```
Call "[WDX]myLocalProgram;getUser",retVal$
```

The following entry in the JavX.properties file would emulate the ProvideX program by simply returning `JavXUser`.

```
myLocalProgram,getUser=$JavXUser
```

Note that the comma is used in the properties file rather than a semi-colon. This is because JavX uses the semi colon as a delimiter when handling the argument string. JavX maps commas in the argument string to semi-colons when calling local ProvideX programs. Also notice that the value `JavXUser` is proceeded by a `$` dollar sign to indicate that this value should be returned to the ProvideX host program as a string.

Usually a static value is not required by your application. Typically the value must be retrieved or calculated dynamically on the client machine. When static mapping of a program name and routine to a variable is not sufficient, it is possible to call a Java class and method. For example, a ProvideX program running on the server calls a local program called `*WIN/COLOUR` to retrieve a color value selected from a palette on the WindX workstation. In JavX, the `*WIN/COLOUR` program does not exist, but a Java class called `MyColorChooser` that displays the Java colour chooser object does exist. The following example loads and instantiates the Java class `MyColorChooser` and call the method `getColor`:

```
*win/colour=[java]myjavaPackage.MyColorChooser.getColor()
```

The `[java]` tag tells JavX not to simply return a string variable, but rather to instantiate the class `MyColorChooser`, and return the value returned by the method `getColor()`.

This is not an ideal way to interface with Java classes because the handle to the class being loaded is not retained, and only one method can be called. To call multiple methods the class would have to be instantiated multiple times. The preferred method of interacting with third party Java is via the ProvideX OCX/ActiveX Interface. For more information, refer to the **ProvideX COM Interface and JavX**, *p.55*.

*windx.utl Utility

JavX supports use of the thin-client utility program called ***windx.utl**. It also supports the syntax and functionality for spawning a new session of ProvideX on the host and an associated JavX session on the client system. A complete list of the available ***windx.utl** functions and syntax descriptions is provided under **Thin-Client Utility**, *p.28*.

Printer Support

End users expect to be able to print to printers that are attached to their local machines. However, unlike WindX, JavX does not allow that level of access to the local system. If the host application is set up to do so, it may allow print requests to be directed to somewhere on the server side of the network, but the JavX client does not have access to any of the print facilities on the user's machine itself.

Alternative Printing Solution

However, it is not difficult to implement *alternative printing solutions*. One solution for printing web content is based on ***Adobe Acrobat***. It uses the **SYSTEM_HELP** directive with JavX to launch a new browser session, where a URL points to a PDF document created on the server. The client workstations can have their browser retrieve the generated PDF. When it is opened in Adobe Reader, it can then be printed to the local printer. The following steps make this possible:

1. On the server, install the full version of Adobe Acrobat, which then adds its own printer driver ***Adobe PDFWriter*** for generating PDFs.
2. Create a new ProvideX device driver that:
   - Creates a file name for the PDF document, with a path to a directory accessible to the web server.
   - Opens `*WINPRT*` directly to the ***Adobe PDFWriter*** printer, and pass it the generated file name using a `FILE=` clause; e.g.,
     `open (`*channel*`)"*winprt*;adobe pdfwriter;file=c:\...\docs\`*xxxa*`.pdf"`
   - Sets the **'*X'** mnemonic in the device driver to call the device driver again at a routine called `OnChannelClose` when the channel is closed:
     `MNEMONIC(`*channel*`)'*X'=PGN;"OnChannelClose;URL=http://www..."+`*filename$*
   - Contains a routine at the end of the device driver with the label `OnChannelClose`. This logic should parse the `URL=`*data$* returned using ***data$*=MNM('*X*',*channel*)**
   - Issues a **SYSTEM_HELP** command to the URL required for a connection to the web server.

The **'*X'** mnemonic is used to call a program when the channel is closed. This program issues a **SYSTEM_HELP** command that causes JavX to launch a new browser session that downloads the PDF automatically. Because the file is being viewed through Acrobat Reader, a standard plug-in for most browsers, the user will be able to print it to any local printer.

More complicated URL requests could be used, such as an HTTPS request to transmit the file using SSL encryption. Also, the request could be sent to a CGI or ProvideX web program with additional information, such as a USERID and PASSWORD to access a particular report, rather than being sent directly to the PDF location. That program could validate the user request and decide whether or not to make the report available to the person attempting to download it, which in this case would be the person using the particular JavX connection.

Image Support

Note: JavX only supports image formats that are supported by Java; i.e., Java does not support ***.bmp*** format.

In ProvideX, an image used by a thin-client must be available to the thin-client itself and may come from several sources:

- An internal bitmap; i.e., the image name starts with an ! *exclamation mark*. Internal bitmaps may come from the *BMP directory on the client, a resource DLL on the client, or from within the ProvideX executable on the client.
- A filename that is resolvable from the client's point of view, such as a local hard disk or a networked mapped drive.

In JavX, displaying images is more complicated due to the fact that Java does not display bitmaps and because JavX does not have access to the local file system when run as an unsigned applet. While JavX must retrieve images in a different manner than standard ProvideX, the basics are similar. The image must still be available to JavX itself, and may come from internal or external sources.

The following instructions apply both to the **{ }** images named in a GUI object (e.g., {!STOP}), and to image names used in the **'PICTURE'** mnemonic.

Internal Images.

Internal images are images whose names start with an **!** *exclamation mark*. To find an internal image, JavX first resolves the image name. If the internal image name does not have a file extension, then .GIF is appended to the image name and the name is forced to lowercase. If a file extension is specified then JavX maintains the case used.

Examples:

```
!STOP  will become "stop.gif"
!MyPict.jpg  will become "MyPict.jpg"
```

JavX then tries to locate the image. The JavX JAR file is scanned first for the internal image name (without the exclamation.) and, if found, it is displayed. If the image is not found within the JAR file, then JavX will use one of two approaches to find the image depending on whether it is running as an application or applet. For information on adding your own internal images, see **JavX Distribution Format (JAR Files)**, *p.44*.

If JavX is running as an application, then it will look for the image as a file on the local hard disk. JavX takes the directory name where its jar file is located, appends the image name to it and attempts to open that file; e.g.,

Image name is:	`!MyLogo.jpg`
JAR is launched as:	`C:\ProvideX\JavX\JavX.jar`
Disk file to search for:	`C:\ProvideX\JavX\MyLogo.jpg`

If JavX is running as an applet, then JavX will generate a URL for the image, and attempt to retrieve the image from the web server. JavX takes the URL that launched the JavX applet, removes the jar file, appends the name of the image, then makes a request to the web server for that URL.; e.g.,

Image requested:	`!STOP`
JavX applet URL:	`http://www.pvx.com/login/JavX.jar`
Image as a URL:	`http://www.pvx.com/login/stop.gif`

External Images

JavX also displays external images, which are images whose names do not start with an **!** *exclamation mark*. The processing of external images depends on whether JavX is running as an application, signed applet, or as an unsigned applet.

When running as an application or as a signed applet, the external file names used in the **{ }** GUI object specification or the **'PICTURE'** mnemonic may come from a file on the local hard disk of the client workstation, or any drive that the workstation has mapped. Alternately you may specify any URL to retrieve the image from a web server. See below for further information on URL usage.

When running as an unsigned applet, JavX cannot access the local file system, and therefore cannot retrieve images from the workstation. You may use a URL. However, to retrieve images from a web server, there is one restriction: an unsigned applet may retrieve only URLs which originate from the same source machine as the applet itself.

For example, if JavX is running as an unsigned applet loaded from the web server at `http://www.pvx.com/javx/JavX.jar`, then information can only be retrieved from URLs at the same web server; i.e., `http://www.pvx.com`.

Note: When using URLs to retrieve image names, ensure that you encode them properly; e.g., `"http://www.pvx.com/images/My%20Background%20Image.jpg"`.

Examples:

```
BUTTON 10,@(40,5,10,1.2)="{http://www.pvx.com/images/butimg.gif}"
PRINT 'PICTURE'(@X(40),@Y(5),@X(1),@Y(2.2),"http://www.pvx.com/
      images/butimg.gif",0),
```

ProvideX COM Interface and JavX

The ProvideX COM interface has been implemented in JavX to enable access to Java classes and applications. Refer to the document *Automation in ProvideX* for more information on the ProvideX COM interface.

Most ProvideX developers will not use the COM support in JavX. However if your ProvideX application needs to interact with third party Java classes, then the JavX OCX interface is a great solution. Accessing Java classes through the ProvideX COM interface in JavX is very similar to accessing COM objects in WindX. For example, to create an instance of a Java AWT Button class, execute the following:

```
DEF OBJECT BTN,@(5,5,5,5),"[WDX]java.awt.Button"
```

Array Support

JavX supports the ProvideX extended object *VARIANT, but does not currently support the following Extended Objects: *VARARRAY, *MASTER, *ERROR. *VARARRAY may be supported in a future release of JavX.

Currently, JavX supports arrays through a JavX-specific extended object *ARRAY. Java is a tightly-typed language. Therefore, an array type is required when creating arrays. The size of the array is also required. For example, the following creates an array of ten strings:

```
DEF OBJECT MY_STRING_ARRAY,"[wdx]*ARRAY,java.lang.String,10"
```

Accessing the array is exactly the same as accessing *VARARRAYs in ProvideX. The following sets the first element in the array to the string "First" (Java arrays are 0 based):

```
STRINGARRAY'VAL.PUT(0,"First")
```

JavX can be used as a gateway to a world of valuable Java classes. For example, all of the major databases (MySQL, Oracle, SQL Server, etc.) have JDBC drivers. The following program loads a Java JDBC Driver and reads a MySQL Database:

```
00010 PRINT 'CS
00020 MULTI_LINE 10,@(10,2,50,15)
00030 DEF OBJECT Driver,"[wdx]com.mysql.jdbc.Driver
00040 DEF OBJECT DRIVERMANAGER,"[wdx]PvxDriverManager"
00050 LET URL$="jdbc:mysql://10.100.29.247/test"
00060 LET connection=DRIVERMANAGER'getConnection(URL$,user$,password$)
00070 LET M=connection'GETMETADATA()
00080 DEF OBJECT STRINGARRAY,"[wdx]*ARRAY,java.lang.String,1"
00090 STRINGARRAY'VAL.PUT(0,"TABLE")
00100 LET RESULTSET=M'GETTABLES("test",*-1,*-1,*STRINGARRAY)
00110 !
00120 DEF OBJECT b1,"[wdx]*VARIANT"
00130 DEF OBJECT b2,"[wdx]*VARIANT"
00140 LET b1'val=1
00150 LET b2'val=1
00160 LET b1'type$="B"
```

```
00170 LET b2'type$="B"
00180 !
00190 LET curItem$="Table and Table Index list"+SEP+SEP
00200 WHILE RESULTSET'NEXT$()="true"
00210 ! loop through tables RS
00220 LET tableName$=RESULTSET'GETSTRING$("TABLE_NAME")
00230 LET curItem$+="Table Name: "+tableName$+":"+SEP
00240 LET
      indxInfo_ResultSet=M'GETINDEXINFO("test",*-1,tableName$,*b1,*b2)
00250 !
00260 LET curItem$+="Index List for "+tableName$+":"+SEP
00270 WHILE indxInfo_ResultSet'NEXT$()="true"
00280 ! loop through the tables indxInfo RS
00290 LET curItem$+="
      "+indxInfo_ResultSet'GETSTRING$("INDEX_NAME")+SEP
00300 WEND
00310 LET curItem$+=SEP
00320 indxInfo_ResultSet'close()
00330 DROP OBJECT indxInfo_ResultSet
00340 !
00350 WEND
00360 RESULTSET'close()
00370 DROP OBJECT RESULTSET
00380 connection'close()
00390 DROP OBJECT connection
00400 DROP OBJECT DRIVERMANAGER
00410 DROP OBJECT Driver
00420 !
00430 MULTI_LINE WRITE 10,curItem$
00440 OBTAIN a$
```

Using Classes Without a "No Argument" Constructor

COM objects always have a public ***no argument*** constructor. A Java class's constructor may be private (typically when implementing a Singleton design pattern), or there may only be a public constructor that requires arguments. To maximize the value of the JavX OCX interface we've added the ability to load a class and then instantiate it. This means that a handle to a class can be retrieved, and then a static method in the class that returns an instance of the class can be called.

Example of a Singleton. The Calendar class (found in the `java.util` package) in Java is a Singleton. There can only be one instance of the Calendar class. Singletons are usually implemented in Java by declaring the contstructor as *not public*. To get an instance of the Calendar class in Java, a client object must call the Calendar class's public static method getInstance( ).

The following example retrieves a handle to the Calendar class, and then finds and calls the getInstance( ) method:

```
00010 !Get a handle to the Calendar class
00020 DEF OBJECT CALCLZZ,"[wdx]java.util.Calendar"
00030 ! find the method we want to call:  getInstance()
00040 ! first create an array that identifies the types for each argument
      the method requires
00060 DEF OBJECT PARAMETERTYPES,"[wdx]*ARRAY,java.lang.Class,0"
00070! Next get a handle to the static method getInstance
00080 LET
      GETINSTANCEMETHOD=CALCLZZ'GETMETHOD("getInstance",*PARAMETERTYPES)
00090 ! Finally call the getInstance method to retrieve a Calendar object
00100 LET CALENDAROBJ=GETINSTANCEMETHOD'INVOKE(*CALCLZZ,*ARGUMENTS)
00110 ! we can now have a Calendar object we can use
00120 LET DATE=CALENDAROBJ'GETTIME()
```

The getInstance( ) method returns an instance of a Calendar class (similar to a constructor). The Calendar object's getTime( ) method is called to return a Date object.

Event Support

The COM support in JavX is similar to COM support in ProvideX. There are a few minor differences; e.g., Java event listeners provide notification when an event has occurred on an object. The following example creates a button and ActionListener (an object that responds to action events):

```
10 DEF OBJECT JBUTTON,@(24.5,5,10,10)="[wdx]java.awt.Button"
20 EXECUTE "[wdx]on event java.awt.event.ActionListener from
      "+STR(JBUTTON)+" preinput 100"
30 OBTAIN A$
40 PRINT A$
```

This example adds `ActionListener` to the button and when an action event occurs, a CTL value of 10 is sent to the ProvideX host program. The following example adds mouse event support only:

```
10 DEF OBJECT JBUTTON,@(24.5,5,10,10)="[wdx]java.awt.Button"
20 EXECUTE "[wdx]on event java.awt.event.MouseListener from
      "+STR(JBUTTON)+" preinput 10"
30 OBTAIN A$
40 PRINT A$
```

In this example, the CTL value 10 will be returned to the host program when the user's mouse enters or exits the button, or the mouse button is pressed/released on the button. Essentially every possible mouse event causes the CTL value 10 to be returned to the ProvideX host.

For more fine grained event support it is possible to specify a specific method in an EventListener class that will cause the CTL value to be sent to the ProvideX host. The following example modifies the previous example so the CTL value 10 is only sent to the host when a mouse button is pressed on the button:

```
10 DEF OBJECT JBUTTON,@(24.5,5,10,10)="[wdx]java.awt.Button"
```

```
20 EXECUTE "[wdx]on event java.awt.event.MouseListener;mousePressed from
     "+STR(JBUTTON)+" preinput 10"
30 OBTAIN A$
40 PRINT A$
```

As in the ProvideX COM interface, the PVXEVENTS$ attribute of any object contains all of the possible events for that object. The following example lists all of the event listeners available to the `java.awt.Button` class and adds six different event listeners to a button (each firing a unique CTL value):

```
00011 PRINT 'CS'
00012 LET LSTBX=20
00013 LET EVENT_LIST_BX=30
00020 DEF OBJECT JBUTTON,@(24.5,5,10,10)="[wdx]java.awt.Button"
00030 LET MTDS$=JBUTTON'*
00040 LIST_BOX LSTBX,@(35,1,40,10),TIP="Attributes and Methods"
00041 LIST_BOX LOAD LSTBX,MTDS$
00042 !
00051 LIST_BOX EVENT_LIST_BX,@(35,12,40,10),TIP="Event listeners"
00061 LIST_BOX LOAD EVENT_LIST_BX,JBUTTON'PVXEVENTS$
00062 !
00070 EXECUTE "[wdx]on event java.awt.event.ActionListener from
     "+STR(JBUTTON)+" preinput 100"
00080 !
00090 EXECUTE "[wdx]on event java.awt.event.MouseListener;mousePressed
     from "+STR(JBUTTON)+" preinput 10"
00100 EXECUTE "[wdx]on event java.awt.event.MouseListener;mouseClicked
     from "+STR(JBUTTON)+" preinput 20"
00110 EXECUTE "[wdx]on event java.awt.event.MouseListener;mouseReleased
     from "+STR(JBUTTON)+" preinput 30"
00120 EXECUTE "[wdx]on event java.awt.event.MouseListener;mouseEntered
     from "+STR(JBUTTON)+" preinput 40"
00130 EXECUTE "[wdx]on event java.awt.event.MouseListener;mouseExited
     from "+STR(JBUTTON)+" preinput 50"
00140 !
00160 JBUTTON'SETLABEL("AWT Button")
00170 OBTAIN A$
00180 LET MYCTL=CTL
00190 SWITCH MYCTL
00200 CASE 10
00210 PRINT "Mouse Pressed:   ctl ="+STR(CTL)
00220 BREAK
00230 CASE 20
00240 PRINT "Mouse Clicked:   ctl ="+STR(CTL)
00250 BREAK
00260 CASE 30
00270 PRINT "Mouse Released:  ctl ="+STR(CTL)
00280 BREAK
```

```
00290 CASE 40
00300 PRINT "Mouse Entered:   ctl ="+STR(CTL)
00310 BREAK
00320 CASE 50
00330 PRINT "Mouse Exited:    ctl ="+STR(CTL)
00340 BREAK
00350 CASE 100
00360 PRINT "Action Event:    ctl ="+STR(CTL)
00370 BREAK
00380 END SWITCH
00390 IF CTL<>4 THEN GOTO 0170
```

JavX for PC Platforms

JavX SE (*Swing Edition*) represents the most full-featured edition of JavX. It is the recommended thin-client for delivering expressive GUIs to end users who may need to access the host application from any variety of desktop/laptop systems. The various client-server options are discussed in the section **Thin-Client Products**, *p.10*. The other JavX editions are documented in the section **JavX for Portable Devices**, *p.73*.

JavX SE is designed to run on platforms that support the Java 2 Standard Edition (J2SE) **Java Runtime Environment** (JRE), and it may be launched either as a Java application (installed on the client machine) or as a Java applet (downloaded to the client's web browser as needed). This version of JavX employs an advanced set of GUI building elements, known in Java as the *Swing* library, to provide a uniform look and feel on all supported platforms.

Runtime Requirements for JavX SE

The JavX SE thin-client architecture is built for the J2SE JRE, which provides the components needed to run Java as an applet or application. The runtime environment only needs to be installed once on a workstation and it includes other key deployment technologies: *Java Plug-In*, which enables applets to run in popular browsers, and *Java Web Start*, which deploys standalone applications over a network.

This version of Java is available for a wide variety of platforms, including UNIX, Linux, Mac OS X, or Windows 9x-2000/Me/NT4/Server 2003/XP/Vista as well as for installation on some mobile and embedded devices. However, the workstation versions are still more common. Apple OS X has Java SE JRE built into the OS. The larger PC/Windows OEMs (Dell, HP, etc.) install Java SE JRE on their pre-packaged systems. Most commercial versions of Linux also come with the Java SE JRE.

Popular web browsers (Netscape, FireFox, etc.) are equipped with the Java SE Plug-in. If Java is not already installed, the web browser usually handles the download and installation of the Java (as with all plug-ins like *Flash* and *Real Player*) seamlessly the first time a Java applet is encountered on a web page.

For more information on the different Java 2 platforms, visit www.java.sun.com.

Note: The **JavX Developer's Kit** can be used to install the latest version of the Java 2 runtime environment. See also **JavX Distribution Format (JAR Files)**, *p.44*.

If JavX SE is deployed as an application, several automated installation tools, such as *InstallShield* or *InstallAnywhere*, can be used to install the JavX **JAR** file (Java ARchive file) along with the required runtime environments. For a JAR file to be run as an application, you will need to install the Java JRE or a Java JIT (Just-In-Time) compiler or the full Java SDK.

If JavX SE is deployed as an applet, then HTML code and Java script may be used within an HTML page to determine if a JRE exists on the workstation. The option to automatically download and install the JRE can also be included on the HTML page. All Java 2 runtime environments can be downloaded from the java.sun.com website. Point your links to JRE locations in java.sun.com, or download the files in advance, and have the links point to locations on your own web/FTP server.

See the **HTML Reference**, *p.63* for details on these types of HTML pages, their requirements, and their usage.

JavX SE Deployment

Developers who are new to JavX and the Java environment should download the **JavX Developer's Kit**, before they get started. The JDK installs a `Documents` folder, two shortcuts that will launch JavX as an application connecting to the ProvideX Application server, and a folder called `JavX_PageGen`. It is important that you read all the available documentation beforehand.

Note: For a review of the limitations that apply to all JavX editions, refer to the section **ProvideX Features not Supported in JavX**, *p.48*. General features and limitations are listed under **JavX vs. WindX**, *p.26*.

While JavX SE may be launched either as a Java ***applet*** or as an ***application***, we recommend that you initially run JavX as an application to better understand how the product works. This option is available to all JavX editions and is fully described under **Installation and Configuration**, *p.44*. If you choose to deploy JavX as an ***applet***, this option is only supported under JavX SE and is fully described in the sections that follow.

As described earlier, when JavX SE is launched as an applet, the JAR is delivered only as it is needed. However, this does not mean that it will be downloaded for every connection to the web server. In fact, browsers generally cache and re-use Java applets (as with other web documents) to avoid unnecessary downloading and improve the startup performance of web pages.

Also, like most other web-based applications, the JavX SE applet does not have full access to local facilities, such as printers. Alternative printing solutions are discussed in the section **Printer Support**, *p.52*. The two security levels used to control access to the local workstation, ***Signed*** and ***Unsigned***, are discussed below in the section **Java Security and Digital Signing of an Applet**, *p.62*.

JavX Applet Syntax

The following JavX-specific parameters and values must appear within the applet reference of an `<APPLET>`, `<OBJECT>`, or `<EMBED>` tag within an HTML page. This is in addition to any parameters and values required by the `<APPLET>`, `<OBJECT>`, or `<EMBED>` tags themselves. For details, see the **HTML Reference**, *p.63*.

All parameters must appear within the following tags.

`Code`	Name of Java Class to run, usually `"NetworkClientApplet.class"` (case-sensitive). However, if connected to an SSL server, the value is `"SSLNetworkClientApplet.class"`
`Archive`	Java archive file to use. Value is always `"JavXSE.jar"` (case-sensitive)
`Args`	All the variables JavX requires to run are passed in one string from the command line, or on an HTML page in a parameter called `"args"`.

Note: The **JavX Developer's Kit** includes an application that handles this for you. The ***Web Page Configuration*** program (`Javxpagegen.pvs`) automatically builds the web pages that will run JavX as an applet. We highly recommend that you use this application instead of manually inserting the various applet formats. For more information refer to the **Web Page Generator**, *p.70*.

Java Security and Digital Signing of an Applet

When JavX is deployed within a web browser, the browser will impose security restrictions on what the Java applets may or may not do based on whether they are signed or unsigned.

A ***signed applet*** is one that contains a digital signature, whereas an ***unsigned applet*** does not. A digital signature contains information about the company that signed it. This allows an applet's origins to be authenticated and traced. Therefore, when an applet is tampered with, the tampering shows up during the validation of the signing authorization. Signing an applet ensures that only an approved copy will be used and that it will arrive intact containing only the functionality specified by the developer.

By signing an applet, a company takes responsibility for the actions of that applet, since signed applets can be traced back to the author who signed it. A malicious applet with a digital signature can be traced back to the responsible person or company. A signed applet has the same rights and permissions as a Java application. It has unrestricted access to the user's workstation and file system.

The functional restrictions imposed on unsigned applets are identified as follows:

- Access to files on the local workstation is denied.
- Access to printers on the local workstation is denied.
- Any TCP/IP connections within the session may only communicate with the same server from which the applet was invoked. An exception to this is when a new browser session is opened, as the new browser session may be pointed to a server other than the original server that served up the applet. This exception works because the new browser session is independent of the originating java session that spawned it.
- An extra message bar is displayed on a Dialogue that indicates the applet is unsigned.

The applet designer controls all the functions used within an applet, and as such, controls what the applet does and is capable of doing. The designer also determines which local files will be accessed and the associated read / write operations. The decision to sign prevents the applet from being tampered with and programmed to perform functions not intended by the signer.

Sage Software Canada Ltd. will not digitally sign JavX as an applet since JavX is designed to respond and act upon commands initiated by the host application. Since Sage Software Canada Ltd. has no control over the host application, we are unable to guarantee that JavX will not adversely effect the operations of the workstation or use the information contained on the workstation inappropriately.

A ProvideX developer responsible for the server-side programming that controls the operation of the JavX applet may elect to digitally sign the JavX applet themselves for use with their application.

HTML Reference

When JavX is launched from a website, both the HTML and the applet must be made available via a web server that has the correct mime types for Java files.

File Extension	*Mime Type*
`.jar`	application/octet-stream
`.class`	application/x-java-applet

The client's browser must support Java applets in some form. However, since an applet is rendered by the browser, there is no easy way to interrupt the browser and force it to use JRE. Therefore, implementing support for the various combinations of browsers and platforms can be a bit complicated; i.e.,

- If a browser has built-in support for Java 2 (Netscape 5 or higher, Mac OS X with MS IE) or the JRE is version 1.4 or higher, then the **<APPLET>** tag can be used within the web page.
- If a browser does not have built-in support for Java 2 (Windows IE) or the JRE is version 1.3 or lower, then the **<OBJECT>** tag reference must be used for defining the Java applet.
- If a browser supports neither the <APPLET> tag nor the <OBJECT> tags for Java 2 applets, (Netscape prior to Version 5), then the **<EMBED>** tag must be used to reference a Java applet.

It should be noted that, while Netscape supports the <APPLET> tag directly, it also still supports the <EMBED> tag.

Note: Further discussion on the tagging structures required for using the Java plug-in (<OBJECT> ,<EMBED> , or <APPLET> tag) can be found on the java.sun.com website.

The following sections explain the HTML that can be used to create a location on the page for a Java applet (JavX in particular). Some sample code is provided that may be used within the web page to determine whether or not the user has a Java 2 Run-Time environment installed. There is also code to automatically download the correct Java 2 Run-Time for the particular OS that the client is using on his workstation.

Applets and HTML

This section provides examples and background information on how to manually insert the appropriate <OBJECT> ,<EMBED> , or <APPLET> tags in an HTML page to download a Windows version of the JRE. The object references define a location on a web page for a Java applet, as well as the characteristics of that applet.

Note: The **JavX Developer's Kit** includes an application that automatically builds a web page that runs JavX as an applet, the **Web Page Generator**, *p.70*. We recommend that you use this application instead of manually inserting the formats described below.

The following tagging examples are browser-specific, but not operating system specific. Therefore, in order to use the same code for multiple operating systems where different JRE installations are required, you must determine the browser and the Operating System the browser is running on.

The parameters shown are the minimum parameters that must be used.

<APPLET> *Tag Settings*

```
<APPLET code="NetworkClientApplet.class"
width=800 height=600 align="baseline">
<PARAM NAME="archive" VALUE="JavX.jar">
<PARAM NAME="code" VALUE="NetworkClientApplet.class">
<PARAM NAME="args" VALUE=" server= www.company.com;
          program= D:\IQ\config\go;
          port=20000; ">
</APPLET>
```

The <APPLET> tag is supported by all browsers that natively support Java 2 or on any machine where the Java 1.4 plug-in is installed. *Name* parameters (i.e., `Codebase` or `Pluginspage`) are not required if the browser supports Java 2 applets directly. Refer to the **HTML Parameters** chart for more information.

<OBJECT> *Tag Settings*

```
<OBJECT Classid="clsid:8AD9C840-044E-11D1-B3E9-00805F499D93"
Width="800" Height="600" Align="baseline"
Codebase="http://www.mydomain.com/downloads/JRE/jre-1_2_2_006-win.exe">
<PARAM NAME="Archive" VALUE="JavX.jar">
<PARAM NAME="Code" VALUE="NetworkClientApplet.class">
<PARAM NAME="Type" VALUE="application/x-java-applet;version=1.2.2">
<PARAM NAME="args" VALUE=" server= www.company.com;
      program= D:\IQ\config\go;
      port=20000; ">
</OBJECT>
```

The <OBJECT> tag is supported by all Window platforms using Microsoft Internet Explorer. Refer to the **HTML Parameters** chart for more information.

<EMBED> *Tag Settings*

```
<EMBED type="application/x-java-applet;version=1.2.2"
width="800" height="600" align="baseline"
archive="JavX.jar"
code="NetworkClientApplet.class"
pluginspage="http://www.mydomain.com/downloads/JRE/jre1_2_2-001-win.exe"
args= " server= www.company.com;
      program= D:\IQ\config\go;
      port=20000; ">
<NOEMBED>
      No Java Run-Time Environment support for JavX
</NOEMBED>
</EMBED>
```

The <EMBED> tag is supported by Netscape prior to Version 5. `Pluginspage` is the equivalent to `Codebase` in the <OBJECT> tag, and determines which download to install if the mime type specified does not exist within the client browser's mime definitions. The HTML parameters are described below.

HTML Parameters

JavX-specific parameters and values must appear within the applet reference of an <APPLET>, <OBJECT>, or <EMBED> tag within an HTML page. If necessary, you can also add your own parameters to the object; e.g., <PARAM name="*parmname*" value="*data*"> can be retrieved later via `FIN(0,"GetParam` *parmname*`")`.

Parameter	*Description*
`Classid`	**<OBJECT>** ***tags only.*** This is the object class identifier used by the browser to determine if there is a software package installed that can execute the object. In the example above, the class ID is the JRE class ID. (This should not be changed.) When the browser attempts to use the class, it determines if the class ID is installed. If so, then it runs the `Code` associated with it. If not, it will run the `Codebase` to find the software to install to support the object.
`Width`	***For all formats.*** Width (in pixels) of the applet area on the web page. Minimum value is 1.
`Height`	***For all formats.*** Height (in pixels) of the applet area on the web page. Minimum value is 1.
`Align`	***For all formats.*** HTML alignment name indicating how to align the area to its location on the web page.

Parameter	Description
Codebase	**<OBJECT>** *tags only.* URL to download if the class ID has not been installed. In the example above, it would download the JRE for Windows from the specified web server. Either make the run-times available from your own web server, or use a URL which would download the runtime directly from java.sun.com.
Pluginspage	**<EMBED>** *tags only.* Equivalent to Codebase above.
Archive	*For all formats.* Name of Java Jar file, in this case JavX.jar.
Code	*For all formats.* Name of the main class file within the jar file to run. For JavX it is always NetworkClientApplet.class.
Type	*For all formats.* Mime type of the object. This is always application/x-java-applet;version=*V#*.
args	*For all formats.* Variables for running JavX. All *required*, *optional*, and *Application Server* arguments are described below.

Required Arguments:

server=	Either the IP address of the host system where the *NTHost or *appserv is running or the host name of the server.
port=	TCP/IP socket number that the *NTHost or *appserv is monitoring.

Optional Arguments:

program=	Program to run on the server. Either one of the following: • Program name; e.g., *nomads would launch a ProvideX session and run NOMADS automatically. • Portion of the ProvideX command line that the server executes from the lead program to the end of the command line; e.g., C:\MY_DIR\MY_PROG.EXE -ARG...
onapplet=	Boolean: true or false (*default* false). onapplet=true indicates that JavX should put windows ***in*** the browser. onapplet=false indicates that JavX should put windows ***on top*** of the browser.
LookAndFeel=	GUI appearance other than default system's look-and-feel.
set_mf=	Multiline factor (default value is 50).
fontsize=	Text plane font size (default value is 12).
basewinwidth=	Base window's width (default value is 80).
basewinheight=	Base window's height (default value is 25).
ConnectString=	String to send to the server daemon in place of the standard string sent by *NTHost or *appserv .

Application Server Arguments:

`applicationserver=`	Boolean: `true` or `false` (*default* `false`). `applicationserver=true` indicates that JavX will be connecting to the ProvideX application server.
`login=`	Boolean: `true` or `false` (*default* `false`). `login=true` indicates that JavX should attempt to login to the ProvideX application server prior to launching the ProvideX session.
`clientFID=`	Value of the FID(0) the client wants the server to use.
`clientstartindirectory=`	Directory where the server will run the application.
`clientcmdoptions=`	Extra command line options to be passed to the server.
`clientarguments=`	Additional command line arguments (`-arg`) to be passed to the server.
`SSL=`	Boolean: `true` or `false` (*default* `false`). `SSL=true` indicates that JavX should encrypt communication with the ProvideX Host.

HTML Example for Detecting OS and Browser Type.

The following example uses JavaScript within a single web page to determine the user's operating system and browser. The correct `<OBJECT>` ,`<EMBED>` , or `<APPLET>` tags can then be specified for the operating system, such that the correct JRE will be downloaded. This does not deal with all possible operating systems, but provides a good starting point:

```
<SCRIPT LANGUAGE="JavaScript"><!--
var _info = navigator.userAgent; var _ns = false;
var _ie = (_info.indexOf("MSIE") > 0 && _info.indexOf("Win") > 0 &&
       _info.indexOf("Windows 3.1") < 0);
</SCRIPT>
<COMMENT><SCRIPT LANGUAGE="JavaScript1.1"><!--
var _ns = (navigator.appName.indexOf("Netscape") >= 0 && (
       (_info.indexOf("Win") > 0 && _info.indexOf("Win16") < 0 &&
       java.lang.System.getProperty("os.version").indexOf("3.5") < 0) ||
       (_info.indexOf("Sun") > 0) || (_info.indexOf("Linux") > 0)));
</SCRIPT></COMMENT>
<SCRIPT LANGUAGE="JavaScript"><!--
if ( _ie == true && _info.indexOf("Win") > 0 )
       document.writeln('<OBJECT
       classid="clsid:8AD9C840-044E-11D1-B3E9-00805F499D93"
       width="1" height="1" lign="baseline"
       codebase="http://www.mydomain.com/download/plugins/jre-1_2_2_006-win.exe">
       <PARAM NAME="archive" VALUE="JavX.jar">
       <PARAM NAME="code" VALUE="NetworkClientApplet.class">
       <PARAM NAME="type"
      VALUE="application/x-java-applet;version=1.2.2">
       <PARAM NAME="args" VALUE=" server= www.company.com;
                program= D:\IQ\config\go;
```

```
                port=20000; ">
       <NOEMBED><XMP>');
else if (_ns == true && _info.indexOf("Linux") > 0 )
       document.writeln('<EMBED
      type="application/x-java-applet;version=1.2.2" width="1" height="1"
      align="baseline" code="NetworkClientApplet.class" archive="JavX.jar"
      pluginspage="http://www.mydomain.com/downloads/plugins/jre-1_2_2_006-w
      in.exe" args= "server= www.company.com;
                program= D:\IQ\config\go;
                port=20000; ">
       <NOEMBED><XMP>' );
else if (_ns == true && _info.indexOf("Win") > 0 )
       document.writeln('<EMBED
      type="application/x-java-applet;version=1.2.2" width="1" height="1"
      align="baseline" code="NetworkClientApplet.class" archive="JavX.jar"
      pluginspage="http://www.mydomain.com/downloads/plugins/jre-1_2_2_006-w
      in.exe" args= "server= www.company.com;
           program= D:\IQ\config\go;
           port=20000; ">
  <NOEMBED><XMP>' );

else if ( _info.indexOf("Mac") > 0 )
       document.writeln('
       <applet code="NetworkClientApplet.class" width=1 height=1 align="baseline">
       <PARAM NAME="archive" VALUE="JavX.jar">
       <PARAM NAME="code" VALUE="NetworkClientApplet.class">
       <PARAM NAME="args" VALUE=" server= www.company.com;
                program= D:\IQ\config\go;
                port=20000; ">
</applet>' );
</"file://-->
</SCRIPT>
```

Setting Focus to the Applet Window

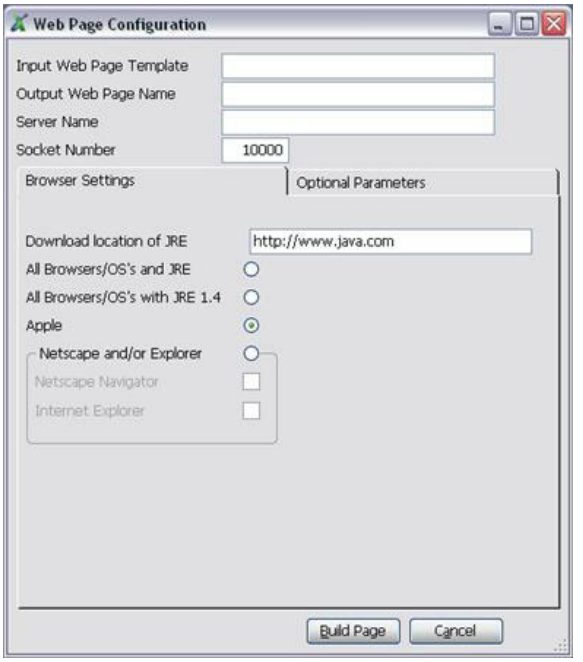
The following example sets the focus to the JavX applet when the JavX applet base window is part of the browser window (when `OnApplet="true"`). The applet is placed within a form, and this allows the forms methods to force focus to the applet itself. Note the need to name the applet (`NAME="JavXApplet"`) and the extra parameters to allow it to be scriptable. With the `<APPLET>` tag, you would have to include a `<PARAM NAME=SCRIPTABLE VALUE="true">`.

```
<form id=hide>
      <OBJECT classid="clsid:8AD9C840-044E-11D1-B3E9-00805F499D93"
       NAME = "JavXApplet" width="800" height="600" align="baseline"
     codebase="http://www.mydomain.com/downloads/plugins/jre1_2_2-001-win.exe">
       <PARAM NAME="SCRIPTABLE" VALUE="true">
       <PARAM NAME="archive" VALUE="JavX.jar">
       <PARAM NAME="code" VALUE="NetworkClientApplet.class">
       <PARAM NAME="type" VALUE="application/x-java-applet;version=1.2.2">
       <PARAM NAME="args" VALUE=" server= www.company.com;
                program= D:\IQ\config\go;
```

```
                    port=20000; ">
          <COMMENT>
           <EMBED type="application/x-java-applet;version=1.2.2" width="800"
           height="600" align="baseline" code="NetworkClientApplet.class"
         archive="JavX.jar"
           pluginspage="http://www.mydomain.com/downloads/plugins/jre1_2_2-001-win.exe"
           args= "server= www.company.com;
                    program= D:\IQ\config\go;
                    port=20000; ">
               <NOEMBED>
                    No JDK 1.2 support for JavX
               </NOEMBED>
           </EMBED>
           </COMMENT>
           </OBJECT>
           <INPUT style="border: 0px;" ReadOnly id=xx maxLength=0 name="JavXField"
         type=text notab>
</form>

<SCRIPT LANGUAGE="JavaScript">

function focusToJavXApplet ()
{
// This pulls focus away from the applet so the next stuff can
// put it back. It seems without this every second try does not
// restore focus to the applet
hide.JavXField.focus();
document.JavXApplet.jsForceFocus();
document.JavXApplet.requestFocus();
}

window.onfocus = focusToJavXApplet;
</SCRIPT>
```

Web Page Generator

A convenient easy-to-use GUI utility is included with the **JavX Developer's Kit** (JDK) to automate the process of embedding JavX in web pages. The JDK is freely downloadable from www.pvx.com.



The correct HTML (and sometimes JavaScript) for configuring JavX is largely dependant on the browser, the operating system, and the Java Runtime Environment present on the client machine. This GUI utility creates the necessary tagging structures using data collected from a central panel. The contents of the fields (web page name, browser/OS type, JRE location, server information, etc.) are assembled into the appropriate HTML tags and parameters then inserted automatically into a web page template.

Once the JDK (plus JavX) is installed, `cwdir` to `JavX_PageGen` from the JavX console and `run JavXpagegen.pvs` to launch the Web Page Configuration program. The central panel is divided into two panels for browser settings and optional parameters, as shown on the previous page.

Full details on the use of Web Page Configuration program can be found in the *Web Page Generator Tutorial* that is installed with the JDK.

Troubleshooting JavX SE

If you are new to JavX and the Java environment, we recommend that you download the **JavX Developer's Kit**, before you get started. We also recommend that you run JavX as an application initially to better understand how the product works. The following sections discuss some the of problems you may encounter.

Problems Launching JavX as an Applet

If you have generated a web page using the **Web Page Generator**, and the JavX applet fails to load or run, follow these steps to determine the cause and resolve the problem:

1. Make sure that Java 1.4.2 or higher is installed on the client machine.
2. Make sure that a running web server has access to the HTML page the JavX applet is embedded on, as well as the `JavXSE.jar` file. Ideally the HTML page and JAR should be in the same directory.
3. Make sure that the ProvideX host is running normally (either the ProvideX Application Server or *NThost).
4. Run the Java Plug-in Control Panel. ***On Windows***, select Start > Settings > Control Panel > Java Plug-in, then select the Java console's Show console radio button. ***On Mac OS X***, select Finder > Applications > Utilities > Console.
5. Restart the web browser. When the web browser encounters a Java applet it should now show the Java Console. The Java Console provides detail on errors encountered while loading and running JavX.

Errors Reported in the Java Console

The following list provides some instructions for resolving errors reported in the Java Console:

java.security.AccessControlException: access denied
: Make sure the IP address specified on the HTML page in the "`server=`" matches the domain/IP address specified in the web browser's URL address text field. For example, if an HTML page called `JavX.HTML` has an `"args"` parameter that indicates `"server=localhost"`, then the web browser URL text field must indicate `http://localhost/JavX.HTML`.

java.net.ConnectException: Connection refused: connect
: Check that a ProvideX host program is running and accessible at the location specified in the HTML page's `"args"` parameter. Check that a firewall is not blocking the JavX connection.

java.lang.ClassNotFoundException: NetworkClientApplet.class
: If you see this message, perform the following:
  - Make sure a web server is serving up the HTML page - do not just double click on the HTML page in Windows Explorer and expect JavX to run.

- Confirm that the web server has access to the `JavXSE.jar` file.
- Confirm the web server has the mime type configured for Java classes and jars (older versions of the ProvideX WebServer did not ship with these mime types pre-configured).

java.lang.ClassNotFoundException: java.io.FileNotFoundException

Typically this is related to an improper mime type. The web server you are using does not know how to transfer a Jar or Class file, nor can it inform the browser of the correct mime type for the transferred file. The browser, therefore, does not understand how to handle the file. See the HTML reference section of this document for more on adding the correct mime type.

JavX for Portable Devices

In recent years, Java has become the universal runtime environment for distributing interactive software to a wide variety of consumer devices. It is responsible for bringing advanced functionality to a wide range of products that includes personal digital assistants (PDAs), mobile phones, pagers, and similar embedded computers. However, due to limitations in the device-oriented Java runtime, the traditional client-server architecture found in **JavX SE** is not ideally suited for this market.

Instead, two JavX editions have been developed to allow ProvideX client-server applications to be run on mobile devices: **JavX AE** (AWT Edition) and **JavX LE** (Light Edition). These JavX products are adapted for the *Java 2 Micro Edition (**J2ME**)* specification, a Java version that has been optimized for constrained/embedded implementations.

Note: JavX LE is currently not available for direct download; however, developers are welcome to contact Sage Software Canada Ltd. to request a copy if they are interested.

For a discussion on the full range of client-server options, refer to the section **Choosing the Right Solution**, *p.10*.

Runtime Requirements for JavX AE/LE

The JavX AE and JavX LE thin-clients are designed for the Java 2 Micro Edition (**J2ME**) **Java Runtime Environment** (*JRE*) which involves several specifications for addressing the diverse needs of a variety of portable devices. This section discusses J2ME and explains the core concepts behind device-oriented Java.

Note: SSL is an optional component in the J2ME package, and not all vendors include these classes in their runtime. If SSL classes are omitted, and SSL is used, JavX will terminate.

About the J2ME Specification

The Java Community Process (JCP) is a process whereby a wide range of hardware and software vendors come together to agree on specifications and standards for new and existing Java APIs. J2ME is a result of the JCP agreeing on configurations and specifications mobile devices and software vendors can support.

According to Sun: *"The Micro Edition of the Java 2 Platform provides an application environment that specifically addresses the needs of commodities in the vast and rapidly growing consumer and embedded space, including mobile phones, pagers, personal digital assistants, set-top boxes, and vehicle telematics systems."*

Currently, Java-based handsets dominate the market for mobile applications. Several major players in the handheld market (*Nokia, Ericsson, Motorola, RIM, IBM*, etc.) have chosen Java as their preferred development environment and devices from the manufactures come with a J2ME JRE installed.

J2ME Configurations and Profiles

J2ME supports a vast array of hardware and operating systems through an abstraction known as a *configurations* and *profiles*. This system of configurations/profiles benefits device manufacturers and application programmers in a number of ways:

- *Device Manufacturers.* Manufacturers choose the J2ME configuration and profile for their devices, or include a compliant JRE.
- *Application Programmers.* When writing an application, developers can target for the J2ME configuration and profile rather than a specific device.

Configurations define minimum platform requirements for a group of devices by considering processor power, memory available, and screen size. There are currently two J2ME Configurations: *Connected Device Configuration* (CDC) and *Connected Limited Device Configuration* (CLDC). The following table compares the two:

Connected Device Configuration (CDC)	*Connected Limited Device Configuration* (CLDC)

• Qualifiers
  • Faster Processor
  • More Memory
  • Faster Network connection
	• Qualifiers
	  • Minimal Processor power
	  • Minimal Memory
	  • Network connection

• Example
  • Pocket PC PDA
  • TV Set-top devices
  • In-vehicle system
	• Example
	  • Mobile Phones
	  • RIM Blackberry
	  • Palm PDAs

• Hardware
  • 16 or 32 bit CPU with 128 or 512 KB of memory.
	• Hardware
	  • 16 or 32 bit CPU with 128 or 512 KB of memory.

Configurations are divided into ***Profiles***, higher level APIs that define the user interfaces available and access to device hardware. Profiles, together with configurations, provide a complete JRE specification for a targeted group of devices; e.g., the *CDC Personal Profile* and the *CLDC Mobile Information Device Profile*.

CLDC Mobile Information Device Profile (MIDP)

The Mobile Information Device profile extends the CLDC configuration and provides a complete specification for J2ME JREs for small devices with limited network speed and a very basic user interface.

MIDP JREs are found typically on cell phones and on entry level personal digital assistants (PDAs); e.g., RIM includes a CLDC MIDP JRE. A MIDP device must meet the following minimum requirements:

- A minimum screen size of 96x54
- 32kb of memory for JRE
- Network Connection.

The Mobile Information Device profile defines and requires the following core functionality for the device:

- User Interface (UI)
- Network connectivity
- Local file access
- Application life-cycle control.

CDC Foundation Profile

The Foundation Profile extends the CDC configuration and provides a complete specification for J2ME JREs for embedded devices with a network connection but no user interface. The Foundation Profile device must meet the following minimum requirements:

- 1024kb of ROM
- 512kb of RAM
- Network Connection

Example devices include network printer, router, residential gateways.

CDC Personal Basis Profile

The Personal Basis Profile extends the CDC configuration and provides a complete specification for J2ME JREs for embedded devices with a network connection and a very basic user interface. Example devices include *interactive television*, *automotive components*, *fixed consumer devices*.

CDC Personal Profile

The Personal Profile extends the CDC configuration and provides a complete specification for J2ME JREs for devices with a network connection and a full Graphical User Interface (GUI). Personal Profile JREs resembles the standard edition of Java found on desktop PCs. Personal Profile devices require at least:

- 2.5 Mb of ROM
- 1 Mb of RAM
- Network Connection

Example devices include handheld bar code scanners, game consoles, and high-end PDAs such as, *iPAQ Pocket PC*.

WebSphere Everyplace Micro Environment (WEME)

While the J2ME specification can be considered a *universal standard*, there are almost as many runtime environments as there are device manufacturers. Some pre-package their devices with a J2ME runtime environment installed. Some deliver their JREs on a "companion CD" along with various supplementary utilities. Others, such as the RIM Blackberry, have a J2ME JRE built right into the device's OS. A search for "J2ME Java Virtual Machines" on the internet will return dozens of different vendors.

JavX was development using IBM's *WebSphere Everyplace Micro Environment (WEME),* a production-ready **Java Runtime Environment** that has been tested/certified to meet J2ME specifications. It comes pre-installed on several devices, but it can also be purchased for under $10. The evaluation version (no expiration) can also be downloaded free of charge. WEME downloads are available at the following locations:

IBM. WEME is included with a suite of tools called *IBM Workplace Client Technology, Micro Edition 5.7* (free evaluation software). Go to www.ibm.com/software.

Handango. WEME is featured under the *Development Tools* category, and may be downloaded for a nominal price. Go to www.handango.com.

WEME supports several different J2ME configurations and profiles depending on the platform; e.g.,

- CDC Foundation Profile, and Personal Profile for more powerful PDAs, Bar Code Scanners, etc.
- CLDC Mobile Information Device Profile (MIDP) for limited PDAs.

JavX AE and ***JavX LE*** should operate in any runtime environment that implements the J2ME CDC specification. However, because JavX AE has been tested against IBM's WEME runtime environment, we currently only offer support for JavX AE running in WEME.

Installing JavX AE/LE on a Portable Device

JavX AE and JavX LE **JAR** files can be installed on a wide variety of devices — in fact, any device with a J2ME CDC JRE. This document cannot provide the installation instructions required for every conceivable J2ME device on the market; therefore, the following procedure will concentrate on one of the more popular platforms, Windows CE (Windows Mobile for Pocket PC).

Example Installation Procedure for Windows CE

The following instructions cover the installation of JavX AE (or JavX LE) on an iPAQ running Windows Mobile 2003. Similar procedures work for most CE devices; e.g., Symbol Bar code scanners.

1. Ensure that your device is docked in its cradle and that the cradle is connected directly to a desktop computer that is running Microsoft ActiveSync.
2. Download IBM's JRE for Windows Mobile 2003 on to the desktop computer. Vendor download sites are described under **WebSphere Everyplace Micro Environment (WEME)**, *p.75*. This may be a large download because it includes numerous additional tools.
3. Unzip the downloaded file to a directory on your desktop and run the installation/setup program provided. The installation menu may include a variety of JREs. Install *WEME Personal Profile 1.0* for Windows Mobile 2003.

4. Using Windows *Explorer* on the desktop, navigate to the directory that comprises the J9 runtime environment on the connected device. Create a shortcut for the `J9.exe` file and rename it to JavXAE (*or JavXLE*).

5. The contents of the new shortcut can be viewed/edited by dragging the JavXAE icon into Windows *Notepad* on the desktop. It file should contain a string similar to `147#"\Program Files\J9\PPRO10\bin\j9.exe"`. Append the following text (without line breaks) to this string, then save the file:

   ```
   -classpath \JavX\JavXAE.jar -jcl:ppro10 localNetworkClientTest
   "server=myServer; port=10000; fontsize=10; "
   ```

   The last string (`"server=myServer; port=10000; fontsize=10; "`) is the argument string (*ArgString$*) that will be passed to JavX at start up. Change the `"server=myServer"` and the `"port=10000"` arguments to reflect the actual server and port number that the host is running on. When running the ProvideX Application Server, add `"applicationserver=true"` to the *ArgString$*.

   For more information on the *ArgString$* see the section **Launching JavX as an Application**, *p.46*. Alternatively, omit the *ArgString$* and JavX will retrieve the start up arguments from the `JavX.properties` file. For more information on the `JavX.properties` file, see the section **Launching JavX Clients without Arguments**, *p.47*.

6. Using Windows *Explorer* on the desktop, create a directory called JavX in the root directory of your device. Copy the `JavXAE.jar` into the new JavX directory.

7. Copy the edited JavXAE shortcut file into the device's \Windows\Start Menu\Programs directory.

8. On the device, select Start > Programs > JavXAE to launch the JavX ProvideX console.

JavX Windows CE Configuration Wizard

The JavX Windows CE Configuration Wizard program uses the ProvideX WinCE Link object to automate the manual installation steps covered in the previous section **Example Installation Procedure for Windows CE**, *p.76*. This wizard steps through the following to simplify the setting up and running of JavX AE/LE on a CE device:

- Collecting arguments required by JavX and creating a shortcut that runs JavX in IBM's WEME Personal Profile JRE.
- Creating the JavX directory on a CE device and copying the JavX AE/LE JAR file into it.
- Copying the shortcut (created in step one) to the specified directory on the CE device.
- Checking the CE device's registry to determine if IBM's WEME JRE is installed. If it is not installed, the JRE installer's CAB file is copied to the device and the installer is run.
- Running JavX on the CE device with the arguments specified.

JavX AE Deployment and Functionality

JavX AE (*AWT Edition*) is designed to run on devices that support the Java 2 Micro Edition (**J2ME**) specification. It requires at least a **CDC Personal Profile** JRE and is considered to be the JavX edition best suited for use in high-end portable devices, such as PDAs and mobile phones. This version of JavX employs the Abstract Windowing Toolkit (AWT), a lightweight set of graphical elements used to build GUIs in Java programs. JavX AE is more accessible than JavX SE, but it provides a slightly less expressive graphical interface.

JavX AE is normally deployed on a handheld device running an OS such as Pocket PC or Windows CE. Because J2ME ***Foundation*** and ***Personal Profiles*** inherit a large subset of the J2SE core API, JavX AE implementations are upwardly compatible with **JavX for PC Platforms**: Apple OS X, UNIX, Linux, and MS Windows 9x,NT,2000,XP.

Available Features

For a general overview of the features available in *all versions* of JavX, refer to the sections listed under **Common Functionality and Limitations**. Since JavX AE is built on the same core classes as **JavX SE**, the same limitations also apply to JavX AE.

As mentioned earlier, the primary difference between JavX AE and JavX SE is that JavX SE uses the Java Swing library of GUI components and JavX AE uses the Java AWT library of GUI components. Java AWT offers a less rich GUI environment, but also has a smaller footprint than the Swing library.

Features not Supported in JavX AE

JavX AE has the following limitations in addition to those described under **ProvideX Features not Supported in JavX**, *p.48*:

1. ***Image Support.*** Images cannot be added to control objects. For example, the stop image below will be ignored:

   ```
   BUTTON 10,@(10,10,10,10)="OK{!stop}"
   ```

2. ***Control Properties***. JavX AE supports a subset of the list of properties, including: **Auto**, **BackColour**, **BackColor**, **Col**, **Cols**, **CtlName**, **Enabled**, **Eom**, **Focus**, **Font**, **Height**, **hWnd**, **Key**, **Left**, **Line**, **Lines**, **Parent**, **TextColour**, **TextColor**, **Tip**, **Top**, **Value**, **Visible**, **Width**, **Msg**, **OnFocusCtl, Id**.

3. **LIST_BOX.** JavX AE only supports standard list boxes. ***Formatted***, ***Treeview***, and ***Listview*** list boxes are not supported. The **LIST_BOX OPT=** options **"~"** (*no height adjustment*), **"B"** (*no border or frame*), **"d"** (*permanently disabled*), and **"h"** (*permanently hidden*) are not supported. Note that **"D"** (*initially disabled*), and **"H"** (*initially hidden*) are supported.

4. **MSGBOX.** A maximum of two buttons can be added to a message box and images are ignored; e.g.,

   ```
   MSGBOX "Click please","INFO","YESNOCANCEL,?",X$
   ```

This example creates a message box with a question mark icon and three buttons with the following text `"YES","NO","CANCEL"` in ProvideX and JavX SE. In JavX AE, a message box with no icon and two buttons with the text `"YES", "NO"` will be created.

5. ***Global Controls.*** JavX AE does not support global controls. In ProvideX, some GUI components can be declared global at creation; e.g.,

   ```
   BUTTON *10,@(10,10,10,10)="ok"
   ```

   The ***** *asterisk* indicates this button is global and can always be accessed even when the window the button is on does not have focus. JavX AE runs on devices with small screen sizes that generally support one window at a time. For example, on Pocket PC only one dialogue will be visible at a time. A dialogue receiving focus will completely hide any other dialogue, therefore global controls are not necessary because a control on a dialogue without focus will not be visible.

6. ***Grids.*** JavX AE does not support grids.

7. **BUTTON**, **RADIO_BUTTON**, **CHECK_BOX.** Most **OPT=** options are ***not supported***. Only **"D"** (*disabled*), and **"H"** (*hidden*) are supported.

8. **MULTI_LINE.** Formatted multi-lines are not supported; e.g., a format mask specified by **FMT=**"*mask$*", is ignored in JavX AE and alpha numeric characters could be entered by an end user. Applications requiring format masks on multi-lines must apply the format on the server side. Only the **OPT=** options **"$"** (*Password $ Mask)*, **">"** (*Include horizontal scrollbar)*, **"A"** (*Auto*), **"t"** (*tab key*), **"T"** (*strip trailing spaces*), **"D"** (*initially disabled*), and **"H"** (*initially hidden*) are supported.

9. **'DIALOGUE'** ***and/or*** **'WINDOW'** ***Mnemonics.*** Only the **OPT=** options **"c"** (*child of current window)*, **"h"** (*no title bar)*, **"i"** (*no icon in the upper left corner)*, **"m"** (*enable "maximize" box in top right corner)*, **"M"** (*window has a menu bar.)*, **"s"** (*return CTL value on state change)* are supported.

10. **MENU_BAR**. JavX AE does not directly support **MENU_BAR** items that do not have a sub-menu. Sub-menus are created by duplicating the main item. For example, the definition `menu_bar menuctl,"-[&File,&Edit],F:[&Open,&Close]"` results in an menu item "Edit" which has a sub-menu item of "Edit".

JavX LE Deployment and Functionality

JavX LE (*Light Edition*) is designed to run on fixed-purpose embedded devices that support the Java 2 Micro Edition (**J2ME**) specification. It requires only a J2ME **CDC Foundation Profile**. While JavX LE has no built-in user interface support, it does have ***Java reflection support*** through the ProvideX OCX interface. If there are GUI classes available on the client device, ProvideX applications can create a GUI via the ProvideX OCX interface.

The primary purpose of JavX LE is to provide access to the local file system and JRE classes via ProvideX OCX support and it is ideally suited for deployment on an embedded system that has a limited user interface; e.g., a network printer or residential gateway.

JavX LE can also be used on any device where JavX serial file access, or the ProvideX OCX interface is required, but a user interface is not. For example, it could be used to load a Java Database Connectivity (JDBC) driver to read from a MySQL or Oracle database (JDBC drivers are available for most databases). Another use for JavX LE may be for the purpose of copying files to/from a PDA, such as MS ActiveSync. JavX LE will run on any J2SE or J2ME CDC-enabled platform, which includes any platform where JavX AE and JavX SE are currently supported.

JavX LE implementations are upwardly compatible with Java-enabled devices (Pocket PC, Windows CE) and on **JavX for PC Platforms** (Apple OS X, UNIX, Linux, and MS Windows 9x,NT,2000,XP).

Available Features

For a general overview of the features available in all versions of JavX, refer to the sections listed under **Common Functionality and Limitations**. JavX LE is built on the same core classes as JavX SE and JavX AE, much of the same functionality and limitations will apply to JavX LE. Since JavX AE is built on the same core classes as JavX SE, the same limitations also apply to JavX AE.

Features not Supported in JavX LE

JavX AE has the same limitations as those described for JavX AE. See **JavX AE Deployment and Functionality**, *p.78*. The primary difference between JavX LE and other versions of JavX is that JavX LE does not have a built-in user interface. There is no character-based, or graphical user interface built into JavX LE.

5
Application Server

The ***Application Server*** is a ProvideX add-on product that allows you to create and maintain a secure **TCP/IP** server environment for connecting WindX and JavX-based implementations via MS Windows, UNIX/Linux, and MAC OS X.

This chapter documents the installation, configuration and functionality of the ProvideX Application Server.

Topics

Enhanced Hosting Facility

This software provides an enhanced, configurable alternative to the built-in client-server processes supplied with the ProvideX base system. While the ***NTHost/*NTSlave** defaults are sufficient for establishing quick and simple client-server connections, they were never designed for use "as is" outside of a closed local area network. Developers who intend to expose their applications to the outside world would endeavor to build and/or acquire additional infrastructures on top of *NTHost/*NTSlave in order to reach the minimum security and administration requirements.

In contrast, the Application Server delivers an all-in-one ProvideX solution for protecting and maintaining your data in adverse network environments – *particularly the Internet:*

- ***Simplified Interface.*** User-friendly utilities are provided for creating, configuring and administering the different characteristics of your ProvideX client-server applications..

- ***Uses only one TCP /IP Socket.*** Default client-server processes in ProvideX can end up using many different sockets depending on the implementation and the number of clients. The Application Server architecture allows you to direct all connections through a single socket.
- ***Designed for firewalls***. If it is too difficult for a firewall to determine which sockets it needs to block, it might as well be disabled. By reducing the number of sockets needing access, the Application Server ensures that your firewall is fully operational.
- ***Session Adminstration.*** One of the primary security features in the Application Server is its ability to monitor and control access through session administration, user authentication, and limiting client access.
- ***Optional SSL encryption***. The Application Server supports use of a TCP/IP-level Secure Socket Library, a security protocol that allows you to encrypt communications.

Activation and Product Components

The Application Server is shipped with ProvideX and it may be activated for use as a part of the ProvideX eCommerce bundle, or as a separate add-on package. In either case, use of this product requires a secondary-level activation on top of the base system activation.

While the program shortcuts and configuration panels are immediately available and can be accessed from a base system installation of ProvideX, the Application Server itself cannot be run unless it has been properly activated. If the activation has been recorded incorrectly, the session will fail, returning the error message: `NO VALID ACTIVATION FOUND!!`.

Note: Contact your dealer/distributor or visit the ProvideX website at www.pvx.com for more information on product options and licensing.

The following software components and supporting files are required to run the Application Server, and should be installed within the server system's ProvideX directory:

`*appserv/server`	Application Server daemon.
`*appserv/config`	Configuration facility
`*client`	Client-side software (included with WindX and JavX)
`*server`	Spawning software
`*appserv/config.en`	NOMADS panel library
`*appserv/apsmesg.en`	Message library
`*appserv/dealer.en`	Dealer panel library.

Server Configuration

The ProvideX Application Server includes a utility for creating and setting up server characteristics, including security and administration. The Windows interface for the configuration utility is a NOMADS application that can be run locally via ProvideX for Windows or by using WindX or JavX connected to a UNIX/Linux server. A character-based method for running this utility is also available.

This utility may be launched from a shortcut supplied with the ProvideX for Windows installation. Otherwise, the command line syntax is described as follows.

In Windows: *PVXpath$* **[-hd] [apscfg.ini] *appserv\config [–ARG "AsService"]**

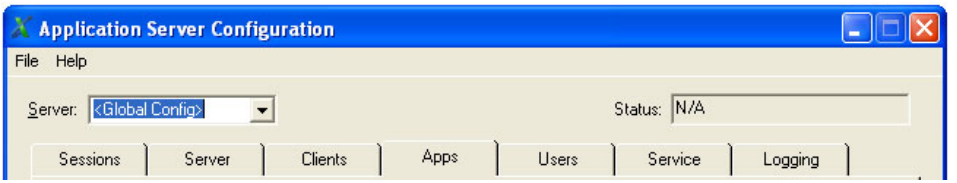
In UNIX/Linux: *PVXpath$* ***appserv/config**

Where:

PVXpath	Path to ProvideX
-hd	Command line option indicating that the utility starts with the initial window hidden. ***MS Windows only.***
apscfg.ini	Configuration's standard INI file. ***MS Windows only.***
AsService	Optional flag to the configuration that informs it not to allow start/stop on the desktop if server daemons are running as Windows services. ***MS Windows only.***

Main Console

Application servers are created and maintained from the main console of this utility.



Sessions, *p.104* **Server**, *p.86* **Clients**, *p.90* **Apps**, *p.93*
Users, *p.97* **Service**, *p.100* **Logging**, *p.102*

The console is divided into several tabbed panels for viewing and/or changing different configuration and administration options. *Panel components are documented by their tab name on the pages indicated in the list above.*

The main console is also where new servers are named and created and it is where the system administrator controls which server is currently selected to be started, stopped or updated.

The next few pages introduce the steps necessary for *creating, modifying, and deleting* an application server using this interface. The specific tab/panel descriptions and lists of options begin after these sections.

Creating a New Server

A new application server is created in this utility by entering a name in the Server field. All information about the new server will then be stored under this name, and it can be recalled later (by server name) for further editing. Once the server is saved, this name can also be used to start the server from the command line via the server daemon program (***appserv\server**).

Server names are case-insensitive and may comprise any legal characters up to 20 characters in length. However, you cannot use characters less than hex 20, spaces, nor any of the following: **" < > , / \ * ? & | ; : { } ( ) $! `** .

Modifying or Deleting Server Properties

If the configuration of any current server has been changed, you will be asked to save those changes before proceeding to configure the next server. To delete a server, select its name from the Server field drop list, then click Delete. All server names can be deleted unless they are currently running.

Any changes made to fields in the Server, Client, User, Application, or Logging panels are considered changes to the currently-selected server. To save these changes, either click the Apply button or switch to a different server name.

Updating On-The-Fly

If you save changes to a server while it is running, the new configuration will be applied automatically. Changes made under <Global Config> will apply to ***all*** servers that are currently running. The update will take place unless you opted to change TCP/IP properties, which results in a conditional update procedure explained below.

Because application servers play "man-in-the-middle", closing the TCP/IP socket will disconnect all connected users. If the server's TCP/IP properties have been changed, and the TCP/IP socket needs to be closed and re-opened, then the server will perform the following:

- If no users are currently connected, the server will simply be updated.
- If users are currently connected, you will receive an `"Ok to Continue (Yes/No)"` warning message.
  - If you proceed with the changes, then all users will be disconnected, the changes will be made, and the server will begin accepting new connections with the new configuration.
  - If you do not proceed, then the changes you made will be saved, but the currently running server will ***not*** be updated. To update it, you will need to either apply the changes later, or stop / start the server.

Switching Between Servers

Switch to another configured server by selecting a different server name from the Server field drop list, or by entering the name directly into the field. You can also switch from a specific server to define <Global Config>. As mentioned earlier, if you change any characteristics of the currently-selected server, you will be asked to save your changes before proceeding.

Starting a Server

- From the ***command line***, use the ***appserv/server** daemon program.
- From the ***configuration utility***, click the Start button that appears under the Server tab while the specific server name is currently selected.

Stopping a Server

- From the ***configuration utility***, click the Stop button that appears under the Server tab while the specific server name is currently selected.
- From a ***Windows OS***, select Close from the right mouse menu, when clicking on the server's taskbar button.

Under ***UNIX/Linux***, use `Kill -2 pid` (where `pid` is the server's process ID).

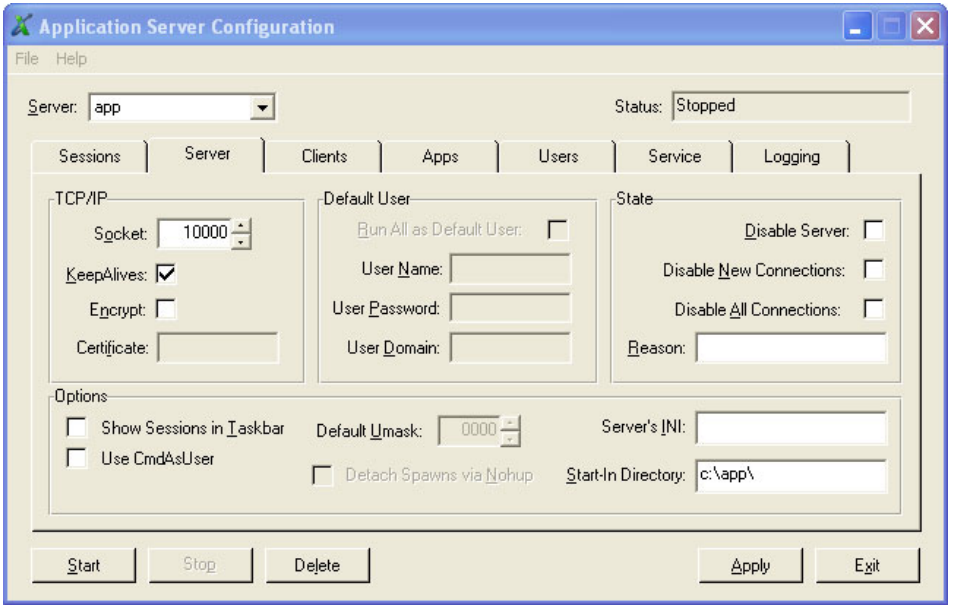
Server Status

Opposite to the Server field is the server Status indicator. This reports if the currently-selected server is Running or Stopped, Starting or Stopping, and when an Update is in progress. The status is only updated when:

- Moving from one tab to the next,
- Selecting the Refresh button on the Session tab,
- Attempting to Start or Stop,
- Clicking the Apply button.

Server Tab – Defining Server Attributes

The Server panel is where the primary server attributes are configured. You can also Start, Stop, Modify, Create, or Delete the currently-selected server. Refer to earlier sections under **Server Configuration** for general details on creating, deleting, and switching between servers.



TCP/IP Fields

Socket: TCP/IP port number (socket) that the server will open and listen for requests on. This socket will be used for all aspects of the server, from the initial connections through handling of session traffic. Select any valid TCP/IP port from 1 to 65535. Sockets 1 to 1024 are usually governed by the OS and only processes running as root, or users classed as *Administrators*, will be allowed to open sockets in this range.

KeepAlives: Checkbox for using TCP/IP KeepAlives on the socket. If this option is selected, the OS will send KeepAlive packets to the server- and client-side processes to check if they are still connected. The amount of time before sending KeepAlives, the number of KeepAlives sent, and the interval between sending them are configured at the OS level.

Encrypt: Checkbox for using SSL encryption on the socket. If this option is selected, the application server will only accept connections from clients who use SSL encryption. All traffic between the server and the client processes will be SSL encrypted. If on, a valid SSL certificate file must be entered (below).

Certificate: SSL certificate file to be used for SSL-encrypted communications. This field is only active if SSL encryption is turned on. The file must be in the standard format that ProvideX SSL requires, which is a plain text file containing both the X509 certificate and the private key.

Options Fields

Show Sessions in Taskbar ***Windows*** option that determines whether or not the client sessions that get launched show up as a button on the Window's task bar or not.

Use CmdAsUser ***Windows*** option that determines whether or not the application server is to use the ***CmdAsUser*** utility to launch sessions on the server as specific users.

Note: *CmdAsUser* is a third party program that allows Windows NT, 2000, and XP software to be run as a specific user. This utility is not supported by Sage Software Canada Ltd.

Default Umask: ***UNIX/Linux*** option for setting the umask to use when running programs that are not configured as applications. Applications that are configured will use their own setting.

Detach Spawns via Nohup ***UNIX/Linux*** option to indicate that any spawns from within this session will use `nohup`. The use of `nohup` determines if a process is to remain attached (or not) to the process that spawned it. If it is attached, then the child process will terminate when the session the parent is running has terminated.

This is the default setting for programs that are run, but not configured, as applications. Applications that are configured will use their own setting.

Note: If the following fields are not specified, then the INI file and start-in directory of the server daemon itself will be used instead, by default.

Server's INI: ***Windows*** option for providing the name of the INI file to use for the `*appserv/server` daemon. It is only required if you use the Start button to start the server daemons from within the configuration utility.

Start-In Directory: Directory where the `*appserver/server` daemon is to be started in. It is only required if you use the Start button to start the server daemons from within the configuration utility.

Default User Fields

These are for configuring default user characteristics for running server-side processes and are only applicable when running in a UNIX/Linux environment, or in MS Windows (in conjunction with **CmdAsUser**).

Run All as Default User: When this checkbox is selected, all client session requests will have their server-side process run as the user given in User Name. Any user name listed in the **Users Tab – Current Users Properties** will be ignored. This causes all ProvideX processes on the server to be run using the default user ID.

Under Windows, all server-side processes will be run as the currently logged-on user, or whatever user name is set when running this software as a service.

In a UNIX/Linux environment, the application server will use **DEF UID** to launch new sessions, if **DEF UID** is supported by the operating system. Otherwise, the `"su"` command will be used.

There is no provision in the `"su"` command to pass the user's password as an argument; therefore, the only way you can run the application server, and have it launch processes as other users, is if the application server daemons are running as `"root"`; i.e., it does not require a password to run processes as other users.

User Name: This provides the user name that the server-side ProvideX sessions will be run as. (This has no effect when running Windows or when not using **CmdAsUser** in a Windows environment.)

Where the client is not forced to log on, any session request that is not preceded by a logon is classified as an anonymous user. The default user name is used to run any server-side ProvideX session for any connecting anonymous user.

User Password: ***Windows*** option for setting the password for the user ID given in the User Name. It is only used by the **CmdAsUser** utility and is not required under UNIX/Linux.

User Domain: ***Windows*** option for specifying the domain name for the user ID given in the User Name. It is only used by the **CmdAsUser** utility and is not required under UNIX/Linux. This is only used when Run All as Default User is set, or when sessions connect anonymously.

State Fields

This section allows an administrator to control the current *state* of the application server being configured.:

Disable Server: Checkbox to determine whether or not this server can be started (if currently not running). If disabled, the server will terminate before it starts accepting connections.

Disable New Connections: Checkbox for disabling new sessions (for maintenance purposes). If set, then a Reason should be specified to present to users attempting to connect.

Only new sessions that are started from the client's PC will be disabled — it does not affect currently-connected users. Current users will still be able to spawn additional sessions, but new connections via *CLIENT will be refused.

This setting does not affect the configuration system's ability to talk to the server daemons.

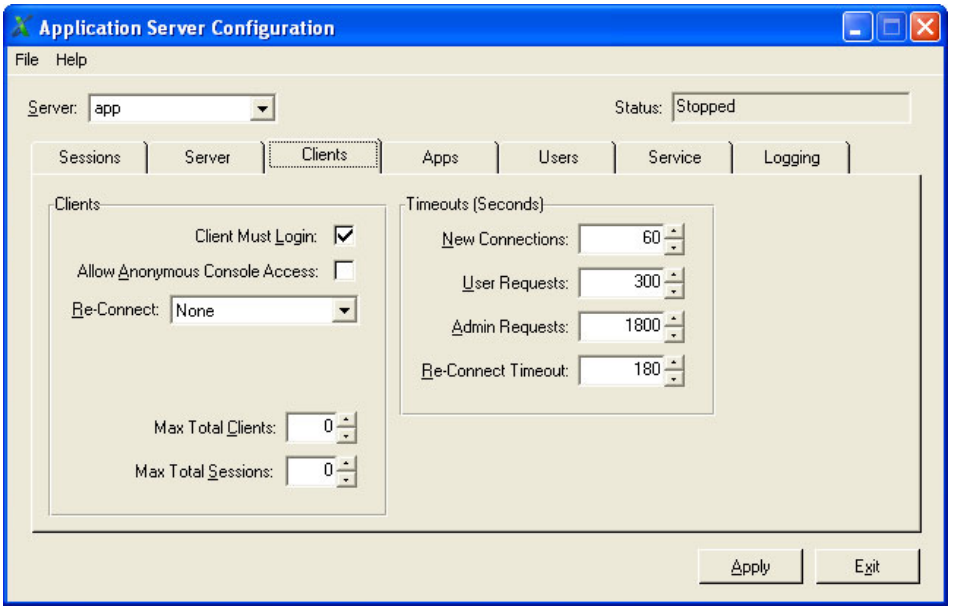
Disable All Connections: Checkbox to prevent all new sessions from being started. This includes new Client sessions (from *CLIENT or JavX) as well as any newly-spawned sessions from currently connected users. This will not affect currently-connected users, but it will prevent current users or potential users from starting new sessions of any kind.

This setting does not affect the configuration system's ability to talk to the server daemons.

Reason: Descriptive reason for disabling the server. If either Disable New Connections or Disable All Connections checkboxes are set, then the user will receive an error message informing them that the server is disabled for the reason provided.

Clients Tab – Defining Client Attributes

The Clients panel is used to configure some general properties of the server as they relate to the clients that will be connecting.



Clients Fields

This section under the Clients tab allows an administrator to control the current *state* of the application server being configured:

Client Must Login: Checkbox to prevent anonymous sessions. ***If selected,*** the server must receive a valid login request before it receives a session request. In this case, the user on the client-side must log in with a valid user name and password as configured in the Users panel. Only valid clients are allowed to request that sessions be run.

Currently-connected clients whose software does a spawn within the server-side process, do not need to log in again. Sessions "spawned" from within a current session pick up the current user's characteristics.

If not selected, then anonymous sessions are accepted pending other validation. The *CLIENT program tries to establish an anonymous session when it first connects.

Using the -LOGIN option on the *CLIENT program, it is possible to have both anonymous and logged-on clients. This argument tells the *CLIENT software to skip the anonymous session attempt, process the Login panel, and log in the user. Using this method, it is possible to have both types of users (where validated users have more privileges than anonymous ones).

Note: Anonymous users can only request and run applications that are configured within the Apps panel and cannot run any other ProvideX program on the server.

Allow Anonymous Console Access: Checkbox to allow users that have not given a valid logon to access the ProvideX console mode. From a security standpoint, anonymous users should never be given this access; however, in a trusted environment you may wish to set this option to simplify connections to the server for *development purposes*.

ReConnect: Options for controlling the ability of client sessions to reconnect to the server if their network connection fails or is interrupted. The settings are as follows:

None	Never allow a client to reconnect if their network connection fails.
Prompted	Inform the client that their connection has failed and ask if they wish to reconnect. This also informs the user if the reconnection is successful or not.
Automatic	Reconnect automatically, then inform the user that their session was interrupted and has been reconnected.
Hidden	Reconnect automatically without informing the user that their session had been interrupted/reconnected.

Max Total Clients: Maximum number of ***unique clients*** who may be connected to the server at one time. If set to 0, then there is no limit. This does not affect the number of active connections or the number of spawned sessions any one client may have.

The number of unique clients is dependent on the Client Must Login option. If the Client Must Login option is selected, then the number of unique clients is the number of different user names that may log in. Otherwise, the number of unique clients is determined by the number of different client IP Addresses that the server sees.

Max Total Sessions: Maximum number of sessions this server will allow at any one time. If set to 0, then there is no limit. The total is determined as a count of both client- and spawn-connected sessions.

Timeouts

This section under the Clients tab represents the timeout values (in seconds) for completing certain session-related operations:

New Connections:	Time limit (in seconds) that a connecting client has to make its first valid request. Any connection that does not make a request in this time will be disconnected. This keeps other TCP/IP-based software (port scanners or hackers) from tying up server resources.
User Requests:	Time limit (in seconds) that a connected client has to make any additional requests after their first valid request. Setting this value keeps the server from holding open or using up resources when a client may have disconnected without notifying the server while in the process of making requests.
Admin Requests:	Time limit (in seconds) for internal administration requests between the Configuration system and running server daemons.
ReConnect Timeout:	Time limit (in seconds) that a connecting client has to perform a reconnect from the moment the Application Server daemon realizes that the connection has been severed. Server-side processes will be kept running for this length of time before terminating. Once terminated, the client will not be able to reconnect to that session.

Note: When the reconnect limit is set to a value other than 0, the Application Server daemon will maintain a 64K buffer of data for each ProvideX process serviced, and the Client will maintain a 64K buffer of data it last sent to the server.

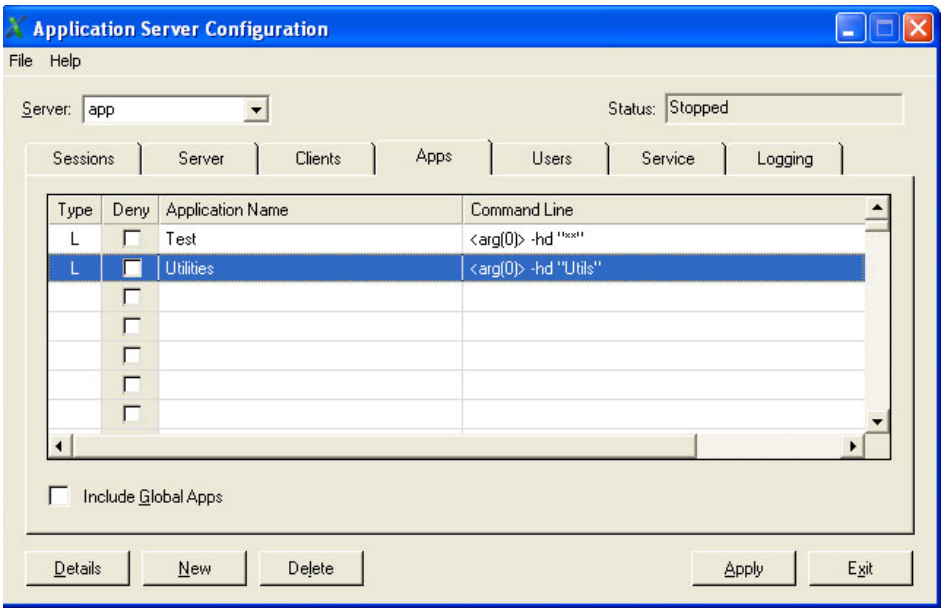
Clients will only be able to reconnect to their sessions within the same instance of the client. If the client exits and its process terminates, then it will not have the information required to reconnect – therefore, a reconnect would not be possible.

The client may not notice that its connection to the server has been severed. The client will only notice this happening if the user is currently performing some task within the client. That is, the client may sit idle for an extended period of time and not notice that the connection had been severed when a user begins to type or click the mouse - the server daemons reconnect timeout may have already expired and user will not be able reconnect.

Setting a long duration for a reconnect timeout may leave processes running on the server for that entire time, consuming server resources until their timeout expires.

Apps Tab – Application Listing

The Apps ***listing*** panel defines the applications that may be run by the currently-selected server. Application entries may be created, changed, or deleted using this panel:



Include Global Apps	Checkbox to determine whether or not the current server should use the globally configured applications (from the <Global Config>) in addition to any locally-configured applications.

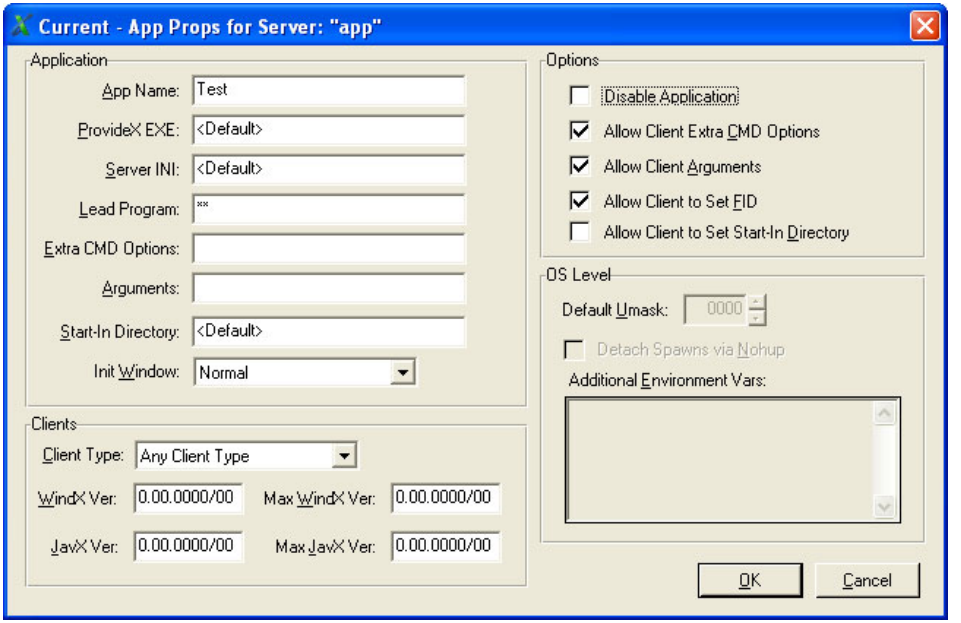
The ***listing*** for applications recognized by the current server is defined as follows:

Type	Indicates whether or not an application is **L** (configured locally within this server) or **G** (configured within the <Global Config>).
Deny	Checkbox (on/off) method for denying access to listed application.
Application Name	Case-insensitive application name. This is the name used in *Client or JavX software to identify and run the application.
Command Line	Representation of ProvideX command line syntax for the defined application. This syntax is not in fact used by the application server, as it requires other properties and programs, but is simply a way to describe properties used by the application when it is launched.

To *view or modify properties* of a listed application, highlight the application entry and select the Details button. To create a new application, select the New button. To define an application that is available to all configured servers, switch the Server field to <Global Config>. Application details are explained below.

Apps Tab – Details

The applications ***details*** panel is invoked by selecting the Details or New button from the Apps panel. It allows an administrator to configure application properties.



Application Fields

These fields in the Apps ***details*** panel provide the name for, and properties of, a given application:

App Name: Descriptive name to identify a set of application properties. These are the properties it takes to run an application.

Each application is created, referenced, and stored by using a descriptive name. This name is also used in *CLIENT or JavX command line syntax to request the specific application. The name entered may contain any legal characters up to 40 characters in length. You cannot use a character less than hex 20 nor any of the following: `"<> ,/\*?&|;:{}()$!` ` Application names are case-insensitive.

ProvideX EXE: Location on the server of the ProvideX executable to run this application. A null or <Default> setting indicates using the same executable that the application server daemon itself is using.

Server INI: ***Windows*** option for naming the INI file to use for the server-side ProvideX process. A null or <Default> setting indicates using the same INI that the application server daemon itself is using.

Lead Program:	Actual name of the ProvideX program to be run, known in ProvideX as the *lead program*. If null or CONSOLE, then this is classified as a console session and is subject to console restrictions.
Extra CMD Options:	Options that are to go between the lead program name and any statement on the command line. It is intended for system parameter settings; e.g., `-XT=1` or `-XT=0 -NE=1`. Extra command line options are supplied to the server-side process after the lead program name, but before any **-ARG** values; e.g., *ServSideCMD* "*LeadProgram*" *XtraCMDoptions$* **-ARG ...** The client can send additional command line arguments. Any such additional arguments will be added to the end of this argument string.
Arguments:	Command line arguments for the server-side process. These arguments will be available to the launching process via the **ARG( )** function and **NAR** system variable. These are supplied to the server side process as **–ARG** values; e.g., *ServSideCMD* "*LeadProgram*" **-ARG** *arguments$*
Start-In Directory:	Server directory where the application server is to launch the process for running the specified lead program. This will become the home working directory (HWD) of the process.

Options

These checkboxes in the Apps ***details*** panel provide on/off functionality for Application properties.

Disable Application	Checkbox to deny access for all users; otherwise, the application will be available to all users who are configured in this server.
Allow Client Extra CMD Options	Checkbox to enable extra command line options. If this is not selected, extra options sent by a client will be ignored. Extra options received by the server are in addition to the application's own; e.g, *ServSideCMD* "*LeadProgram*" *ServXtraCMDoptions$* [**-ARG ...**]
Allow Client Arguments	Checkbox to enable clients to send additional command line Arguments. Otherwise, such arguments sent by a client will be ignored. Extra arguments sent by a client are used in addition to the application's predefined arguments; e.g., *ServSideCMD* "*LeadProgram*" **-ARG** *App_Args$ AddClient_Args$*
Allow Client to Set FID	Checkbox to set **FID(0)** for the server side process to match the value used/requested by the client system. Otherwise, the server-side process lets ProvideX select whatever **FID(0)** value it chooses.
Allow Client to Set Start-In Directory	Checkbox to enable the client to request a different Start-In directory for the server-side process. Otherwise, any such request by a client is ignored.

Clients Fields

These fields in the Apps ***details*** panel can be used to control the type (WindX, JavX, or both) as well as the minimum/maximum version numbers of the client software accessing this application.

Client Type:	Client software permitted to access this application. Valid settings are: Any Client Type, WindX Clients Only, or JavX Clients Only.
WindX Ver:	*Minimum* version of WindX that may run this application. See the discussion on version codes (below) for details.
Max WindX Ver:	*Maximum* version of WindX that may run this application. See the discussion on version codes (below) for details.
JavX Ver:	*Minimum* version of JavX that may run this application. See the discussion on version codes (below) for details.
Max JavX Ver:	*Maximum* version of WindX that may run this application. See the discussion on version codes (below) for details.

Version codes are 9 characters in length, split into 2 parts. The first part is the 7-digit software revision; i.e., **TCB(29)**. The second part is the 2-digit thin-client version number; i.e., **TCB(88)**. When the application server evaluates the minimum / maximum versions numbers, it evaluates the 2 parts separately. If the 7-digit software revision number is 0, then no check will be made. If the 2-digit internal client level is 0, then no check will be made.

You can set just the software revision, just the internal client level, or both; e.g.,

0.00.0000/00	No level checking is performed.
5.01.0000/00	Only a software level of "5.01.0000" checked.
0.00.0000/08	Only an internal client level of "08" is checked.
5.01.0123/09	Both the software revision and the internal client level are checked.

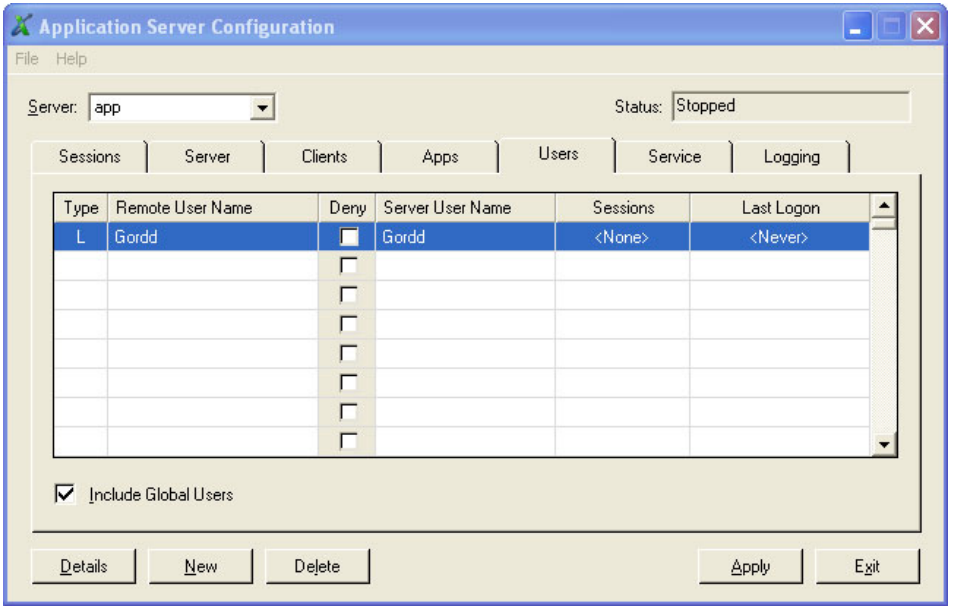
OS Level

These fields in the Apps ***details*** panel set properties needed for running applications under UNIX/Linux.

Default Umask:	Umask setting for running this application. Note, not all operating systems have this override.
Detach Spawns via Nohup	Checkbox to determine whether or not any spawns from within this application session will use nohup. The use of nohup determines if a process is to remain attached (or not) to the process that spawned it; and, if it is attached, then the child process will terminate when the session the parent is running has terminated.
Additional Environment Vars:	Operating system environment variables for the process that is to run this application.

Users Tab – Valid User Listing

The User panel lists all of the valid or configured users that may log into this server.:



Include Global Users	Checkbox to determine whether or not the current server should use the globally-configured user listing (from the <Global Config>) in addition to any locally-configured users.

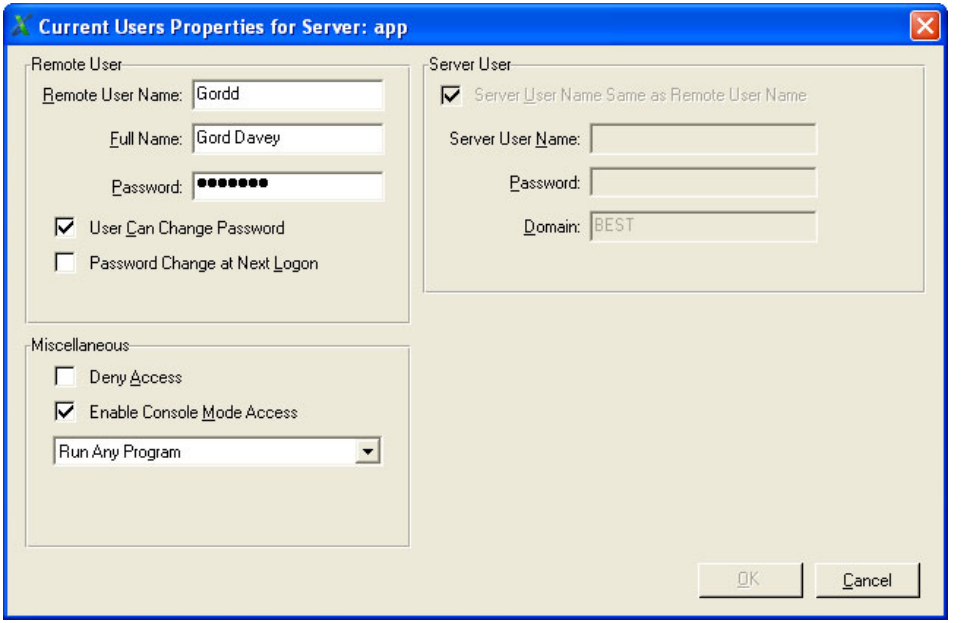
The ***listing*** for users who may log on to the current server is defined as follows:

Type	Indicates whether or not a user name is **L** (configured locally within this server) or **G** (configured within the <Global Config>).
Remote User Name:	Case-insensitive name that the user logs in as. This is the login ID users must supply to their *Client or JavX software.
Deny:	Checkbox (on/off) method for denying access to the listed user.
Server User Name	User ID associated with this user when running processes on the server. The User Detail panel allows you to associate the user ID of the remote users with a user ID the server can use.
Sessions	Indicates the number of sessions a listed user is currently running. The number of c (client) and s (spawned) sessions is given.
Last Login	Date the user last logged onto any of the configured servers.

To *view or modify properties* of a listed user name, highlight a user entry and select the Details button. To create a new remote user name, select the New button. To define a users that has access to all configured servers, switch the Server field to <Global Config>. User details are explained below.

Users Tab – Current Users Properties

This panel is invoked by selecting the Details or New button from the Users panel. It allows an administrator to configure a user's abilities to access the server.



Remote User Fields

These fields in the User ***details*** panel define access for a remote user name:

Remote User Name:	Case-insensitive login name for the remote user. This is the name the user would enter in any Login dialogue on the remote workstation running *CLIENT or JavX. (Maximum 60 characters). Internally this name is the one that appears within a login request to the server.
Full Name:	Descriptive name for the remote user provided for reference.
Password:	Password the user must enter while logging into the server.
User Can Change Password	Checkbox to enable users to change their password from the *CLIENT or JavX programs.

Password Change at Next Login	Checkbox to force the user to change a password the next time they log into the application server. Internally, when set, the server refuses any command from a user other than a "PASSWORD" change request after it receives a "LOGIN" request from a client workstation. Once a successful password change request has been completed, this setting is automatically turned off by the server.

Miscellaneous

These options in the user ***details*** panel control user access.

Deny Access	Checkbox to deny access. When selected, the user will receive an `Invalid Login` message when they attempt to log in.
Enable Console Mode Access	Checkbox to allow the user to access a ProvideX console prompt, either by sending a null, or by using `CONSOLE` as an application name. Also, this setting controls access to ProvideX console mode for all spawned sessions of this user.
Run Any Program / Configured App	This field determines what the user may run: any program, or configured applications. Run Configured App means that the listed user will have access only to those application names listed on the Apps dialogue. Run Any Program means the user will be able to access any application defined on the Apps dialog, and if the application is not found, then the application name given will be used as the lead program name of an application to run. For example, if a user is set to Run Any Program, and if the requested application name `Test` is not configured in the Apps dialogue, then the server will run `Test` if it exists.

Server User

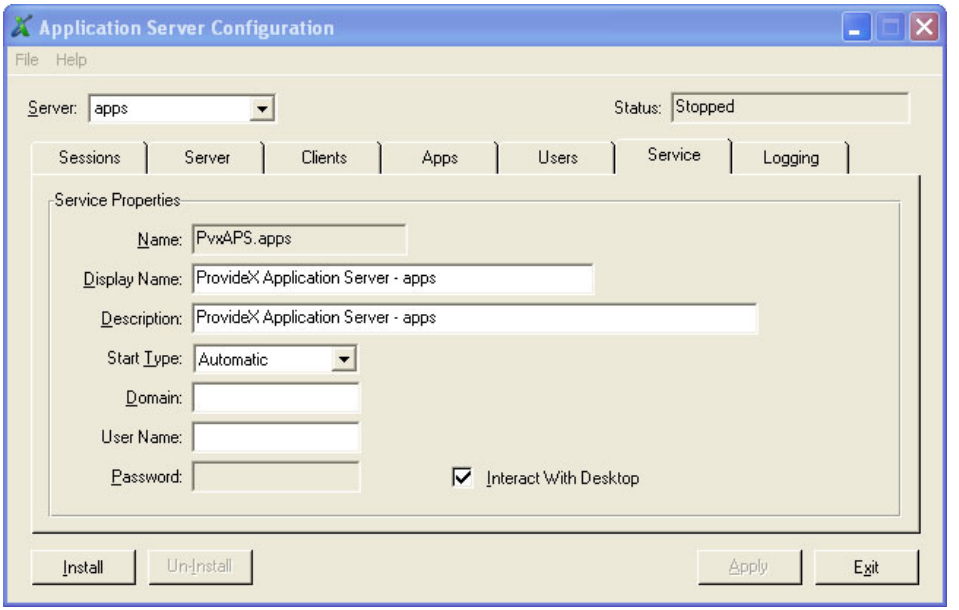
These options in the user ***details*** panel allow you to define the name, password and Windows domain that the server uses when launching processes on the server for a remote user. These settings are only in effect when running either under a UNIX/Linux environment, or when using **CmdAsUser** in a Windows environment.

Note: The setting in the **Server** tab called Run All as Default User has an affect on these settings — when this option is selected, these settings are ignored.

Server User Name Same as Remote User Name	Checkbox to enable the Remote User Name and Password along, with the default user's (or current Windows Domain) to be used to start the server process. Otherwise, the following values must be set:
Server User Name:	User ID on the server for running these processes.
Password:	Windows password required for the user account given to run processes on this server as the Server User Name.
Domain:	Windows domain name required for the user account given to run these processes on this server. The domain defaults to the domain name of the currently logged in user if available.

Service Tab – Windows Services

The Service Properties panel sets up the application server daemon to be run as a Windows service.



Name:	A unique service name, generated automatically.
Display Name: Description:	These are what the users will see displayed in the MS Windows Services listing.
Start Type:	Options to set up the Windows service for Automatic or Manual start or to have it Disabled.

Domain: User Name: Password:	If the service is to run as a user other than “local system”, then these properties may be set.
Interact With Desktop	Checkbox to enable access to the desktop. This is only valid when running as the "local system" account (not as another user):
Install: Un-Install:	Adds or removes the service from Windows.

To start or stop the service, use the Start/Stop buttons on the **Server Tab – Defining Server Attributes**, *p.86*. If the service is not installed, then selecting Start will start it on the Windows desktop.

Note: Changing any NT Service properties when the App Server daemon is running will require the daemon to be stopped, un-installed, re-installed and re-started.

Running each server daemon as a separate service can cause difficulties with ProvideX user count licensing. If you have more than one daemon running, and they are set to interact with the desktop, then they can share the total user license count. However, if they do not interact with the desktop, then they will not be able to share the user count.

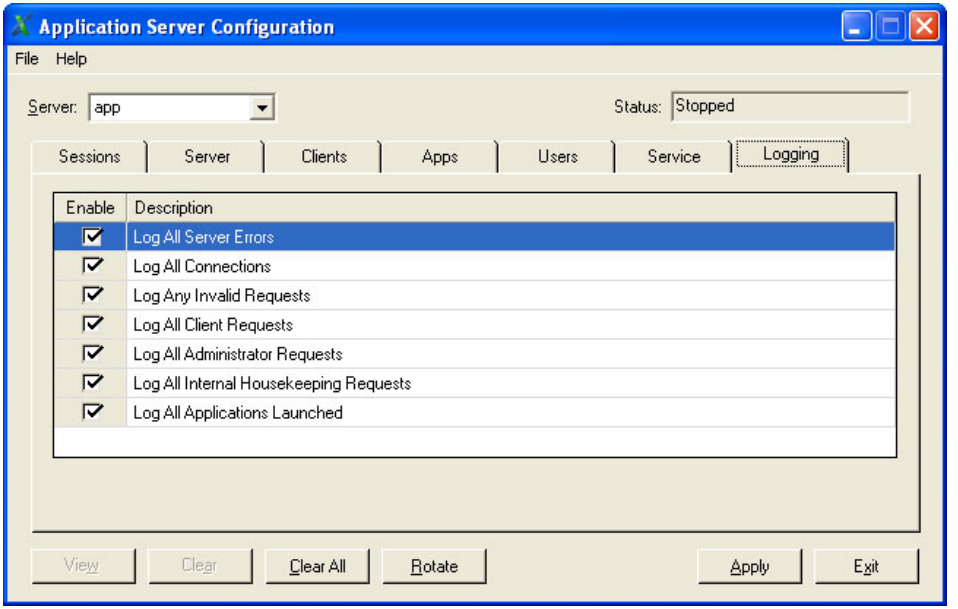
Example:

There are two server daemons (A and B) and 5 workstations (1,2,3,4 and 5). If all 5 connect to Daemon A, you will use 6 user slots - one for the daemon and one for each of the WindX workstations 1 through 5.

If all 5 of those workstations also connect to server B, and each runs one session, then you will use another 6 user slots - one for the B server daemon and one for each of the 5 workstations. This means you will have used 12 user slots in total. However, if you set both daemons A and B to interact with the desktop, and each of the 5 workstations connect one session to both server A and server B, then you will only use a total of 6 user slots, rather than a total of 12.

Logging Tab – List of Log Files

The Logging panel allows administrators to maintain aspects of the Application Server's logging functions. All log files are saved as plain ASCII text files in a subdirectory of the *appserv directory, which has the same name as the server.



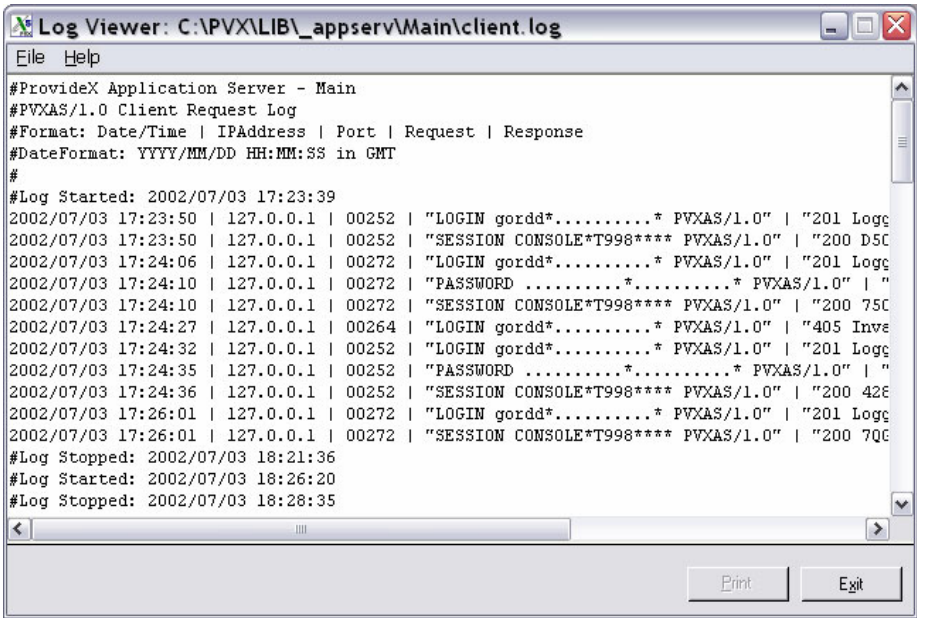
Use the check box in the Enable column to select logging of the different listed actions.

Log All Server Errors	Log of internal errors involving the application server daemon. Errors encountered in this log should be reported immediately.
Log All Connections	Simple log of all machines that connect or disconnect from the server, including clients and server side connections.
Log Any Invalid Requests	Log of all the requests that are invalid, unknown, or not supported, and of all the responses from the server. An example of an invalid request would be when someone opens the application server daemon's socket and attempts an HTTP or FTP request.
Log All Client Requests	Log of acceptable requests made by client workstations, and the response they received from the server.
Log All Administrator Requests	Log of administrative commands received by the server daemon and its response to that request. Currently, only the Application Server's Configuration tool makes administrative requests.
Log All Internal Housekeeping Requests	Log of internal requests and their responses. These are requests that the servers made when their processes were invoked, as well as server and client requests when their processes were spawned.

Log All Applications Launched	Log of all the sessions that clients requested and the command lines the server used to launch those sessions.
View	Button for *viewing* currently-selected log. See Log Viewer, below.
Clear / Clear All	Buttons for *clearing* selected/all logs.
Rotate	Button to *rename* all logs with a leading date; e.g., `20050622.140253.error.log YYYYMMDD.HHMMSS.*`

Logging Tab – Log Viewer

This is invoked from the Logging panel by clicking the View button for a selected log.



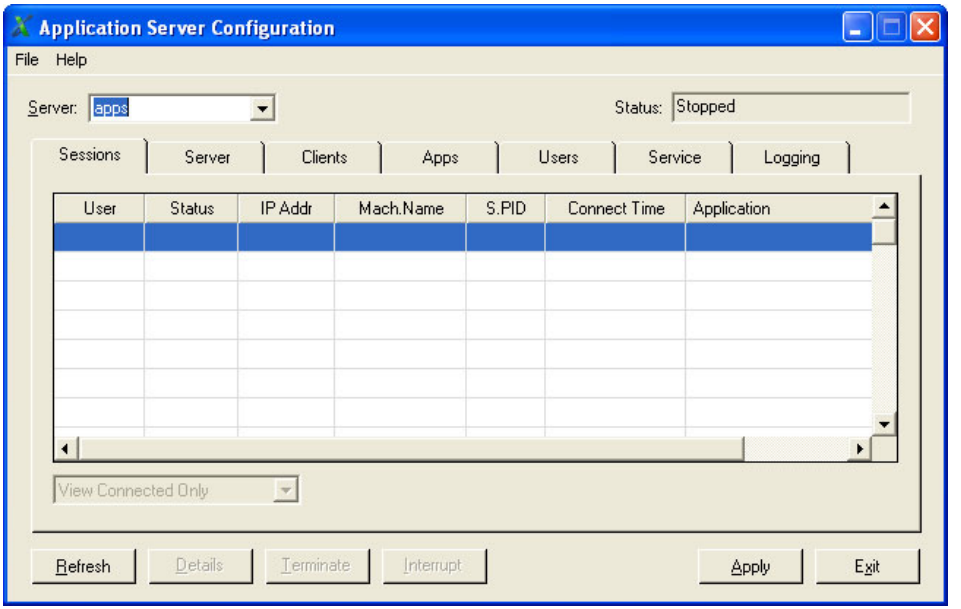
All log files are in ASCII text and follow a standard format where applicable. The pipe ' | ' symbol is used as the separator between items of information, as shown in the #format statement. The * *asterisk* is used in place of a ProvideX $8A$ field separator. All passwords in valid requests are hidden using 10 periods (…….).

Logs typically show the following information:

- Date / Time the entry was made.
- IP address and internal handle (port) number of the connection making the request.
- Request that was made.
- Response from the server.

Sessions Tab – Current Sessions Listing

The Sessions panel lists (and allows termination of) currently- running sessions.



Options for viewing sessions include:

View Connected Only	Currently running sessions.
View Terminated Only	Information about sessions that are no longer running.
View All	Both active and terminated sessions in a single listing.

The list is purged each time the server is started. Contents of the sessions ***listing*** is defined as follows:

User	Remote user name that the user signed on to the Application Server with to start their session.
Client	Type of client software the client workstation is using to access the server, either ***WindX*** or ***JavX***.
Type	Specific type of session the user is running, either ***Client*** or ***Spawn***.
Status	State of the session, either ***Connected*** or ***Terminated***.
Application	Name of the application the user requested to be run, or the lead program if the application was not configured.
Connect Time	Either the amount of time (in days, hours, minutes, seconds) an active session has been running, or the amount of time a terminated session had been running prior to termination.

Refresh	Button to update the sessions list. The list will also be refreshed whenever you switch back from another tab.
Details	Button to display an new window for details on the currently-selected session. Session details are explained below.
Terminate	Button to drop a session. This disconnects the user and terminates the session on the server, no matter what that session is doing.

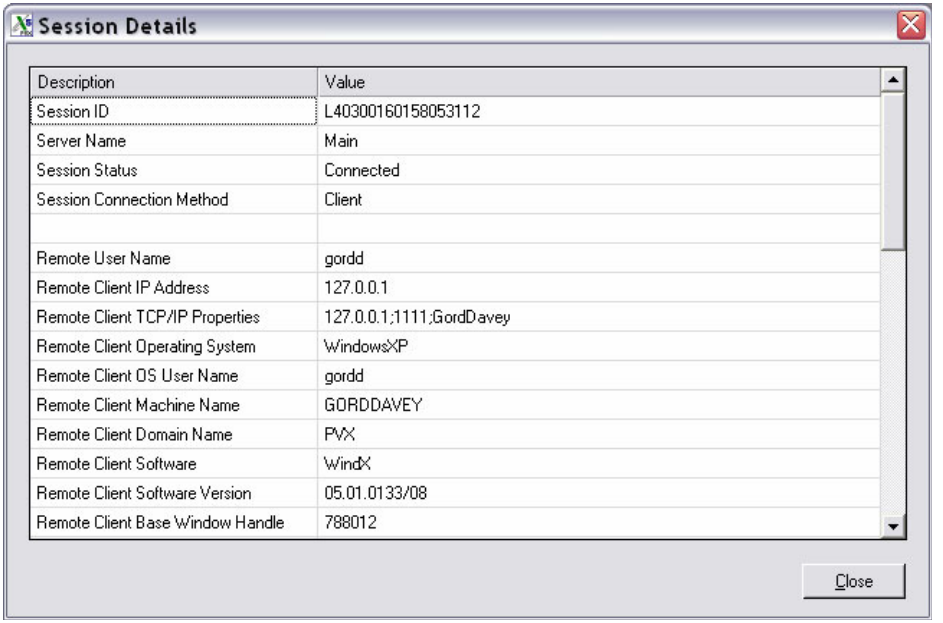
Note: Terminating without knowing the state of a session is dangerous since you could interrupt critical file processing. Exercise extreme caution when using this option.

Interrupt	Button to interrupt a session and drop the user into console mode on the client side (for troubleshooting purposes).

Note: The columns shown in the session listing are customizable. Right-click any column header to select which columns are listed. Drag on the headers to change the order of the columns.

Sessions Tab – Current Sessions Listing

This is invoked from the Sessions panel by clicking the Details button for a selected session. It provides further information about the session



Scroll down the list for complete details of the session, organized as follows:

- Session ID, its status, how and when the session was created or terminated, and the running time of the session.
- Client's user ID, IP address, operating system, software, handle numbers, process ID's, and information from the user's workstation itself.
- Which application the user chose to run and the properties used to launch it.
- User name that was used by the process on the server as well as the server's process ID, handles, and TCP/IP socket parameters.

Running the Server

The operation side of the Application Server is handled by the ***server daemon***, a platform-independent ProvideX program called `*appserv/server`. This program is designed to run as a ***background task*** and it is responsible for:

- Listening for client/server connections
- Responding to requests
- Launching new sessions
- Moving traffic between the client- and server-side processes.

The Start-In directory for the server daemon is an important consideration. When the daemon launches new processes (applications) it will start them in the same directory that the server daemon was started in. This occurs unless either the application requested had a designated Start-In directory, or the client requested a specific Start-In directory.

Associated files should be given a path that is relative to the Start-In directory of the server daemon.

Command Line Syntax

The server daemon command line syntax is described as follows.

In Windows: *PVXpath$* [*state*] [*ini*] ***appserv\server –ARG** *ServerName*

In UNIX/Linux: *PVXpath$* ***appserv/server –ARG** *ServerName*

Where:

PVXpath	Path to ProvideX
state	Option setting for governing the initial WindX window. Either null for normal window, **-mn** to start minimized, **–hd** to start hidden.
ini	Optional user-defined INI file (or `..\Lib\_appserv\apssrv.ini`). Do not use a * in the INI path, as ProvideX does not resolve them as it does program names. ***MS Windows only.***
ServerName	Name of a server defined in the configuration. Case-insensitive. May be quoted.

Starting the Server Daemon

The following are examples of a start command for an application server daemon.

Under Windows

If the Start-In directory is `C:\Pvx`; e.g.,

```
C:\Pvx\Pvxwin32.exe -mn "C:\Pvx\Lib\_appserv\appsrv.ini"
     *appserv\server -ARG "MainServer"
```

Under UNIX/Linux

Method 1. From a shell script; e.g.,

```
#!/bin/sh
TERM=ansi; export TERM
cd /usr/pvx
/usr/pvx/pvx \*appserv/server -ARG MainServer </dev/null >/dev/null 2>&1
```

Method 2. Directly from `/etc/inittab`. This uses "/ "as its starting directory; e.g.,

```
pvd1:2345:respawn:/usr/pvx/pvx \*appserv/server -ARG MainServer
     </dev/null >/dev/null 2>&1
```

Method 3. From the `/etc/rc` set of directories. (Linux method). Create a file in `/etc/init.d`. The simplest method is to copy an existing script from that directory and modify it to start the above command line. The script should contain the **START( )** and **STOP( )** functions required. Use symbolic links in the `/etc/rc?.d` directories to link to your script.

Stopping the Server Daemon

Under Windows

Right click on the ProvideX Application Server icon for the server daemon in the Windows Taskbar, then select close or interrupt.

Under UNIX/Linux

Send a `SIGBREAK` to the server daemon process; e.g., `Kill -2 pid` (pid is the process ID of the server daemon).

Client Configuration

A **WindX** or **JavX** thin client implementation is required in order to establish a user interface connection from a remote system to the ProvideX Application Server. The syntax elements required to launch thin-clients using a connection to the ProvideX Application Server are outlined in the sections below.

WindX

The Windows thin-client employs the Application Server ***CLIENT** program to connect to the server. ***CLIENT** is also used by the spawning process to initiate new sessions on the client workstation. Complete details on launching and using the **WindX** thin-client are provided in *Chapter 3.*

The command line syntax is described as follows:

PVXpath$ **[***state***] [***ini***] *client –ARG** *ServName* [*Socket*] ["*App*"] [*parms*]

Where

PVXpath	Path to ProvideX (e.g., `c:\pvx\pvxwin32.exe`)
state	Option setting for governing the initial WindX window. Either null for normal window, **-mn** to start minimized, **–hd** to start hidden.
ini	Optional user-defined INI file. The initial session will use this INI for its client-side ProvideX characteristics. Spawned sessions will re-use the same INI, unless specifically overridden during the spawn.
ServName	IP Address or DNS resolvable name of the server to connect to.
Socket	TCP/IP port (socket) that the server daemon is listening to. The default is 10000.
"*App*"	Optional lead program or configured application name that the server is to run. Quotes are required if it contains spaces. If Null or not given, then the request is for a ProvideX Console Mode Session.
parms	Optional dashed parameters: **-FID=***xxx* **-DIR=***xxx* **-CMD=***xxx* **-ARG=***xxx* **-SSL -LOGIN -KA -LANG=***xx* **-USR1=***xxx* **-USR2=***xxx* **-USR3=***xxx* **-USR4=***xxx*. These parameters are explained below.

*CLIENT *Dashed Parameters*

The following parameters may be included on the command line in any order as long as they are entered after the lead program name, "*App*" (or *Socket*, if no application is given):

-FID=*xxx*

FID(0) value requested. This is an override value, whereby you can request a specific **FID(0)** for this session. The Application configuration may be set to not allow such an override. The maximum **FID(0)** value is 12 characters (quotes are optional).

Example: `-FID=T999 or -FID="JoesPC"`

-DIR=*xxx* Server-side Start-In directory. This is an override value, whereby you can request a specific directory to start the server session in. The Application Configuration may be set to not allow such an override (quoting of value is optional even if the path contains spaces).

-CMD=*xxx* This is an "addition to" value, whereby you can request additional command line options for the server-side process.

Examples: `-CMD=-XT=1 or -CMD=-XT=0 -NE=1` *or* `-CMD="-XT=0 -NE=1"`

Extra command line options are supplied to the server-side process after the program name, but before any –**ARG** *xxx* values; i.e., *Server-Side_Process_Command Application XtraCmdLineOptions$* -ARG … …

The application configuration may be set to refuse such an addition (quoting is optional, but required if there are spaces).

-ARG=*xxx* This is an "addition to" value, whereby you can request additional command line arguments for the server-side process. These are in addition to any application configured command line arguments. Extra command line arguments are supplied to the server-side process after any configured –ARG values; i.e., *Server-Side_Process_Command Application$ -ARG arg1 arg2 ExtraArg1 ExtraArg2*

The application configuration may be set to refuse such an addition (quoting is optional, but required if there are spaces).

-SSL Indication that the ***CLIENT** program is to connect to the server using SSL. If the server is using SSL, the client must also use it.

-LOGIN Forces a login to the server. By default, ***CLIENT** will first attempt an anonymous connection to the server. Depending on the server's configuration, the server may accept anonymous requests or it may require a login. If the server is set for login, then the user is presented with the login screen.

By setting –**LOGIN**, you are telling the ***CLIENT** program to skip the attempt at an anonymous session and force the user to log in.

-LANG=*xx* Two-character Language Code for displaying message boxes and dialogue information; e.g, `-LANG=EN` for *english*.

The following rules are used to select the correct language suffix. An attempt is made to load the `apsmesg.??` message library in a specific order. If an attempt succeeds, then that is considered to be the correct language suffix for all message library and NOMADS Panel library files. Attempts will continue until successful or "en" is chosen as the default.

The order of the attempts is: #1 - %LANG$ Global variable set possibly via a `START_UP` program, #2 - Environment variable `PVXLANG`, #3 - Environment variable `LANG`, #4 - Default of `EN`.

-KA Indication that the client is to use TCP/IP *KeepAlives* on its end of the connection.

-USR1=*xxx* *to* **–USR4=*xxx*** You may specify user defined strings containing any information you wish. This information will be available to your server side process from the `%APS` object as properties. If the strings supplied contain spaces, then you should enclose the string in quotes; e.g., `-USR1="My Directory Info"`

JavX

The client-side software for accessing the application server is built into the Java-based thin-client. To use the application server with JavX, follow the standard documented methods for starting a JavX application from the command line, or from an applet tag on a Web Page. Complete details on launching and using the **JavX** thin-client are provided in *Chapter 4.*

JavX accepts a variable called **ARGS** within a web page. To use the application server, pass **APPLICATIONSERVER=TRUE** within the arguments, and then pass any other argument references necessary to start the session. The argument listing for possible use in the JavX **ARGS** parameter is described below:

`ApplicationServer=TRUE`

Required parameter. If not given, JavX will default to the ***NTHOST** method which will fail against an application server daemon.

`Server=`*`ServerName`*

Either the IP address or DNS-resolvable machine name of server.

`Port=`*`SocketNum`*

The TCP/IP socket that the server daemon will listen to.

`Program=`*`ProgName`*

The name of the application or program to run. For a console mode session, pass either null or `CONSOLE`.

`Login=TRUE`

Optional parameter. If given, then JavX forces the client to login. If not given, then an anonymous session is attempted first.

`ClientFID=`*Value*

Optional parameter. The value of the `FID(0)` the client wants the server to use (may be ignored by the server).

`ClientStartInDirectory=`*Value*

Optional parameter. The start-in directory where the server will run the application (may be ignored by the server).

`ClientCMDOptions=`*Value*

Optional parameter. Extra command line options that may be passed to the server (may be ignored by the server).\

`ClientArguments=`*Value*

Optional parameter. Additional command line arguments (`-ARG`) which are passed to the server (may be ignored by the server).

Note: Arguments to JavX within the **ARGS** parameter must be semi-colon separated.

Sample Configuration

The following is a sample JavX configuration for use as an application server client:

```
<OBJECT classid="clsid:8AD9C840-044E-11D1-B3E9-00805F499D93" width="200"
       height="50" align="baseline"
       codebase="http://java.sun.com/products/plugin/1.3/jinstall-13-
       win32.cab#Version=1,3,0,0">
<PARAM NAME="code" VALUE="NetworkClientApplet.class">
<PARAM NAME="type" VALUE="application/x-java applet;version=1.3">
<PARAM NAME="args" VALUE="ApplicationServer=True;
       server=www.mydomain.com;
       program=myprog.pvx;
       port=10000; ">
<COMMENT>
       <EMBED type="application/x-java-applet;version=1.3" width="200"
       height="50" align="baseline" code="NetworkClientApplet.class"
       args="applicationserver=true;server=www.somedomain.com;
       program=myprog.pvx; port=10000;"
       pluginspage="http://java.sun.com/products/plugin/1.3/plugininstall. html">
<NOEMBED>
No JDK 1.3 support for JavX
</EMBED>
</COMMENT>
</OBJECT>
```

The following example launches JavX for a connection to the application server:

```
java -jar JavX.jar "ApplicationServer=True; Server=192.192.192.192;
      Port=20000; Program=Console; Login=True"
```

Session Spawning

The ProvideX Application Server also includes a component, ***SERVER**, that can be run on the server to *programmatically* establish a new independent session of ProvideX within which another application may be run. This is commonly referred to as ***spawning*** a new session.

Note: This process requires WindX version 5.10 or higher, or JavX version 1.10 or higher.

The ***SERVER** program takes information from the current session of ProvideX regarding the application server, and information from the programmer about what to run. It launches a new *instance* of ProvideX on the server using the credentials of the user from the current session, while also forcing the client's workstation to start a new instance of the client software.

The ***SERVER** program generates a session ticket for the application server and informs both the new server-side process and the client-side process about the session. At this point, the new server- and client-side processes connect to the application server and begin a new independent session. This session will be running the application with the characteristics for which the programmer coded, within the **CALL** list of the ***SERVER** program.

The session of ProvideX which is issuing the **CALL** to the ***SERVER;SPAWN** command must itself be an application server-connected session, from either a ***CLIENT** or previous ***SERVER;SPAWN**.

The ***SERVER** program has two line label entry points: **Launch**, which is an internal entry point used by newly-invoked processes on the server, and **Spawn**, which is an entry point used by programmers to spawn new independent sessions of ProvideX and the applicable client on the workstation.

The command line syntax is described as follows:

CALL "*SERVER;SPAWN",*LeadPrg$*,*AppName$*,*StartInDir$*,*ServINI$*,*ClientINI$*, *ClientsState$*, *ExCMDOpts$*,*Args$*,*FIDVal$*,*NoHup$*

Where

LeadPrg$	Either null, indicating a console session, or a valid pathname to a ProvideX program.
AppName$	Either null, or a descriptive name to appear in the configurator's session listing.
StartInDir$	Either null, indicating the current directory, or a specific directory where the server-side process is to start.

ServINI$ Either null, or a valid pathname to an INI on the server to use for this new session. If it is null, then the INI used for the new ProvideX session on the server will be the same as that for the current session.

ClientINI$ *WindX clients only.* Either null in which case the WindX client will launch a new instance using the same INI it is currently using, or a valid pathname to an INI on the client PC for the new WindX process to use.

ClientsState$ Allows you to control the initial viewing state of the new process on the client. This is a string of one-character options:

H Hide initial dialogue
M Minimize initial dialogue
N Show new process dialogue in the "Normal" state
X Maximize new process dialogue (currently not implemented)
E ***JavX only.*** Embed new process into the browser's window
F ***JavX only.*** Float new process dialogue above browser window

ExCMDOpts$ Extra command line options for server-side process, whereby you can request additional command line options for the server-side process. (May be null if no arguments are required.)

Examples: `"-XT=1" or "XT=0 -NE=1"`

Extra command line options are supplied to the server-side process after the program name, but before any –ARG values; e.g.,

ServSideCommand LeadProgram$ ExCmdOpts$ -ARG …

Args$ Additional command line arguments for the server-side process, whereby you can request command line arguments for the server-side process. (May be null if no arguments are required.) Embedded quotes should be used as necessary.

Extra command line arguments are supplied to the server-side process as –ARG values; i.e.,

ServSideCommand LeadProgram$ -ARG Args$

FIDVal$ The **FID(0)** value requested. If this value is null, then the newly-spawned session will have the same **FID(0)** value as the current session. The maximum **FID(0)** value is 12 characters.

NoHup$ *UNIX/Linux Only.* Allows you to decide whether the newly spawned session is to be started using nohup or not. The use of nohup determines if a process is to remain attached (or not) to the process that spawned it; and, if it is attached, then the child process will terminate when the session the parent is running has terminated.

This is a one-character string option:

"" (*null*)	Use the settings in the application server to determine if the sessions are nohup'd or not.
"Y"	Force the session to be nohup'd.
"N"	Force the session to remain connected to the parent process (***not*** nohup'd).

Additional Notes About Spawning

- The newly-spawned session will be spawned as the current user.
- If the current session (the one doing the ***SERVER;SPAWN**) is SSL-connected, then the spawned session will also be SSL-connected.
- Access to ProvideX console mode for ***SERVER;SPAWN** sessions is governed by the Application Server's configuration. If the user was not allowed to access console mode in the Application Server configuration, then they will not be allowed to access it via a spawn.
- The full parameter list on the **CALL** to ***SERVER;SPAWN** is not required; e.g.,

```
CALL "*sever;spawn"  Spawns new console session
CALL "*server;spawn","***"  Spawns new session running ProvideX utilities
```

- All text and messages used by ***SERVER** are stored in the ***APPSERV/APSMESG.EN** message library. **EN** can be changed to another language suffix.
- **%APS** is an object itentifier that has information available to the session about its characteristics. To view the list of session properties, **PRINT %APS'***. It is recommended that you never set any of these object properties or use any of its functions. Doing so may make the current session unable to spawn new sessions. The properties are outlined in the section **Session Object Properties**, *p.115*.

Session Object Properties

Each new session, whether created from a request by ***CLIENT**, or from a ***SERVER;SPAWN** has an OOP object whose object identifier is stored in **%APS** that can be used to get information about a session's characteristics.

For example, the following test can be applied to determine if program code is running within an application server session:

```
if %APS and %APS'SessionID$>""
   then UsingAppServer=1
```

Not all properties will be available in all sessions or for all circumstances.

Server-Side Information

The following **%APS** properties provide information about the ProvideX process on the server-side:

Administrator$	Unused - future feature.
AllowConsoleMode$	**0** = No access to console mode, **1** = Console mode allowed.
AllowReconnect	Numeric value representing the type of reconnect allowed. **0** = None.
AppEnvVars$	Unused - future feature.
AppName$	Name of the application the user requested to be run.
AppUMask$	(*UNIX/Linux*) File creation umask that the session was created with (not honored by all OS's).
Arguments$	Additional arguments passed to this session by a request from the client.
ClientConnectionID$	Internal socket number (IND value) the server daemon has for the client side of the connection.
ClientConnTimeStamp	8-decimal place Julian time/date when the client side process connected to the server daemon.
ClientDomain$	(*Windows only*) OS domain name from the client-side workstation.
ClientLang$	The 2-character language code the client is using.
ClientNID$	ProvideX **NID** (machine ID) from the client-side process.
ClientOS$	Descriptive name of the client's operating system.
ClientPID$	OS process ID for the client-side process (from client).
ClientSoftware$	"WindX" or "JavX" depending on which client software is connecting.

ClientSoftwareVersion$	9 digit number representing the client's software version. **7** being the ProvideX **TCB(29)** and **2** being the WindX major revision code.
ClientsOfType$	Client access where **1** = Any kind of client can access, **2** = WindX clients only, **3** = JavX clients only.
ClientTCPProperties$	String of TCP/IP characteristics: IP; socket; machine for the channel the client-side process has opened to the server daemon.
ClientUID$	ProvideX UID from the client side process.
ClientUSR1$	The client's user-defined string information from the **–USR1** argument.
ClientUSR2$	The client's user-defined string information from the **–USR2** argument.
ClientUSR3$	The client's user-defined string information from the **–USR3** argument.
ClientUSR4$	The client's user-defined string information from the **–USR4** argument.
ClientWHnd$	(*Windows only*) OS Window handle for the client-side process (from the client machine).
ClientWHO$	ProvideX **WHO** from the client-side process.
ConnMethod$	**""** if the session was started by ***CLIENT**, or **"Spawn"** if started by ***SERVER;SPAWN**.
CreatedTimeStamp	8 decimal place Julian time/date when the session was first created.
DestroyedTimeStamp	8 decimal place Julian time/date when the session was terminated.
DetachSpawns$	0 = Spawns new sessions as attached processes, **1** = nohup any new spawns.
ExtraCMDOptions$	Additional command line parameters passed to this session by a request from the client.
FIDValue$	**FID(0)** value requested by the client.
KeepAlives$	0 = No KeepAlives, 1= Using KeepAlives.
Lang$	Language code: EN or FR, etc.
LeadProgram$	CONSOLE for a console mode session, or ProvideX program name used as the lead program for the server-side session (**LPG** contains internal use information only).
ListeningSocket$	TCP/IP socket number that the server daemon is listening to for *ServerName$*.
MaxJavXVer$	9 digit number representing the maximum JavX version that may connect (all 0's is any).

MaxWindXVer$	9 digit number representing the maximum WindX version which may connect (all 0's is any).
MinJavXVer$	9 digit number representing the minimum JavX version that may connect (all 0's is any).
MinWindXVer$	9 digit number representing the minimum WindX version which may connect (all 0's is any).
Protocol$	Always set to **PVXAS/1.0**.
ProvideXEXELocation$	Location of the ProvideX executable used to start the client process, and to start any subsequent spawns from the application server.
SecureSSL$	0 = Normal session, 1= SSL-encrypted session.
ServerConnectionID$	Internal socket number the server daemon has for the server side of the connection.
ServerConnTimeStamp	8-decimal place Julian time/date of when the server side process connected to the server daemon.
ServerDir$	Always set to ***APPSERV**.
ServerName$	Name of the application server the session is being run through.
ServerPID$	OS process ID for the server side process.
ServerProg$	Name of the server-side program that was launched to start the physical session, usually ***SERVER**.
ServerProgLabel$	Name of the entry point for the server-side program that was launched to start the physical session, usually **Launch**.
ServersINI$	INI file the server side process was told to use by the application server.
ServerTCPProperties$	A string of TCP/IP characteristics: IP; socket; machine for the channel the server side process has opened to the server daemon.
ServerUID$	ProvideX **UID** for the server side process.
ServerUserDomain$	(*Windows only*) OS domain name the server-side session was launched as.
ServerUserName$	OS user name the server side session was launched as.
ServerWHnd$	(*Windows only*) OS window handle for the server-side process.
ServerWHO$	ProvideX **WHO** for the server-side process.
SessionID$	Session identification token. Each session is identified by a unique token between 17 and 24 characters.
SessionStatus	Session status whe 0 = None connected, 1 = One side connected, 2 = Both sides connected, 255 = Terminated.

ShowSpawnsOnDesktop$	(*Windows only*) 1 = Server to show tasks on taskbar, 0 = Server to hide task bar buttons.
SocketOptions$	TCP/IP socket options the server side session is using to talk to the server daemon.
StartInDirectory$	Directory the current session was told to start in by the application.

Client-Side Information

The following **%APS** properties provide information about the ProvideX process on the client-side:

AllowReconnect	Numeric value indicating what type of reconnect if any is allowed for this session.
AppName$	Name of the application the user requested to be run.
ClientArguments$	Additional arguments for the server process that the client requested.
ClientCMDOptions$	Additional command line parameters for the server process that the client requested.
ClientDomain$	(*Windows Only*) Client workstation's OS domain name.
ClientFID$	**FID(0)** value requested by the client.
ClientINI$	INI file in use for the client side process.
ClientLabel$	Name of the entry point for the client side program which was launched to start the physical session, usually "" or **Spawn**.
ClientNID$	ProvideX **NID** (machine ID) from the client-side process.
ClientOS$	Descriptive name of the client's operating system.
ClientPID$	OS Process ID for the client side process.
ClientSoftwareVersion	9 digit number representing the client software's version. Version: 7 being the ProvideX **TCB(29)** and 2 being the WindX major revision code.
ClientStartinDirectory$	The start-in directory for the server process that the client requested (not the client's start-in directory).
ClientTCPProperties$	String of TCP/IP characteristics: IP; socket; machine for the channel the client-side process has opened to the server daemon.
ClientUID$	ProvideX **UID** from the client side process.
ClientUSR1$	Value that has passed to the client on the command line for **–USR1**.
ClientUSR2$	Value that has passed to the client on the command line for **–USR2**.

ClientUSR3$	Value that has passed to the client on the command line for **–USR3**.
ClientUSR4$	Value that has passed to the client on the command line for **–USR4**.
ClientWHnd$	(*Windows only*) OS Window handle for the client-side process.
ClientWHO$	ProvideX **WHO** from the client-side process.
KeepAlives$	0 = Do not use KeepAlives, 1 = Use KeepAlives
Lang$	Language code: EN or FR, etc.
LeadProgram$	Name of the lead program the user supplied from a spawn request.
Protocol$	Always set to **PVXAS/1.0**.
ProvideXEXELocation$	Location of the ProvideX executable used to start the client process, and to start any subsequent spawns from the application server.
SecureSSL$	0 = Normal session, 1= SSL-encrypted session.
ServerDir$	Always set to ***APPSERV**.
ServerName$	IP address or DNS resolvable machine name of the server that the client used to contact the server initially.
ServerSocket$	TCP/IP port or socket number the client used to contact the server.
SessionID$	Session identification token. Each session is identified by a unique token between 17 and 24 characters.
SocketOptions$	TCP/IP socket options the client side session is using to talk to the server daemon.

Customizing

The following sections describe some of the customizable features of the ProvideX Application Server., including the text language, login and change password dialogues, and ***CLIENT**.

Language Code

All displayed text for the configuration system, the server, and the clients is located in a single message library: ***APPSERV/APSMESG.*??*.** The default language suffix for displayed text is **EN** (English). You can change the language by using a different language code for the extension.

The Application Server language is dependant on how the message library is loaded (in a specific order). If an attempt to load the message library succeeds, then that is considered to be the correct language suffix for all message library and NOMADS panel library files. Attempts to load the file will continue until successful, or **EN** is chosen as the default. The order of the attempts is:

1) %LANG$ global variable set possibly via a START_UP program
2) Environment variable PVXLANG
3) Environment variable LANG
4) Default of **EN**.

Login and Change Password Dialogues

***CLIENT** attempts to use dealer-supplied panels before loading the default panels.

Login Screen

Attempted Panel From Library: ***APPSERV/DEALER.***??*

Arguments: *LoginID$, LoginPassword$, ChgPasswordFlag$*

Where *??* represents the language code. The arguments returned by the panel include:

LoginID$	String containing the user ID to send to the server daemon. If this is null, then ***CLIENT** will terminate without logging in.
LoginPasswrd$	String containing the password for *LoginID$* given.
ChgPasswordFlag$	String containing a **1**, which indicates the user wishes to change their password, or any other value, which is ignored.

Change Password Screen

Attempted Panel From Library: ***APPSERV/DEALER.***??*

Arguments: *Password1$, Password2$*

Where *??* represents the language code. The arguments returned by the panel include:

Password1$	String of the new password. If null, then the change password operation is aborted and the ***CLIENT** program is terminated.
Password2$	String that must match *Password1$*.

Custom ***CLIENT** Programs

Because the application server uses a protocol concept to connect the clients to the server, you may customize the ***CLIENT** program, or create your own to provide your software with a custom interface for your end-users.

The protocol for the requests that the server accepts is similar to HTTP protocol. You send a ***request*** and the server will respond with an answer. Requests must be issued in the form:

<METHOD><SPACE><REQUEST><SPACE><PROTOCOL><CRLF>
<METHOD> can be: LOGIN, SESSION, PASSWORD, QUIT
<REQUEST> is the data required by the method, in encoded form.
<PROTOCOL> is always "PVXAS/1.0"
<CRLF> is $0D0A$
<SPACE> is always " " or 20

The encoding is done using the following function:

```
def fnEncrypt$(local dat$)=hta(rdx(hta(dat$)))
```

While this encoding is not difficult to break, it ensures that the text of all commands is encoded and not sent as readable text. This makes it much more difficult for people to obtain user ID's and passwords when you are not running SSL encryption. Only the commands sent to the server are encoded. The responses from the server are not. Also, once the session has been started successfully and WindX or JavX has taken control, there is no further encoding in place.

Many customizations features exist. The **%APS** object and the ***CLIENT** program for the client side of the connection contain properties and methods which may be used to establish communications and send and receive responses. Some of the methods include the ability to transfer files and so on.

Protocol Methods

All protocols require a $8A$ between any arguments in their <REQUEST> section, including a trailing **SEP** for the last field in the <REQUEST>.

LOGIN	Request Fields: *LoginID$*, *LoginPassword$*
SESSION	Request Fields: *Application$*, *Fid$*, *StartInDirectory$*, *ExtraCmdOptions$*, *ExtraArguments$*
PASSWORD	Request Fields: *Password1$*, *Password2$*
QUIT	Request Fields: NONE

The following is an example of a LOGIN request, where the User ID is "Harry" and the password "YaItsMe".

```
OUTSTRING$="LOGIN"+"
      "+fnEncrypt$("Harry"+$8a$+"YaItsMe"+$8a$)+"
      "+"PVXAS/1.0"+$0D0A$
```

All responses from the server are in the form of a single line of text ending in a $0D0A$. The single line of text is broken down into 3 sections

<RESULT CODE><SPACE><TEXT><SPACE><PROTOCOL><CRLF>

The Result Codes are 3 digit numerical codes:

Codes 100-199 are not sent to the clients but are used internally by the server.
Codes 200-299 indicate acceptance.
Codes 300-399 indicate acceptance but further action is required.
Codes 400-499 indicate failure.
Codes 500-599 indicate a server level failure.

With codes 400 or higher, the server will automatically close the connection to the client right after it sends its response. If you wish to talk to the server you must reestablish the connection.

The <TEXT> given will indicate the nature of the response, or at times it will give back information that should be maintained for future reference. The methods LOGIN, PASSWORD and QUIT will only ever get one response from the server.

The method SESSION will get one, possibly two responses from the server.

If a successful SESSION request is received by the server, then the server will respond with a 200 result. The text of the result will contain a Session Token which the client should hold on to for future reference. When the client receives a 200 response it must immediately recheck its receive channel, and wait for a 205 response which indicates Ok To Go Ahead, meaning go ahead and run WindX.

If the SESSION request was unsuccessful, then the client will only receive 1 response indicating why it failed.

An example of a SESSION request would be:

```
Application$="MyApp"
```

```
Fid$="T99"
StartInDirectory$=""
ExtraCmdOptions$=""
ExtraArguments$=""

OUTSTRING$="SESSION"+" "+fnEncrypt$
(Application$+$8a$+Fid$+$8a$+StartInDirectory$+$8a$+ExtraCmdOptions$+$8a$
      +ExtraArguments$+$8a$ )+" "+"PVXAS/1.0"+$0D0A$
```

An example of a response you would receive would be:

```
"200"+" "+"ABC34FG91ETSFJL34923"+" "+"PVXAS/1.0"+$0D0A$
```

The response codes typically received for a session request are:

`200`	Successful, Session Token contained in text. Now wait for a 205.
`205`	Go Ahead, both server and client are ready to connect to each other.
`301`	You must Login first.
`410`	Console Mode is not allowed.
`412`	Access to Application is Denied.
`414` ***to*** `502`	Other errors indicating specific failure, the specifics of the error is given with the <Text> returned.

Sample Setup Procedure

This section describes a typical procedure for setting up and running the Application Server for use with a ProvideX client-server application. Configuration details are slightly different depending on whether the server is a Windows or UNIX/Linux system.

The procedure uses the following sample ProvideX application saved as `app1`:

```
0010 PRINT 'CS'
0020 BEGIN:
0030 PRINT @(0,0),"Wecome to the Applicaton Server",
0040 PRINT @(0,3),"You are currently running "+PGN,
0050 PRINT @(0,4),"Local Work Directory: "+LWD,
0060 PRINT @(0,5),"User ID: "+WHO
0070 PRINT @(0,7),"Press any key to escape to console mode:",
0080 OBTAIN (0,SIZ=1)X$
0090 ESCAPE
0100 GOTO BEGIN
```

MS Windows: The Windows configuration assumes that `app1` is saved on the server system in the following directory:

```
C:\app\
```

UNIX/Linux: The UNIX\Linux configuration assumes that `app1` is saved on the server system in the following directory:

```
/home/user1/app/
```

UNIX\Linux examples also assume a user login on a machine named `user1`, which has access to run ProvideX. This user also has a home directory of `/home/users1/` and a password of `password`. Change these references to ***your user name***, ***password***, and ***home directory***.

Using JavX for the UNIX/Linux Configuration

The *GUI* **Server Configuration** utility will not run directly under a UNIX/Linux version of ProvideX unless it is accessed using a graphical thin-client (JavX SE or WindX). The difficulty for the initial setup is that the thin-client will not be functional unless the hosting facility is already configured.

A ***simple solution*** to this particular dilemma is to run the configuration via JavX SE using ***NTHost/*NTSlave** *directly on the host itself.* In other words, JavX provides the GUI environment necessary to run the user interface for the Application Server configuration without the need for remote access. For complete documentation on JavX SE, see **JavX for PC Platforms**, *p.60*.

The UNIX/Linux procedure for ***Step 1.***(below) explains how to use JavX to launch the user interface of the Application Server configuration utility.

Step 1. Start the Configuration Utility

Launch the user interface to the **Server Configuration** utility.

MS Windows: From the Windows Start menu, select:
Start > Programs > Sage Software > ProvideX V*x.xx* > Application Server >App Server Configuration

or

In ProvideX console mode, enter:

```
->run "*appserv/config"
```

UNIX/Linux: Start *NTHost. At the system prompt, enter:

```
#\usr\pvx \*nthost -arg 10000 root 000 10005
```

This will launch an *NTHost session on the server as user ID root starting from port 10000 to 10005.

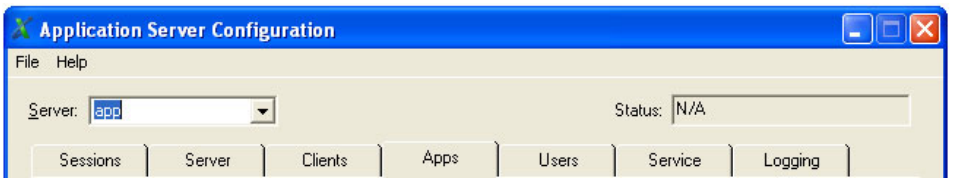
If JavX is not already installed, download and install the JavX Developers Kit (or just the JAR files) from the ProvideX website. Run JavX to connect to *NTHost and run the Application Server configuration:

```
java -jar -JavXSE.jar
     "server=localhost;port=10000;program=*appserv/config"
```

If you do not have the PATH variable set to the paths were the java bin directory or where the `JavXSE.jar` is located, then you must use *absolute paths*.

Step 2. Create a New Application Server

Once the **Server Configuration** utility is started, the interface console will appear in a new window. It is divided into tabbed panels indicating the different options and fields available for configuring the new server.

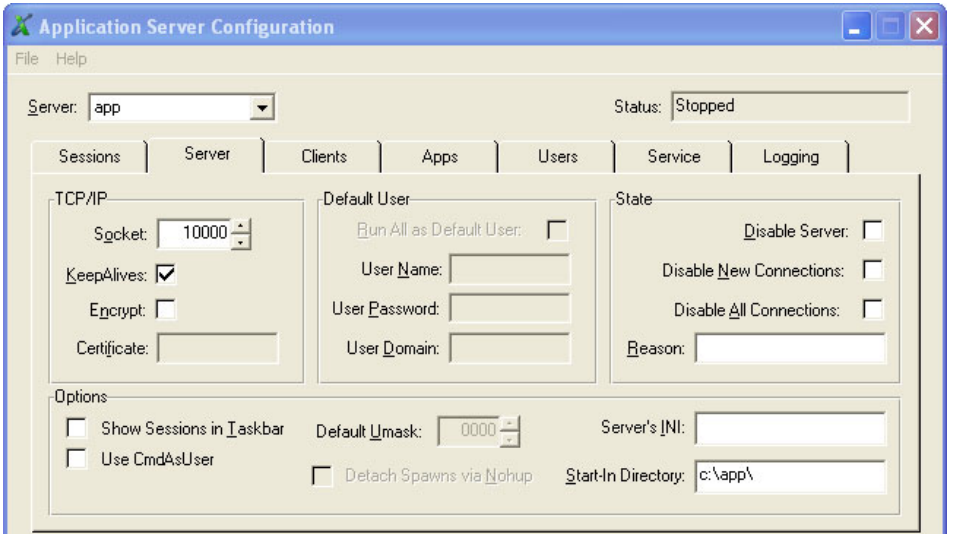


Enter a new name in the Server: field (e.g., app) then select New at the bottom of the main console.

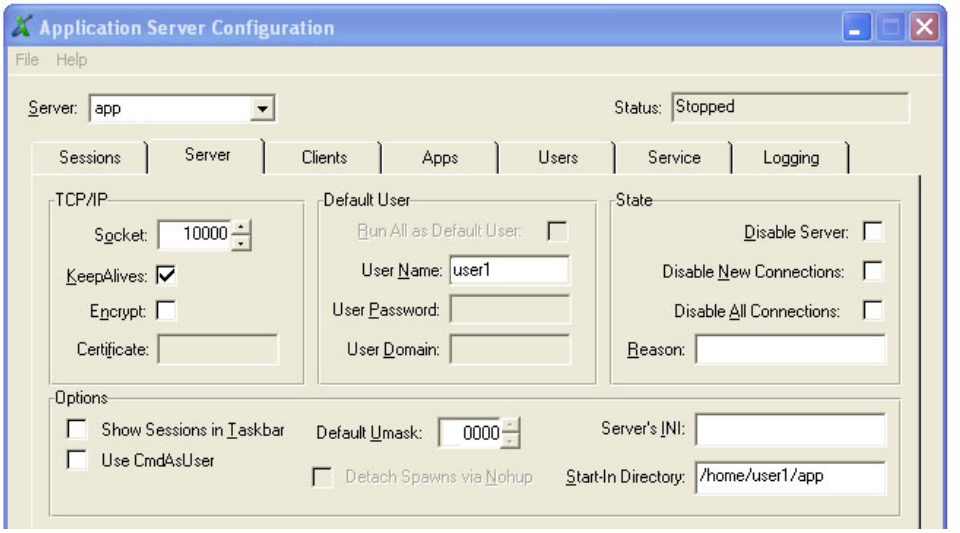
Step 3. Specify Server Properties

Click on the Server tab to begin entering the primary attributes for the currently-selected server.

MS Windows: In the Server tab/panel, keep Socket: 10000, select the KeepAlives checkbox, and change the Start-In Directory to "`C:\app\`".

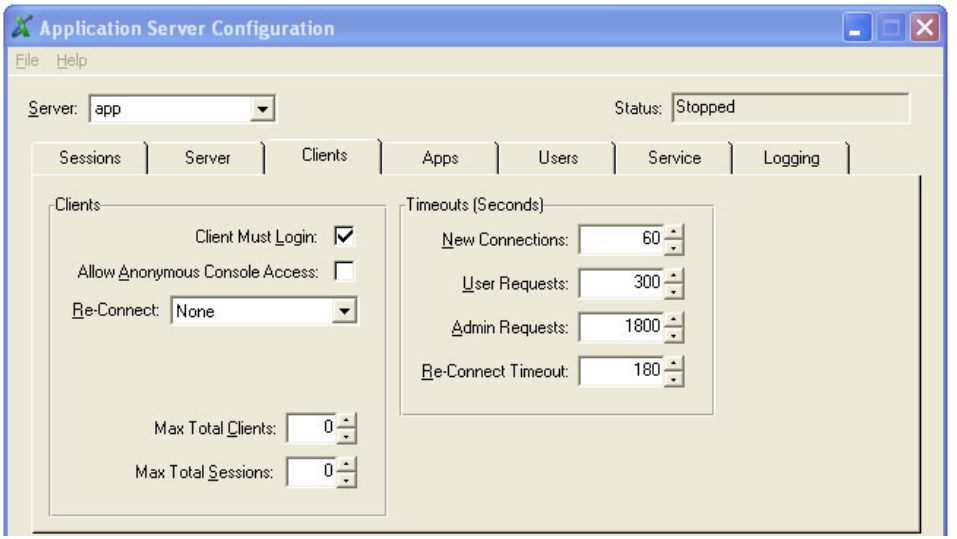


UNIX/Linux: In the Server tab/panel, keep Socket: 10000, enter (*Default User*) User Name: `user1`, and change the Start-In Directory to "`/home/user1/app`".



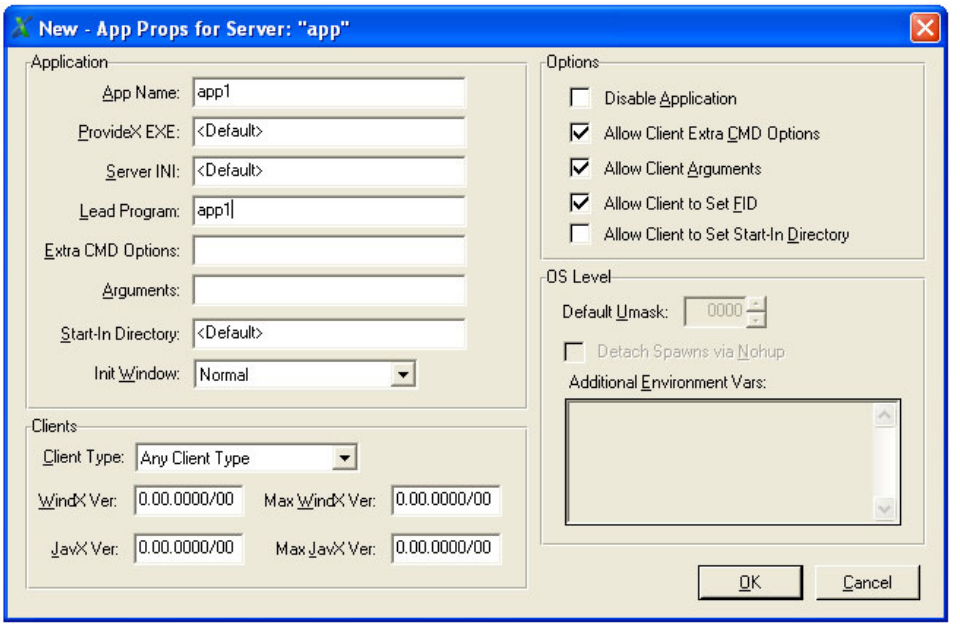
Step 4. Define Client Preferences

In the Clients tab/panel, ensure that the Client Must Login checkbox is selected, and keep the following (default) information:



Step 5. Configure Application Properties

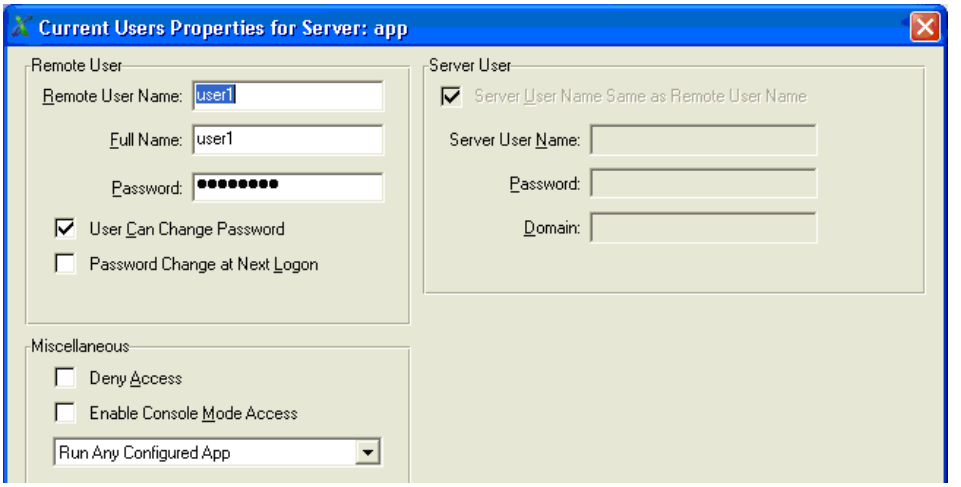
In the Apps tab/panel, select the New button. In the New - App Props panel, enter App Name: app1, Lead Program: app1, and Client Type: Any Client Type (from the drop list).



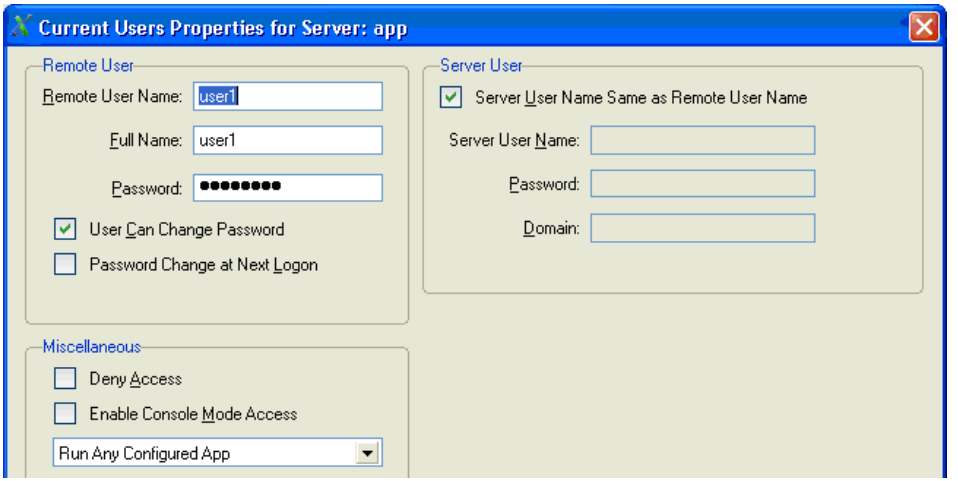
Step 6. Configure User Properties

In the Users tab/panel, select the New button.

MS Windows: In the New / Current Users Properties panel, enter Remote Users Name: `user1`, Full Name: `user1`, and Password: `password`. Select the User Can Change Password checkbox. Select Run Any Configured App (from the drop list).

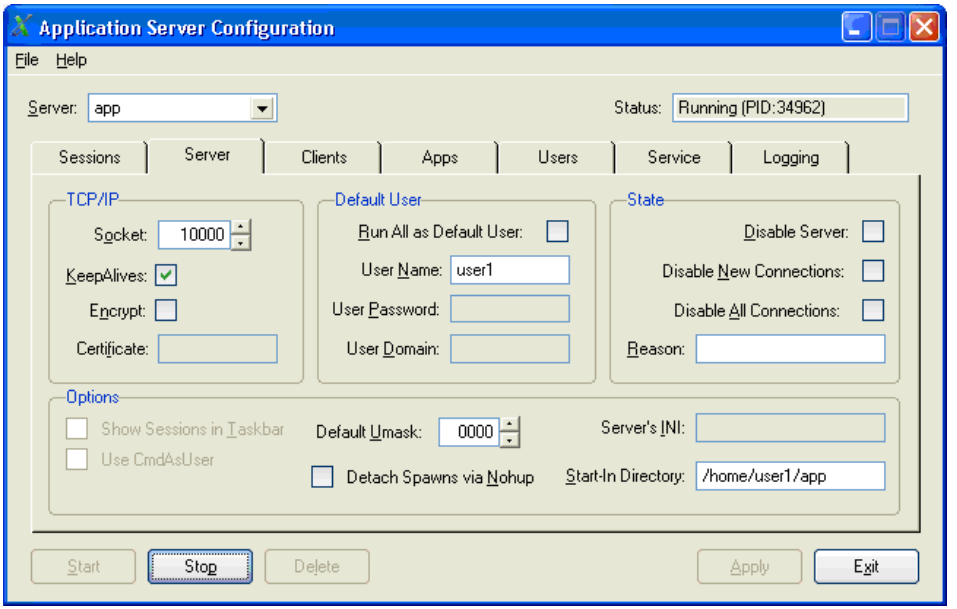


UNIX/Linux: In the New / Current Users Properties panel, enter Remote Users Name: `user1`, Full Name: `user1`, and Password: `password` (i.e., valid user ID and password). Select the User Can Change Password checkbox. Select Run Any Configured App (from the drop list). Select the Server User Name Same as Remote User Name checkbox.



Step 7. Start the Server

In the Server tab/panel, select the Start button to run the server..



Once the server has started, a PID (Process ID) will appear in the Status: box to indicate that the server is now running.

Step 8. Run the WindX Client

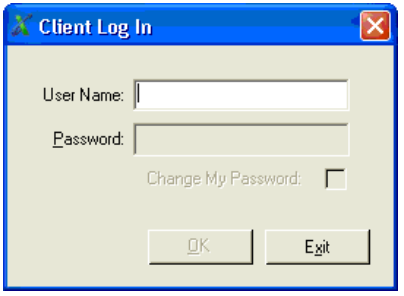
On the client system, ensure that WindX (Plug-in or Standalone) is installed. Create a shortcut that includes a Target similar to the following:

```
"c:\pvx\pvxwin32.exe" -mn "*client" -ARG 1.160.10.240 10000 "app1"
```

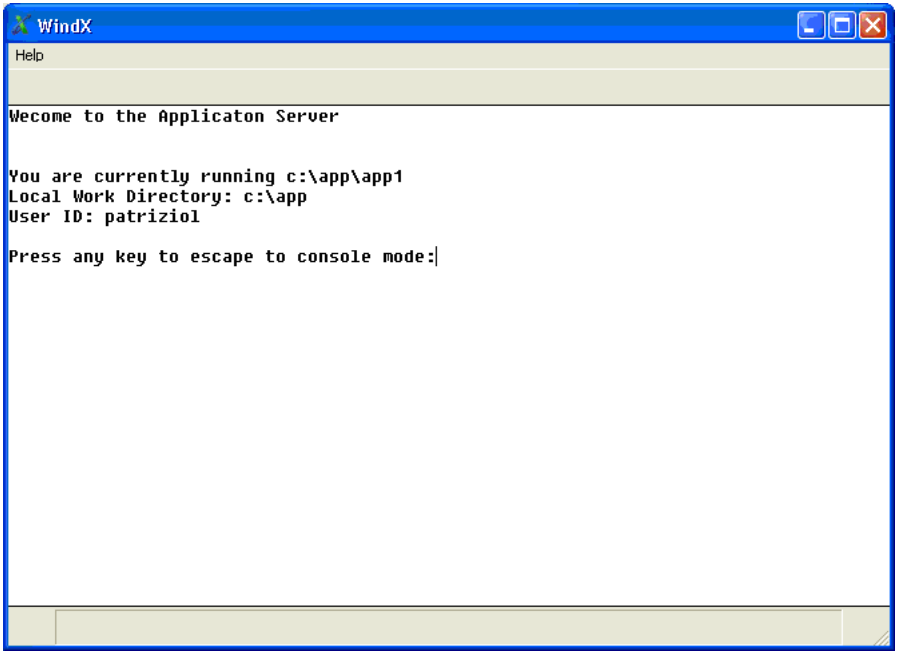
Where

`"c:\pvx\pvxwin32.exe"`	***Example path*** to ProvideX on the client system.
`-mn`	Starts minimized.
`"*client"`	ProvideX program used to connect the client to the Application Server.
`-ARG`	Keyword marking the start of the argument list
`1.160.10.240`	***Example IP address*** of the server.
`10000`	TCP/IP port (socket) that the server daemon is listening to.
`"app1"`	Lead program that the server is to run (explained at the beginning of the procedure, *p. 124*)

When you click on (run) the shortcut, the following dialogue will appear:



Enter User Name: `user1` and Password: `password` then click OK to connect to the application server. If the connection is successful, the WindX session will start up in a new interface window (console), as indicated below:



Troubleshooting

This section provides general troubleshooting information regarding some of the issues you may encounter when running the Application Server.

Network Connectivity

Verify IP connections if the client is unable to communicate with the server. Even if a client has successfully connected to the server in the past, several events could cause unsuccessful attempts at any time; i.e., changed TCP/IP settings, router problems, DNS services down, duplicate IP address on the network, etc.

Server-Side Issues

Difficulties can be caused by several server-specific issues; i.e., incorrect permission settings, daemon not running, invalid daemon command line syntax, socket in use by another application, daemon starting in the wrong directory, etc.

The most common problems involve UNIX ***file permissions***. When the application server's configuration system is first run, and every time the software is upgraded, the `*.pvk` files in the `lib/_appserv` directory are either created or recreated as required. If the user that you are logged in as has not set a `UMASK 000`, then these files may not be writable by other users. The `lib/_appserv/sessions.pvk` requires Read/Write file permissions for all users for whom sessions will be launched.

File permissions on the ProvideX ACTIVATE.PVX file must be Read/Write and all utility programs must have at least Read permissions for the user the sessions will be run under.

Client-Side Issues

Verify that the client configuration is using the correct domain and password. The client may fail if the server is unreachable, which is a symptom related to **Network Connectivity**. The client's *firewall configuration* may be the cause of a failure if programs are unable to receive data from the server.

6 AutoUpdater

The ProvideX base activation includes a utility, called the ***AutoUpdater***, that provides an automatic method to check for and update client applications whenever they connect to a copy of ProvideX on the server. It offers several features, including:

- Repository based system
- Software to be pushed resides on the server
- No client software installation required
- Customizable to include your software requirements
- Requires V7 ProvideX on server, with a V7 activation key
- Performs upgrades, downgrades or repairs
- Intelligent restart capability if session is severed
- Configurable user interaction
- Automated COM and DLL registration
- Detects and shuts down other client sessions currently invoked on workstation that are using the same executable
- No client reboot required after an upgrade, downgrade, or repair
- Windows 9X, NT, 2000, XP, Server 2003 compliant
- Logging capability.

The AutoUpdater supports WindX (stand-alone or plug-in Version 4.11 or higher) and ProvideX with WindX (Version 5.10 or higher). This chapter documents the configuration and functionality of the ProvideX AutoUpdater.

Topics

Activation and Product Components

The AutoUpdater is shipped with ProvideX Version 7 (or higher) and it requires a Version 7 (or higher) activation. Once ProvideX is installed on your server, a new directory will be created called **_updater** under the **lib** directory. This directory contains 4 files and 1 directory:

`cinfo`	Information utility to be placed on the client workstation.
`updater.pvc`	AutoUpdater engine.
`windx.upd`	Main program called by the Windows terminal device for talking to the AutoUpdater.
`autoupdate.conf`	Default configuration file for the AutoUpdater.
`images`	Directory of all the required images for the AutoUpdater screens which are to be moved to the client workstation.

The AutoUpdater software requires that a copy of the `autoupdate.conf` file (in the `\lib\_updater\` directory) be saved as `autoupdatecustom.conf` within the same directory on the server. We recommend that you create a copy – do not overwrite the `autoupdate.conf`.

Download the required repository file for the version of ProvideX that you wish to auto update. For more information, see **The Repository File**, *p.145*.

Enabling the AutoUpdater

To enable the AutoUpdater, open the newly-created `autoupdatecustom.conf` file in a text editor (e.g., Notepad) and change `AutoUpdate=Off` to `AutoUpdate=On` under the `[DEFAULTS]` section of the file. Save the `autoupdatecustom.conf` file. At this point, any new WindX sessions that try to connect to the ProvideX on your server will invoke the AutoUpdater software to start looking for updates.

AutoUpdater Configuration

The `autoupdatecustom.conf` file is divided into three main sections:

`[Default]`	Default values.
`[Debug]`	Debugging options or logging levels.
`[Product]`	Possible products to check.

The contents of the configuration file will appear similar to the following:

```
[DEFAULTS]
AutoUpdate=Off
Interactive=Prompted
AllowClientInitiation=Y
AllowUpgrade=Y
AllowDowngrade=Y
AllowCompare=Y
AllowClientCancellation=Y
CompareFrequency=D
LookAndFeel=4
KillProcessingMsg=Y

[Debug]
LogLevel=2
AutoUpdateDebug=0

[Products]
1 = ProvideX
2 = WindX Standalone
3 = WindX Plugin

[Product:ProvideX]
DisplayName="ProvideX"
CurrentVersion=6.50.0500
RepositoryPrefix="pvx"

[Product:WindX Standalone]
DisplayName="WindX (Standalone)"
Identifier=WXS
CurrentVersion=6.50.0500
RepositoryPrefix="pvx"

[Product:WindX Plugin]
DisplayName="WindX (Plugin)"
Identifier=WXP
CurrentVersion=6.50.0500
RepositoryPrefix="pvx"
```

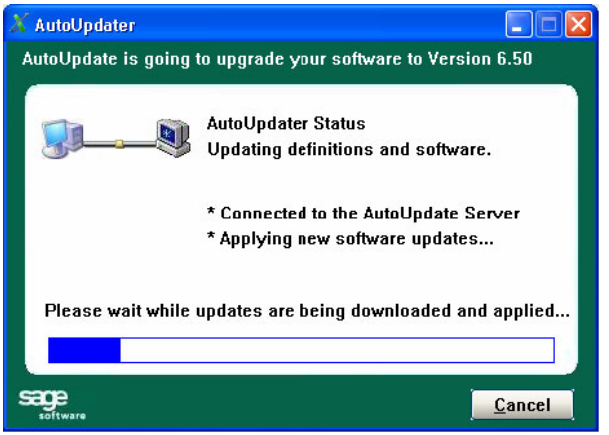
Configuration settings, options, and default values are described below.

`AutoUpdate=`

`Off`	***Default***. AutoUpdater is disabled.
`On`	AutoUpdater is active and ready.

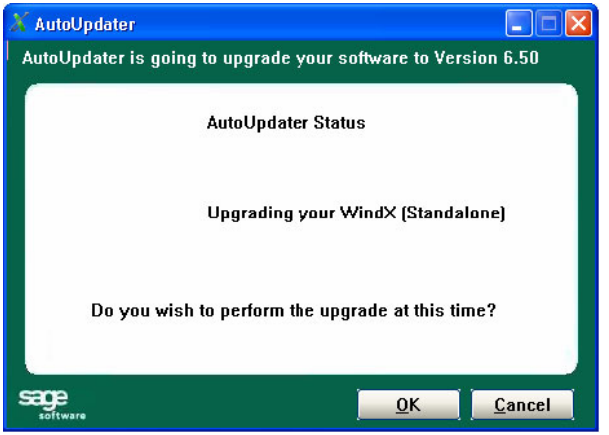
`Interactive=`

`Prompted`	***Default***. Display a dialogue to the user describing the status of the AutoUpdater and showing a progress bar (example below).
`Silent`	No information is displayed.



`AllowClientInitiation=`

`Y`	Display initial screen advising the user as to what is going to take place (example below).
`N`	***Default***. No initial screen displayed before performing task.



AllowUpgrade=

Y Perform ***upgrade*** if client workstation version is ***lower*** than version obtained from **CurrentVersion=**.

N ***Default***. Do not perform upgrade.

AllowDowngrade=

Y Perform ***downgrade*** if client workstation version is ***higher*** than version obtained from **CurrentVersion=**.

N ***Default***. Do not perform downgrade.

AllowCompare=

Y Perform ***repair*** operation if client workstation version is the same as the version obtained from **CurrentVersion=**.

N ***Default***. Do not perform a repair operation.

AllowClientCancellation=

Y Adds Cancel button that allows client to cancel an auto update in prompted mode.

N ***Default***. User is not allowed to stop the AutoUpdater.

CompareFrequency=

Compare if the versions are the same:

D Once a day.

W Once a week.

M ***Default***. Once a month.

nDay Day of the month – where *n* (1-4) and *Day* (mon, tue, wed, thu, fri, sat, sun); e.g., CompareFrequency=2tue is the second Tuesday of the month.

LookAndFeel=

Windows look and feel if Interactive=Prompted:

2 Windows 3.1 style.

3 Windows 95 style.

4 XP style.

A null defaults to the look and feel of the current session.

KillProcessingMsg=

- Y — Issues message box advising user that a session running with the same executable is going to be terminated in order to update files that may be locked or in use.
- N — ***Default***. No message is issued and the task is killed.

LogLevel=

- 0 — Logging disabled.
- 1-5 — ***Default***. Log the start and end routine messages as well as error messages.
- 6-9 — Log the start and end routine messages.

AutoUpdateDebug=

Internal use only. Values 1 through 7 will execute specific escapes during the product update program execution; i.e., in the `windx.upd` program.

DisplayName=

Name to display in the AutoUpdater Window if `Interactive=Prompted.`

Identifier=

Product ID used by AutoUpdater to determine which settings to read, as well as what files need to be moved from the `manifest.conf` file.

- PVX — ProvideX.
- WXS — WindX (Standalone).
- WXP — WindX (Plug-In).

CurrentVersion=

Value to be compared against the client workstation version as well as part of the key to the Repository directory location. Format is `MM.NN.PPPP` where:

- M — Major version.
- N — Minor version.
- P — Patch

RepositoryPrefix=

Used along with **`CurrentVersion=`** to identify which directory the AutoUpdater needs to look in; e.g.,

```
CurrentVersion=7.00.0000
RepositoryPrefix="pvx"
```

The AutoUpdater will use `\lib\_repository\pvx7.00.0000\` directory to obtain all manifest and files that need to be moved to the workstation

Note: Surround values in quotations " " to preserve the case.

MaxConnections =

Cap for the number of clients allowed to perform an Upgrade/Downgrade/Compare at the same time. For example,

```
MaxConnections = 20.
```

In this case, only 20 clients will be allowed to interact at any time with the AutoUpdater. The 21st will either be prompted with a choice to retry or procced with no update (if `Interactive=Prompted)` or will continue as normal (if `Interactive=Silent`).

Stime =
ETime =
DaysOfWeek =

These allow you to specify the start/end times and the day(s) of the week when updates will be permitted to occur, if required. If `am` or `pm` is not specified, the AutoUpdater will default to `am`. All times and days of the week are local to the Server. In the following example, auto updates will only happen between 1pm - 7pm on Monday, Thursday, Saturday and Sunday:

```
Stime = 1 pm
ETime = 7 pm
DaysOfWeek=1,4,6,7
```

FixAttributes =

Y This will change only the ProvideX files that need to be updated from read-only to read-write, which allows the AutoUpdater to push the live files without any user intervention.

N ***Default***. Users will be prompted with a message box advising them which files need to have their read-only permissions modified to complete the AutoUpdate.

Options for IP-Specific Configuration

Configuration options may be grouped to target specific IP addresses (or ranges of IP addresses). These appear as section headings similar to `[10.100.20.*]` or `[10.100.28.254]/2`. The following will target all clients that connect using an IP address that starts with "10.100" to have the flag `AllowUpgrade` set to `No`:

```
[10.100.*]
AllowUpgrade = N
```

This following method of masking allows you to specify how many bits from the right hand side of the IP address may be considered wild cards:

```
[10.100.28.254]/20
```

In this case, 20 bits from the right hand side may be any value; i.e.,

```
   10       100       28        254
00001010.01100100.00011100.11111110
<---Match---><--Anything---------->
```

This indicates a range of

```
00001010.01100000.*.*   =  10.96.*.*
```

to

```
00001010.01101111.*.*   =  10.111.*.*
```

Choosing IP Addresses for Upgrade, Downgrade or Compare

Specific IP address can also be included or excluded from an upgrade/downgrade or compare. The choices are described in the examples below:

`[UpgradeIncludeIP]`
`10.100.20.1`
`10.100.20.2`
`10.100.20.3`
	Sets `AllowUpgrade = Y` for the IP addresses `10.100.20.1`, `10.100.20.2` and `10.100.20.3`.

`[UpgradeExcludeIP]` `10.100.28.252`	Sets `AllowUpgrade = N` for `10.100.28.252`.
`[DowngradeIncludeIP]` `10.100.40.1`	Sets `AllowDowngrade = Y` for `10.100.40.1`.
`[DowngradeExcludeIP]` `10.100.50.1`	Sets `AllowDowngrade = N` for `10.100.50.1`.
`[CompareIncludeIP]` `10.100.80.1`	Sets `AllowCompare = Y` for `10.100.80.1`.
`[CompareExcludeIP]` `10.100.90.1`	Sets `AllowCompare = N` for `10.100.90.1`.

Configuration File Flow

The `updater.pvc` program processes the configuration information as follows:

1. The `autoupdate.conf` file is read top to bottom into a memory file.
2. The `autoupdatecustom.conf` file is read. If the custom file has a setting of `reset=Y`, then the memory file is purged and re-loaded with information from `autoupdatecustom.conf`. This overwrites any settings that might have been previously set.
3. The values located under `[Defaults]` are read and set.
4. The values located under `[Debug]` are read and set.
5. The `updater.pvc` program scans through the product list to locate a product that contains the same `Identifier` as the one that `cinfo` or `cinfocustom` found on the workstation. The items associated with that Product are read and set.
6. The values located under `[IP Address:MachineName]` are read and set.
7. The values located under `[UserID]` are read and set.

Example:

```
[DEFAULTS]
AutoUpdate=Off
Interactive=Prompted
AllowClientInitiation=Y
AllowUpgrade=Y
AllowDowngrade=Y
AllowCompare=Y
AllowClientCancellation=Y
CompareFrequency=D
LookAndFeel=4
KillProcessMsg=Y

[Debug]
LogLevel=2
AutoUpdateDebug=0

[Products]
1 = ProvideX
2 = WindX Standalone
3 = WindX Plugin

[Product:ProvideX]
DisplayName="ProvideX"
CurrentVersion=6.50.0500
RepositoryPrefix="pvx"

[Product:WindX Standalone]
DisplayName="WindX (Standalone)"
Identifier=WXS
CurrentVersion=6.50.0500
```

```
RepositoryPrefix="pvx"

[Product:WindX Plugin]
DisplayName="WindX (Plugin)"
Identifier=WXP
CurrentVersion=6.50.0500
RepositoryPrefix="pvx"

[10.100.10.10]
AllowUpgrade=N

[me]
AllowDowngrade=N
```

In this example, all clients connecting to the AutoUpdater server with the ProvideX or WindX (Standalone) products will receive an upgrade. Any client's connecting with the WindX (Plug-in) product or with an IP address of 10.100.10.10 and users with the ID of `[me]` will not be able to perform a downgrade.

How it all Works

The AutoUpdater works on the idea of a "client-to-server" relationship, where the server runs ProvideX Version 7 (or higher) with either the ProvideX **Application Server**, ***NTHost/*NTSlave**, or Telnet service. Clients may only connect using WindX (Standalone or Plug-in) or ProvideX with WindX.

The following will happen when a workstation connects to a server running ProvideX and the AutoUpdater correctly installed and configured:

1. `/dev/winterm` determines if this is WindX. If so, it calls `*updater/windx.upd` if present; otherwise, it continues as normal.
2. `windx.upd` takes control and begins checking which `cinfo` program exists on the server (`cinfo` or `cinfocustom`).
3. `cinfo` or `cinfocustom` is called on the client workstation. If this program doesn't exist on the client workstation it will be copied to the client workstation under `/lib/_updater/`.
4. `cinfo` or `cinfocustom` returns information about the client workstation.
   - `cinfo` version
   - IP address
   - Network ID
   - User ID
   - ProvideX version
   - Product ID of WindX (Standalone–Plug-in) or ProvideX
   - Current date and time.

5. `windx.upd` verifies the version of `cinfo` or `cinfocustom`. If the client version is the same as the server's, then the file from the server will be moved over and re-called.
6. `windx.upd` obtains all configuration information from the AutoUpdater engine (`updater.pvc` program). See **AutoUpdater Configuration**, *p.135*.
7. The client's version is compared against the version read from `autoupdate.conf`.
8. If the client's version is greater than the value stored in `autoupdate.conf`, a downgrade will be performed, if allowed.
9. If the client's version is less than the value stored in `autoupdate.conf`, an upgrade will be performed, if allowed.
10. If the client's version is equal to the value stored in `autoupdate.conf`, a repair will be performed, if allowed.

What Happens During an Upgrade, Downgrade, or Repair?

A file is created on the server and locked. This file resides in the ***updater** directory and uses information returned by (`cinfo` or `cinfocustom`) to build the name.

<IP Address><Network ID><ProvideX Version><Product ID>

Example: `10.100.10.10mybox6100000wxs`

This file contains the last successfully-moved file, which is where the AutoUpdater will pick up in the event of a connection failure.

If a client tries to start a new session using the same `pvxwin32.exe` to connect to the same server after an upgrade, downgrade or repair has been initiated, they will receive a dialogue indicating that a process is currently being used by the AutoUpdater and the session will time out.

During an upgrade or downgrade:

1. `windx.upd` calls `cinfo` or `cinfocustom` to obtain all information about who initiated this current session writes out how to re-launch the session in a text file (`autoupdatelaunch.txt`) on the client workstation under ***updater**.
2. `windx.upd` will get `updater.pvc` to load all manifest information into memory.
3. `windx.upd` spins through this memory file and decides if the currently-read file needs to be moved to the client workstation.
4. If the file's last modified version number is higher than the workstation version, then this file will be moved to the workstation under the `\*updater\tmp\` directory. In the case of a downgrade, ***all files*** will be moved.

5. If it is determined that a file will be moved, a ProvideX keyed file (`autoupdatecompare`) will be created with a header record indicating the date/time the AutoUpdater started, and containing records of each file moved (date/timestamp) with a checksum value to the size of the file.
6. After the manifest has been completed and read, `windx.upd` will then use `updater.pvc` to create a VBScript file (`autoupdatelaunch.vbs`) on the client workstation under the ***updater** directory.
7. `windx.upd` takes control, invokes the `autoupdatelaunch.vbs` file, and then exits.
8. `autoupdatelaunch.vbs` takes over and moves all the files on the client's workstation `\*updater\tmp\` to the live system. It rereads `autoupdatelaunch.txt` and restarts WindX with the options that originally started this process.

During a repair:

1. `windx.upd` opens and reads the header record of `autoupdatecompare` and verifies settings in the `autoupdate.conf` file that a repair needs to take place.
2. If it is determined that the repair is to take place, `windx.upd` reads each record found in `autoupdatecompare` and compares the date/timestamps as well as the checksum. If `windx.upd` finds a discrepancy, it will reload this file from the server's repository directory, and re-update this record as well as the header record with the last compare date.
3. After the `autoupdatecompare` file has been completed, `windx.upd` then invokes `updater.pvc` to create a VBScript file (`autoupdatelaunch.vbs`) on the client workstation under the ***updater** directory.
4. `windx.upd` is given control and invokes the `autoupdatelaunch.vbs` file. It then returns to `/dev/winterm` and continues as normal.

The Repository File

Repository Installation

Once the auto update has been configured, all that remains is to download the repository that you want your client system to be upgraded or downgraded to.

UNIX or Linux

We assume the default installation directory name of `/pvx.*`. From the ProvideX website, download the required `repository-providex-X.XXX.XXXX.taz`.

1. Copy the `repository-providex-X.XXX.XXXX.taz` file to `/tmp`.
2. Switch to the ProvideX `LIB` directory.
3. Enter the following commands to expand/install the software:
```
umask 0
cd /tmp
mv repository-providex-X.XXX.XXXX.taz repository-providex-X.XXX.XXXX.tar.Z
```
4. Extract `repository-providex-X.XXX.XXXX.tar.Z`.
```
cd /pvx/lib
tar -xvf /tmp/repository-providex-X.XXX.XXXX.tar
```

You should now have a `_repository` directory under the `LIB` directory.

Windows

We assume the default installation directory name of `C:\pvx.*`. From the ProvideX website, download the required `repository-providex-X.XXX.XXXX.zip`.

1. Copy the `repository-providex.6.50.0500.zip` file to `C:\tmp`.
2. Extract the `repository-providex.6.50.0500.zip` file to `C:\pvx\lib\` directory.

You should now have a `_repository` directory under the `LIB` directory.

Repository Manifest information

Once you have successfully deployed the repository file you will find a new directory called `_repository` under the `LIB` directory. This directory holds a sub directory that the `autoupdate.conf` or `autoupdatecustom.conf` file points to.

The default name for this subdirectory is **PVX***x.xx.xxxx*, where *x.xx.xxxx* represents the version of the files in this subdirectory.

Within this subdirectory is the `manifest.conf` file that holds all the information about the files; e.g.,

```
pvxcom.exe;Ver=5.10.2000;ASCII;ID=WXS,PVX;Reg=9;RegInstall=pvxcom.exe
      /unregserver;RegUnInstall= /unregservermycom.exe
```

Where

`VER` ProvideX Version number when this file was last modified.

`ASCII` Transfer format–either ASCII or Binary (***default***).

`ID` Product ID that are to receive this file – if not specified, then it applies to all.

`Reg` What needs to be done to the Windows registry.
`1` indicates register and unregister the file with `regsvr32`.
`9` indicates a custom expression is to be issued; i.e.,
`RegInstall=pvxcom.exe /unregserver;RegUnInstall= /unregserver`

`#` Remark line.

During an upgrade the `VER` information is compared against the version of the workstation. If the `VER` is greater than the workstation version and `ID` is equal to the Workstation ID returned from `cinfo` or `cinfocustom`, then this file will be moved to `\*updater\tmp\` on the client workstation. During a downgrade, all files are moved regardless.

Customizing the AutoUpdater

Adding custom files in the AutoUpdater is a two-step process.

1. Make your modifications to the repository directory for the version that the AutoUpdater is currently looking at.
2. Make your modifications to the `manifest.conf` file of the repository that the AutoUpdater is currently looking at.

Example:

In this example we will assume that `RepositoryPrefix="pvx"`, `Currentversion=7.10.0000`, and the install path is `/pvx/` on a UNIX box.

Create a new directory under **LIB** and add some programs and files. Our version of `database.pvc` is to be deployed to all workstations for a Version 7.10.000 upgrade.

Log on to the AutoUpdater server and point to the directory `/pvx/lib/_repository/pvx7.10.000/lib`. In this directory, create the new directory `/_mydir/` (i.e., `/pvx/lib/_repository/pvx7.10.000/lib/_mydir/`). Move the files, `myprog1` and `mydata1`, to this directory.

Open our version of `database.pvc` to the appropriate path of `/pvx/lib/_dict/`. Open the manifest file located in the repository that has just been modified.

There is no need to change the `lib\_dict\database.pvc` entry, since this file is already present and all we did is modify what program is to be deployed.

Add the two new files to the `manifest.conf`. Since we want these new files deployed only with ProvideX version 7.10.0000 and higher, set the VER option to 7.10.0000.

```
lib\_mydir\myprog1 ;VER=7.10.0000
lib\_mydir\mydata1 ;VER=7.10.0000
```

Now, any new clients that require an upgrade will receive our custom version of `database.pvc`, our new directory, and two files.

Troubleshooting

The AutoUpdater uses a VBScript to move files on the client's workstation from the `\lib\_updater\tmp\` directory to the live system after it has closed the current session. If your workstation is running any spyware detection software, it will advise you that a script is trying to execute called `autoupdaterlaunch.vbs`. This script must be allowed to operate so that the AutoUpdater can complete the update.

During the execution of the VBScript, some users might receive a message advising them that the directory the script is trying to move a file to (or file it is trying to replace) doesn't have write permissions. At this point, you will receive a message box alerting you to what directory is having this issue. Change the permissions and allow the script to continue to move files.