JavaPOS™ FAQ

What is an FAQ?

An FAQ is a list of frequently asked questions. In addition to supplying background material, this document will provide answers for the most frequently asked questions about JavaPOS, posted by visitors to our website (see http://www.javapos.com).

What is JavaPOS?

It was recognized early on that the emergence of the Java language on the computing scene offered several major advantages to the developers of retail applications.

The JavaPOS (Java for Point Of Sale) standard committee was formed by a collection of retail vendors and end users to examine the ways in which these Java advantages could best be exploited in the retail environment.

The original JavaPOS programming standard (v1.2) was the end result of ten months of effort by this committee. It was followed by JavaPOS v1.3, which defined Java interfaces for three additional retail POS devices (Fiscal Printer, PIN Pad and Remote Order Display), and JavaPOS v1.4, which added the Credit Authorization Terminal (CAT) interface requested by Japanese retailers.

What is the relationship between the OPOS and JavaPOS standards?

The OLE Point of Sale (OPOS) standard architecture was used as the starting point for the JavaPOS effort. There were several reasons for this approach.

OPOS was a good first step

The primary goal of OPOS is to permit retail application developers to be independent of the proprietary details (ex: special escape sequences) of the retail peripheral devices they accessed.

This was the starting point for the larger goal facing the JavaPOS committee: to permit the retail application developer to be independent of the proprietary details of BOTH the peripheral devices they access AND the POS platform on which the application itself runs. For example, the JavaPOS standard eliminates the OPOS dependency on the NT Registry.

Reuse of Retail Peripheral Device Models

Over 90% of the voluminous OPOS documentation is devoted to specifying the properties, events, methods and error codes of the seventeen (as of OPOS v1.3) defined retail peripheral devices.

These device models are both language AND platform independent, which allowed their direct incorporation into the JavaPOS standard.

Reduced Learning Curves

Many retail application developers already had experience using the OPOS APIs, and many retail hardware vendors had experience implementing the OPOS Control APIs.

Adopting this approach therefore reduced the learning curve for the very audience JavaPOS was targeting.

Two-Phase Deployment

By sharing the same device models as OPOS, it becomes possible to use a generic “OPOS bridge” to map existing OPOS Device Controls and Services into their JavaPOS equivalents. One example of such a bridge has long been made freely available by Wincor Nixdorf, and others have been created by NCR and Datafit Corporation.

This bridging capability provides a means of prototyping and testing JavaPOS-compliant retail applications on existing OPOS terminal configurations.

What is the relationship between the UnifiedPOS and JavaPOS standards?

The Unified Point of Service (UnifiedPOS) committee was formed by the Association for Retail Standards (ARTS) under the auspices of the National Retail Federation (NRF), at the request of leading retailers in the United States. It was created to ensure that future (post v1.3) releases of JavaPOS and OPOS would continue to share the same POS device architecture (i.e. each defined device type would possess the identical set of properties, methods and events in both standards).

The resulting UnifiedPOS retail device standard (see http://www.nrf-arts.org) is primarily defined in Unified Modeling Language (UML) and is thus both O/S independent and language neutral.

The UnifiedPOS technical committee draws its membership primarily from the JavaPOS and OPOS committees. Both committees have agreed that any new retail device type (ex: ForeCourt) will first be modeled in UML and added to the UnifiedPOS standard. Only after such UnifiedPOS approval is obtained, will the device UML be mapped to the Windows/OLE platform (by OPOS) and to the Java platform (by JavaPOS).

As a result, starting with the v1.4 version, a formalized procedure was put in place to ensure that both the OPOS and JavaPOS specification will continue to contain the identical interfaces (properties, methods and events) to an identical set of retail devices.

What are the advantages that JavaPOS brings to retail Point of Sale?

POS systems which are conformant to the JavaPOS standard provide several significant advantages for the retailer.

Reduced POS Terminal Costs

Applications written in the Java language execute by having a Java Virtual Machine (JVM) interpret their platform independent byte codes. These applications will therefore execute wherever such a JVM is present

By lowering the minimum requirements for a Point of Sale terminal to a system capable of supporting a single JVM, the JavaPOS standard enables retail applications to be run on thin client platforms which are often less costly than more traditional Windows PC configurations.

Platform Independence

The JavaPOS standard utilizes the JVM as its retail platform, whether present in a browser, an operating system, or directly embedded within the microcode of a specialized computer chip.

This language-centric platform is decoupled from any hardware or operating system specifics. Therefore a JavaPOS-compliant retail application is capable of running equally well on a thick client Windows95/98, SCO, IBM 4690 or Linux operating system, or on a thin client SunRay, IBM 4690, Linux, Windows-CE or Palm OS based system.

The only remaining proprietary element in a JavaPOS-compliant retail application is then the application code itself.

Reduced administration costs of thin clients

The capability of the Java language platform to reside on thin clients offers additional cost savings to those sites that run JavaPOS-compliant applications.

If the client portion of a retail application is written as a Java applet or loadable application, all of the application code resides on the in-store server. This means that installing or upgrading the POS software on the server results in the automatic loading of the new software into each local POS terminal, when the terminal is next booted.

The system administrator need only install a retail application once, and it becomes installed everywhere in the store. Fix the application once, and it is fixed everywhere.

The absence of persistent storage on a thin client also eliminates the need to perform POS terminal data backup and recovery. However local disks could still be useful to cache pricing information and outgoing purchase transaction data, so as to allow the thin client POS terminal to continue to function effectively, even if the server connection was lost for an extended period of time.

For sites with large numbers of POS terminals, these advantages can result in a considerable savings in system administrative overhead costs.

Is JavaPOS a complete standard?

JavaPOS is a complete standard for retail point of sale systems in exactly the same sense that OPOS is.

It maps the UnifiedPOS retail device architecture to the Java platform. Thus a JavaPOS configuration encompasses most OPOS hardware/software configurations, as well as a wide range of others.

It standardizes the Java interfaces used by retail POS applications (the Device Controls) to access and control POS devices.

It standardizes the internal interface used by these Device Controls to locate and load the set of device drivers (Device Services) appropriate for the POS terminal configuration, via the Java Configuration Loader (JCL).

It standardizes the interfaces supported by these Device Services, which must respond to requests from the JavaPOS applications (as relayed through the corresponding Device Control), and pass back asynchronous events.

It references the JavaComm API which allows pure Java Device Services to locate and access their respective peripherals for RS 232 type devices. For other device attachment options such as RS 485 and USB, the Device Service must (currently) still access the individual OS-specific port protocol routines directly.

Working implementations for all the above JavaPOS components except the Device Services (typically supplied by the device manufacturer) and of course the JavaPOS application itself, are freely available (either directly or through linkage) on the JavaPOS website.

What are the exact JavaPOS Deliverables?

The JavaPOS standards committee has produced the following set of deliverables:

JavaPOS Architectural Whitepaper

This whitepaper describes the philosophy and architecture of the JavaPOS standard and serves as an introduction to the other documents.

Java for Retail POS Programming Guide

This 600+ page document defines the JavaPOS standard. It is comparable in many ways to both the OPOS Application Programmer's Guide and the OPOS Control Programmer's Guide combined. It specifies the application-level Java language interface to the set of all defined retail peripheral devices, as well as the APIs for their corresponding Device Services.

JavaPOS Frequently Asked Questions (FAQ) List

Answers to a set of frequently asked questions concerning the JavaPOS standard. You are in fact reading this FAQ now.

JavaPOS Source Files

These files provide the foundation upon which JavaPOS-compliant applications and device services must be constructed. They are all available on the JavaPOS web site.

1. Codes

   The collection of all JavaPOS status and error codes

2. Exceptions and Events

   The set of all defined JavaPOS Exception and Event subclasses

3. Retail Device Controls

   The base Device Control Interface, and for each retail device category (ex: Scanner, Cash Drawer):

   a. A derived JavaPOS Device Control Interface (currently v1.4)

   b. A set of Device Control constants

   c. A completely implemented Device Control class

4. Retail Device Services

   The base Device Service Interface

   A derived JavaPOS Device Server Interface (currently v1.4) for each category.

   A sample device service.

JavaPOS Configuration / Loader (JCL)

The jpos.config / loader (JCL) is a simple binding (configuration & loading) API which allows a JavaPOS control to bind to the correct JavaPOS service in a manner independent of the actual configuration mechanism. For POS applications, it represents a somewhat minimum (but extensible) functional equivalent of the NT Registry.

All JavaPOS Device Controls (v1.4 and above) provided on the JavaPOS website will use this API.

A reference open source implementation of the JCL is also available on this website, maintained by IBM and the JavaPOS technical committee. This reference JCL comes complete with JavaDoc documentation, examples, sample code, a browser-based configuration editor and its very own FAQ.

Can I construct a 100% pure Java solution for retail POS?

By combining the above software on the JavaPOS website with the publicly available JavaComm v2.0 API and platform implementations (see http://java.sun.com/products/javacomm), it now becomes possible to construct COMPLETE Java based solutions for retail POS, that include platform-neutral retail applications as well as platform neutral device services.

Essentially:

- The Device Services use the JavaComm API to access device port data streams.
- The Device Controls use the JCL API to determine which Device Services correspond to the hardware devices connected to a particular POS station.
- The Applications use the various Device Control APIs to access and control these devices.

and the implementations for all three (as well as the source code for the last two) are, as indicated, openly available on the net.

How do you deliver JavaPOS on a Windows Terminal

There is a choice: you can either use Pure Java Device Services (which can support their peripherals on multiple platforms) or you can use an OPOS bridge. In either case, the JavaPOS application is unaffected.

Platform-neutral Java deployment

Windows 95 and Windows NT systems can be converted into JavaPOS platforms with the addition of the 100% pure Java JVM, and the Windows implementation of the JavaComm API (both freely available from the JavaSoft website).

By combining a full compliment of JavaPOS compliant Device Services with a JavaPOS compliant application, the entire retail system (application and device drivers) becomes portable.

It becomes possible for example to load Linux onto the same Intel configuration, reboot, and aside from editing a single configuration file, have the same application access the same set of retail peripherals, USING THE EXISTING SET OF DEVICE SERVICES. No code modification or recompiling is required.

Windows-only Java deployment

Alternatively, if you do not have pure Java Device Services for the devices you want to drive, but do have preexisting OPOS drivers, you can use a JavaPOS/OPOS bridge. Note that such bridges are not 100% pure Java, as they must invokes native code written in C.

Any true JavaPOS/OPOS bridge must deliver the complete JavaPOS functionality to a Java POS application using OPOS drivers. It it possible to do so because the device functionality of JavaPOS exactly mirrors that of OPOS. Such bridges have been deployed by customers of Wincor Nixdorf and demonstrated by NCR and Datafit, and other companies have developed them as well

Note that JavaPOS applications utilizing a JavaPOS/OPOS bridge can be subsequently redeployed on another operating system (ex: Linux) without modification, although a new set of device services (based upon JavaComm, or specific to that OS platform) would have to be obtained.

When can I start implementing my own JavaPOS solution?

The JCL is available today as a 100% pure Java package to configure the POS terminal and provide linkage from the JavaPOS Device Controls to the Device Services layer. All the JavaPOS Device Controls are also implemented and available.

By adding an OPOS bridge over an existing set of OPOS drivers (see above), JavaPOS-compliant retail applications can be written today and deployed on Windows 95/98/NT systems.

Likewise, fully compliant JavaPOS solutions can be deployed today on Windows 95/98/NT, IBM's 4690 OS Version 2, Solaris and Linux, provided there is access to a corresponding Java Device Service for each configured POS hardware device on the station.

How do I get my comments in for consideration by the JavaPOS committee?

Post them directly from the JavaPOS web site. All such comments will be reviewed and responded to by members of the JavaPOS committee.

Is it possible to write efficient, robust 100% Pure Java device drivers?

The part of a device driver that needs to be so efficient that you might consider writing it in assembler, or at least C, is the low-level port handling. With Java, the handling of the serial, parallel or (soon) USB port itself is done within the Java Communications 2.0 API implementation. Support for this API is available today from JavaSoft for the JDK 1.1.4 release and above on Solaris or Windows, and is available externally for Linux.

The JVM which executes the Java byte codes of the device driver is compiled from native code for the target machine, using assembler or C, as the JVM writer deems appropriate.

The 100% pure Java Device Service is thus given platform independence and is responsible only for translating the byte streams between the program and the relevant port. For instance, it might interpret a program command "cut paper" as "Escape X Y".

The Java language is ideally suited to doing this kind of translation, and, in practice, delivers more than adequate efficiency, especially when the driver code has been precompiled or JIT (Just In Time) compiled. In the latter cases, the overall performance of Java is comparable to native-mode compiled C++.

How do I migrate from an OPOS to a JavaPOS solution?

A clear OPOS to JavaPOS migration strategy exists. Its overarching goal is to “free” existing POS applications from their dependency on the Windows operating system, and allow end users and ISVs the same flexibility in selecting an OS and hardware platform vendor that they have today in selecting among multiple suppliers for each POS device type.

In brief, the strategy consists of a few well defined steps:

1. Use an OPOS bridge to access existing OPOS Device Services. This allows JavaPOS-compliant applications to be written and deployed immediately which support the set of existing devices in the store. Such deployments will be limited only to thick-client Windows-based systems however.

2. As true JavaPOS-compliant Device Services appear (developed and tested on any platform which supports the JavaComm API ... including Windows) the OPOS bridge begins to “narrow”. Once the last pure Java Device Service appears, the OPOS bridge can be discarded altogether.

3. At this point the entire retail POS application (including the JavaPOS Device Services) may now be installed and run on ANY JavaPOS-compliant platform (thick or thin client) without further change.

How can I prepare for JavaPOS?

Three main areas need to be addressed in your IT planning for JavaPOS: infrastructure, asset reusability and organization.

For infrastructure, you will minimally need to support TCP/IP on the local area network in the store. If IP is not present you will need to develop an IP addressing scheme and determine how you are going to administer it. DHCP and DNS will provide lots of flexibility but you will need to architect addressing properly and consider the impact on your enterprise.

Depending on the size of your stores and your Wide Area Network (WAN) you may need an httpd process running on your in-store servers. You will need to budget head count and capital for the infrastructure improvements. If you are just getting started with an IP WAN make sure to address security issues up front. Firewalls and access lists will affect your network design and overall cost.

Asset reusability planning refers to determining if your existing registers and servers can be utilized. Depending on your requirements they may need to be upgraded or replaced. While your vendor may offer some assistance, you should dedicate some staff to prototyping and evaluating after market upgrade solutions. You have to consider both the hardware characteristics and the operating software you are going to use. The registers are likely to run a DOS variant or MS Windows now, with possibly Linux as an option for the future. The investigation effort should provide details for the capital plan.

If you outsource development the organizational impact is minimized. If you develop your own applications you need to consider if your staff is capable of taking on a new development paradigm. Determine the level of training your existing personnel need and whether you need to supplement their skills with new hires and/or consultants. It would be wise to pick a small project and use this as a training opportunity for your team. A kiosk project for example may be a good one to gain experience on before taking on POS. During the test project, validate everything; object design skills, development tools, operating environment, ...

You also need to determine if your organization's structure will complicate things and if so, whether you need to reorganize. An advantage of writing in Java is the ability to reuse objects across a variety of platforms. For example, a POS object could be used in a handheld terminal or in an internet application. If the POS, handheld and internet programming is done in three distinct groups you have to account for extending the object for specific needs and the impact one groups changes will have on the others. You may need to establish "owners" of reusable objects to avoid versioning problems while leveraging existing expertise.

What about international coverage?

With their international perspective, the JavaPOS-Japan committee members have ensured the applicability of JavaPOS to the Japanese market. A complete Japanese translation of all key JavaPOS documents is directly accessible from the JavaPOS website.

What are JavaPOS plans for 2000?

Under the current arrangement, all future retail POS device specifications will be issued by the UnifiedPOS committee, and then adopted simultaneously by the JavaPOS and OPOS committees.

The JavaPOS committee will therefore concentrate on:

- Mapping new UnifiedPOS device specifications to Java
- Organizing "connectathons" between developers of JavaPOS retail applications and device services
- Supporting the software (Device Controls and JCL) currently available on the JavaPOS website

- Providing a focal point for JavaPOS expertise and serving as the primary means of communication between JavaPOS developers