ProvideX

VERSION 7

JavX 2.5

Java-Based Thin Client

sage software

Publication Release: JavX Version 2.50

Java-Based Thin Client

Introduction

The ProvideX Java-based thin-client, ***JavX***, offers a platform-independent and multi-user solution for displaying and interacting with your server-based ProvideX applications.

JavX allows you to optimize software development and deployment to serve a diverse user base. With minimal changes to the ProvideX source code, you can leverage the JavX thin-client architecture to maintain heavy processing and data storage on a secure central server while delivering a flexible user interface to a variety of clients, from web browsers to 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**, which is designed for desktop platforms 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.29*.
- **JavX AE**, which is 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.44*.
- **JavX LE**, which is 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.44*.

No additional host software is necessary, apart from ProvideX. This minimizes the system resources required to service a large application. Bandwidth usage is also highly optimized. Only user interface information travels across the network, which greatly reduces network traffic (allowing for more concurrent users on a connection) and increases data integrity (data processing remains on the server).

ProvideX applications are easily adapted for use in JavX. System calls and functions that require access to the graphics engine are routed to the client station automatically. Graphical commands invoked by the application are tokenized and sent to the client for processing. The thin-client functionality is fully integrated into ProvideX, so it allows for a completely seamless environment to develop and run client-server applications.

About ProvideX Thin Clients

Historically, ProvideX developers have had two thin clients to choose between when delivering multi-user GUIs from server-based ProvideX applications: ***WindX***, which is written 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 based on a remote computer. Along with the Windows GUI, 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 is a more ***flexible alternative*** to the Windows-only thin client because it takes advantage of Java's portability and platform independence. However, it is not possible to recreate the entire Microsoft Windows environment within a Java framework, so some WindX functionality cannot be provided 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 JRE, the Java is a free download that can be installed just prior to running JavX.

For specific differences between WindX and JavX, refer to the section **JavX Thin-Client Functionality**, *p.13*.

Choosing the Right Solution

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

With more choices available, developers need to be sure that they are applying the correct client-server implementation. Each client type has its own set of advantages in functionality and flexibility. So, which is the right solution for your client-server application? Refer to the **Thin-Client Requirements Table** provided on the following page.

Accessibility vs. Expressiveness

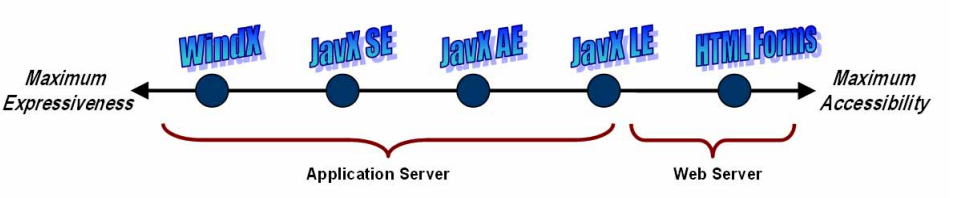
Choosing the appropriate thin-client requires an analysis of the trade-offs between ***accessibility*** and ***expressiveness***. The most expressive applications are the least accessible, and wider accessibility comes at the cost of limited expressiveness.

The richest full-featured GUI applications require the closest ties to a specific platform, which places strict limits on accessibility. When an application is designed to run on a wide variety of platforms, then GUI functionality has to be abstracted away from the client machine, which sacrifices expressiveness.

To provide the most expressive client-server application, the best ProvideX solution might employ WindX to deliver a powerful Windows-specific GUI. Conversely, you could employ the ProvideX Web Server 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 machine to deliver a rich graphical environment (along with auto-update capability) from any remote ProvideX host system to any 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 have control over which end-user platform is to be used.

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.29*.

JavX AE The *JavX AWT Edition* provides a simpler GUI designed specifically for handheld devices which 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.44*.

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 Portable Devices**, *p.44*.

HTML Forms A ProvideX WebServer / HTML implementation allows for a basic web 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 some form fields that are presented in a typical web page.

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.*

Prerequisites and System Requirements

For optimal results, the JavX installation requires a secure, stable TCP / IP connection, a ProvideX host facility, and the appropriate **Java Runtime Environment**. These requirements are outlined below:

Licensing

All JavX editions use a similar licensing method to the WindX plug-in. JavX only connects to a Professional or eCommerce-licensed copy of ProvideX that is activated for Version 5 or higher. It uses one user slot from the server-side ProvideX user license for its initial connection. Subsequent sessions that are spawned programmatically from the same workstation will not require additional user license slots.

While freely downloadable, the use of JavX is subject to a License Agreement. For details, please refer to `JavXLicense.txt` available from **www.pvx.com**.

TCP/IP Connections

JavX only works with direct TCP / IP connections. It does not work with Serial Mode (RS-232) type connections. It interacts with the TCP / IP (Sockets) layer on the workstation and does not look at the type of connection. Therefore, any TCP / IP-based connection will work; i.e., Ethernet cards or dial-up network connections based on PPP or SLIP.

A feature called a `ConnectionString` may be used within the parameters of a JavX applet or the command line to have Java send a custom connection message to a server side host daemon created by the application designer.

Configuring the address and TCP socket number of the server and the name of the program to run is done by placing this information in the properties of the `<APPLET>`, `<OBJECT>`, or `<EMBED>` tags on the HTML page that defines the applet location.

ProvideX Host Facilities

One of the following ProvideX client-server connection systems must be installed and running in order for ProvideX thin client technology to work:

- *NTHost/*NTSlave facility supplied with ProvideX (*NTSlave is built directly into JavX and cannot be changed).
- *ProvideX Application Server*, full-featured client-server connection system.

For more information on the installation and configuration of *NTHost / *NTSlave, refer to the Direxions presentation *ProvideX Thin-Client Technologies*. Documentation on the *ProvideX Application Server* is available from the ProvideX website, www.pvx.com.

Secure Socket Layer (SSL): Support for TCP/IP-level SSL-encryption security protocol is also available. Prior to version 1.5 JavX and the Java plug-in did not support SSL, thus encryption of communications with the ProvideX host was not possible. Sun added support for SSL to version 1.4 of the Java plug-in. JavX version 1.5 also provides support for SSL but requires the *ProvideX Application Server* as the host (***not*** *NTHost), and the client machine must have a Java 1.4 or later Java plug-in.

Java Runtime Environment

As mentioned earlier, JavX (***JavX SE/AE/LE***) runs 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.29*.

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.44*.

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

Downloading / Installing JavX

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 on both the host and client side of your applications.

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.

Installation Methods

Depending on the implementation, the client requirements, and on 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.

Installation procedures specific to JavX SE are provided in the section **JavX for PC Platforms**, *p.29*. Installation procedures specific to launching JavX AE and JavX LE on a device are provided in the section **JavX for Portable Devices**, *p.44*.

General installation methods for **Launching JavX as an Application** and **Launching JavX as an Applet** are discussed in the sections that follow.

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.37*.

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.29*. 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.

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 *Page Generator* 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 **Page Generator Utility**, *p.41*.
- A folder containing all JavX documentation, including this document and documentation on using the JavX CE Install Wizard and the JavX Page Generator program.
- A folder called JavX Versions that contains the three different JavX jar files (JavX SE, JavX AE, and JavX LE).

- A folder called `JavX CE Install Wizard` that contains a ProvideX program that will install the J2ME versions of JavX (JavX AE and JavX LE) on a WinCE device (e.g.,Windows Mobile 2003).
- Optional Java 2 Runtime Environment (JRE). All three version 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 **Prerequisites and System Requirements**, *p.7*.

JavX Thin-Client Functionality

Once JavX is installed and ready for client/server deployment, ProvideX applications should be able to use JavX with minimal code changes.

Thin-client functionality has been integrated directly into ProvideX to create a seamless environment in which to create and run applications. Once the ProvideX session on the server recognizes the terminal as a ProvideX client station, it changes internal settings that allow graphical requests to be routed correctly. All graphical directives and functions invoked by the application are tokenized and sent to the client for processing.

This section begins with an overview of ProvideX thin-client functionality describing the **Standard Behaviour** and **Language Enhancements for Thin-Clients**, *p.15*. The advantages and/or limitations of the different ProvideX thin-client products is discussed under **JavX vs. WindX**, *p.17*. JavX-specific fuctionality, involving ***windx.utl**, **Image Support**, and the ProvideX **COM Interface** is covered in the sections that follow.

Note: If you are programming with JavX for the first time, we recommend that you download and install the **JavX Developer's Kit**, *p.11*.

Standard Behaviour Involving Thin-Clients

Thin-client 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

JavX responds 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 JavX 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 by JavX 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 JavX 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.

JavX supports a Turbo mode which allows them 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, JavX 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.

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, specifying channel 0 (zero), to retrieve thin client information.

`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. Refer to the **HTML Reference** for further details.
`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 **~** (tilde) or **^** (caret)

The 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.)

The 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 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"`. The logical font `"SansSerif"` is guaranteed to be available in any JRE. The Java JRE will map the logical font name `"SansSerif"` to an appropriate physical font on any platform. For example, on Windows `"SansSerif"` is mapped back to "MS Sans Serif".
- 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

While most ProvideX applications that work with WindX can be used with JavX, there are a few differences to be noted. Each JavX client (JavX SE, JavX AE, JavX LE) has additional restrictions or enhancements. Refer to the sections **JavX for PC Platforms** and **JavX for Portable Devices** for functionality that is specific to each client format. ***The functionality described in this section apply to all JavX editions.***

WindX runs in conjunction with a copy of ProvideX for Windows, so WindX can issue almost any command of which ProvideX is capable. On the other hand, JavX uses Java equivalents to match ProvideX language features and while it allows GUI ProvideX applications to run on virtually any machine, not all features are supported.

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

- Simpler World Wide Web application.
- Platform independence.
- *Look-and-Feel* flexibility.
- No local ProvideX copies required.

- High speed communications.
- Less data transmission required.
- Improved data integrity.

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.

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]...**). There are exceptions which have been specially programmed into JavX; e.g., Many of the `WINDX.UTL` routines are emulated in JavX. For a complete list of ProvideX programs emulated by JavX see ***windx.utl**, *p.20*. For details on how to call or emulate calls to ProvideX programs in JavX, see **Calls To Local ProvideX Programs**, *p.20*.
2. ***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 **Printing via JavX SE**, *p.32*, 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.22*.
6. Format specifications within multilines (**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: When the ProvideX server side software 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. Enhanced **'FILL'** mnemonic for gradient shading. Gradient shading will change gradually from the color specified in the **'FILL'** mnemonic to the color specified in the last **'PEN'** mnemonic. Fill patterns #4 and #7 are different between WindX and JavX. WindX uses cross-line and crosshatch fill for patterns #4 and #7, whereas JavX uses gradient shading.

   Use fill pattern #4 to gradient fill from left to right. Use fill pattern #7 to gradient fill from right to left.

2. **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 **Printing via JavX SE**, *p.32*.

3. **'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")
```

WindX Utility Program - *windx.utl

JavX supports use of the utility program *windx.utl. This facility provides several functions to simplify the development of applications for thin clients. While designed originally for use with WindX, the following *windx.utl functionality is also available under JavX.

`call "[WDX]*windx.utl;get_addr", x$`

Returns the IP address of the client workstation.

`call "[WDX]*windx.utl;get_tcb", x`

Returns the value of the TCB function specified by *x* in x; i.e., *x*=TCB(*x*).

`call "[WDX]*windx.utl;get_val",x$,y$`

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

`call "[WDX]*windx.utl;get_num",x$,y`

Evaluates and returns the value of the 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

Calls To Local ProvideX Programs

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 to run ProvideX programs on the local machine. However, for compatibility purposes, it can *emulate* calls to local ProvideX programs. 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 comma 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.24*.

Launching Multiple JavX Sessions on a Client

Use the following syntax to spawn a new session of ProvideX on the host, and an associated JavX session on the client:

```
call "[WDX]*windx.utl;spawn", x$,w$,f$
```

Where:

x$ specifies the program to run on the host
w$ 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 *w$* is "#embed", the new JavX session appears ***in*** the web browser; otherwise, JavX creates a new window ***on top*** of the browser. For more information, see onapplet=true described under **HTML Parameters**, *p.37*. 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.

The following syntax is similar to;spawn but it detaches the session from the main user task so that if the main task terminates, then the spawned task continues executing.

```
call "[WDX]*windx.utl;spawn_nohup", x$,w$,f$
```

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 (i.e., {!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.9*.

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 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 **About ProvideX Thin Clients**, *p.4*. The other JavX editions are documented in the section **JavX for Portable Devices**, *p.44*.

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.9*.

If JavX SE is deployed as an application, several automated install 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.35* 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.

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 in the section **Installation Methods**, *p.9*. 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 **Printing via JavX SE**, *p.32*. 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.31*.

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.35*.

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 *Page Generator* 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 **Page Generator Utility**, *p.41*.

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.

Printing via JavX SE

JavX SE's ability to run as an unsigned applet on any JRE-enabled platform restricts its ability to access 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.

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.

Specific Features not Supported in JavX SE

The JavX SE architecture has the following limitations in addition to those described in the section **JavX vs. WindX**, *p.17*.

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.15*.

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 To Long"
   ```

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

   ```
   BUTTON 10,@(5,5,5,5)="<html>This Text is To 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" (*Activate implied decimal*)
"S" (*Scroll*)
"i" (*Suppress implied decimal*)
"R" (*Right Justify*)

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

1. Some **GRID** properties are not supported under JavX SE: **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 grid 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"`

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 window that logically attaches to the current window (i.e., leaves the current window active and shares controls)

***** Asterisk - creates 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 **'4D'** mnemonic is set, and the current "look-and-feel" is set to "Windows" or "os default". 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.

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 **Page Generator Utility**, *p.41*. 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.
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 the (default value is 50).
fontsize=	Text planes 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>
                </COMMENT>
                No JDK 1.2 support for JavX
            </NOEMBED>
        </EMBED>
        </OBJECT>
        <INPUT style="border: 0px;" ReadOnly id=xx maxLength=0 name="JavXField"
     type=text notab>
</form>

<SCRIPT LANGUAGE="JavaScript">

function focusToJavXApplet ()
```

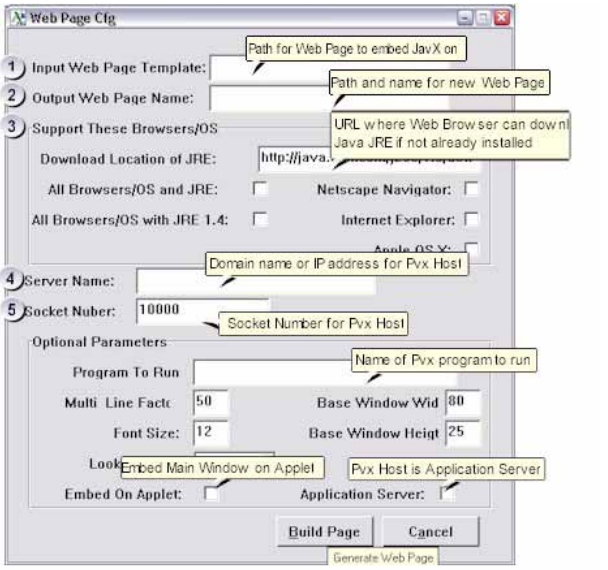
```
{
// This pulls focus away from the applet so the next stuff can
// put it back. It seems without this every second try does not
// restore focus to the applet
hide.JavXField.focus();
document.JavXApplet.jsForceFocus();
document.JavXApplet.requestFocus();
}

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

Page Generator Utility

The ***JavX Page Generator*** (JDK) is a convenient, easy-to-use utility that automates the process of embedding JavX in web pages. The JDK file, `Javxpagegen.pvs`, is included with the **JavX Developer's Kit**, which 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 start the *Page Generator*. The central Web Page Cfg panel appears as follows:



In this illustration, mandatory fields are indicated with numbers on the left and most of the optional fields are described briefly by floating tips. Full details on the installation and use of `Javxpagegen.pvs` can be found in the *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 problems you may encounter.

Problems Launching JavX as an Applet

If you have generated a web page using the **Page Generator Utility**, 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. Unfortunately, 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 new JavX editions have been developed to allow ProvideX client-server applications to be run on mobile devices: ***JavX AE*** and ***JavX LE***. These JavX products run under the *Java 2 Micro Edition (J2ME)* specification, which is a version of Java that has been optimized for constrained / embedded implementations.

For a discussion on the full range of client-server options, refer to the section **About ProvideX Thin Clients**, *p.4*.

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.

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

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

• 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) cans 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 sold for $5.99 and is a featured product listed under the category of *Development Tools*. 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 PocketPC and the Sharp Zaurus.
- CLDC Mobile Information Device Profile (MIDP) for palmOne Tungsten and PocketPC devices.

JavX AE and ***JavX LE*** should work in any runtime environment that has been designed within Java 2 Micro Edition (J2ME) specifications. However, in the event that JavX is not supported under your device-supplied JRE, try downloading and installing 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; therefore, the following procedure concentrates on one of the more popular platforms, Windows CE (Windows Mobile for Pocket PC).

Installation Procedure for Windows CE

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

1. Download and Install IBM's JRE for Windows Mobile 2003. A download link is provided under **WebSphere Everyplace Micro Environment (WEME)**, *p.46*. This is a large download because it includes numerous tools.
2. Unzip the downloaded file to a directory on your desktop, then run the launchpad.exe program. This presents you with a menu that allows you to install several JREs. Install WEME Personal Profile 1.0 for Windows Mobile 2003.
3. Create a shortcut called JavXAE (or JavXLE) and edit the shortcut with ***Notepad***. Add the following to the `JavXAE.lnk` shortcut file:

   ```
   147#"\Program Files\J9\PPRO10\bin\j9.exe"  -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$* see the section **Launching JavX as an Application**, *p.10*. Alternatively, omit the *ArgString$* 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.11*.

4. Using Windows Explorer, create a directory called JavX in the root directory of your device.
5. Copy the `JavXAE.jar` (using ActiveSync) into the JavX directory you created in the previous step.
6. Copy the JavXAE shortcut (using ActiveSync) into the device's \Windows\Start Menu\Programs directory.
7. On the PocketPC select Start > Programs > JavXAE. You should then see a JavX ProvideX console on your PocketPC.

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 **Installation Procedure for Windows CE**, *p.47*. 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) **Java Runtime Environment** (JRE). 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.

Avialable Features

For a general overview of the features available in all versions of JavX, refer to the sections listed under **JavX Thin-Client Functionality**. Since JavX AE is built on the same core classes as JavX SE, the same limitations will also apply to JavX AE. These are described in the section **JavX SE Deployment**, *p.30*.

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.

Specific Features not Supported in JavX AE

JavX AE has the following limitations in addition to those described under **JavX SE Deployment**, *p.30*:

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. **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.

3. **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.

4. ***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.

5. ***Grids.*** JavX AE does not support grids.
6. **BUTTON**, **RADIO_BUTTON**, **CHECK_BOX.** Most **OPT=** options are ***not supported***. Only **"D"** (*disabled*), and **"H"** (*hidden*) are supported.
7. **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.
8. **'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.

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) **Java Runtime Environment** (JRE). 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 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 AE 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 **JavX Thin-Client Functionality**. JavX LE is built on the same core classes as JavX SE and JavX AE, the much of the same functionality and limitations will apply to JavX LE.

Specific 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.48*. 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.