

JavaPOSTM Introduction:



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) standards 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.

Java Advantages

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, whether in a browser, an operating system, or directly embedded within the microcode of a specialized computer chip.

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 platforms which are less costly than more traditional Windows95/98 configurations.

When such POS terminals do not store persistent information they are referred to as “thin clients”. While thin clients may possess flash memory or local disk storage, these are commonly dedicated to providing “no-stop” operation by, for example, temporarily buffering POS data such as bar codes and prices.

The minimal configuration requirements of such thin retail clients allow the potential “sidegrading” (reuse) of existing store hardware (including Intel 486 computers) as Java Point Of Sale terminals.

Platform Independence

The JavaPOS standard utilizes the JVM as its retail platform. This language-centric platform is decoupled from any hardware or operating system specifics. Therefore any JavaPOS-compliant retail application will run equally well on a thick client Windows95/98, SCO, DOS, or JavaPC operating system or on a thin client JavaOS or Windows-CE system.

The only remaining proprietary element in such a retail application is then the application code itself.

Reduced administration costs

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.

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

Strategy

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

Similarities of Purpose

The primary goal of OPOS was 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 accessed AND the POS platform on which the application itself ran. For example, the JavaPOS standard eliminated 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 specifications are for the most part both language AND platform independent, which allowed their direct incorporation into the JavaPOS standard.

Reduced Learning Curves

Many retail application vendors already have experience using the OPOS APIs, so producing a similar set of JavaPOS APIs reduces their learning curve.

Many retail hardware vendors already have experience implementing the OPOS Control APIs, so this approach reduces their JavaPOS learning curve as well.

Early Deployment

By sharing the same device models, JavaPOS has made it possible to create a generic “OPOS bridge” for migrating existing OPOS Device Controls and Services into their JavaPOS equivalents.

This capability provides an early means of prototyping and testing JavaPOS-compliant retail applications.

Deliverables

The JavaPOS standards committee was chartered with producing 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. It is in fact what you are reading now.

JavaPOS Guide for Application and Device Programmers

This document defines the JavaPOS standard. It consists of two main sections.

The first is comparable in many ways to the OPOS Application Programmer’s Guide. It defines the application-level Java language interface to the set of all defined retail peripheral devices. It is targeted at both retail application and system developers.

The second is comparable in many ways to the OPOS Control Programmer’s Guide. It is targeted primarily at those system developers who will write a JavaPOS Device Service.

JavaPOS Frequently Asked Questions (FAQ) List

The JavaPOS FAQ provides answers to a set of frequently asked questions concerning the JavaPOS standard.

JavaPOS Source Files

This set of standard Java language 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. JavaPOS Codes

The collection of all JavaPOS status and error codes

2. JavaPOS Exceptions and Events

The set of all defined JavaPOS Exception and Event subclasses

3. Retail Device Controls

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

- a. A derived JavaPOS v1.3 Device Control Interface
- b. A set of Device Control constants
- c. A completely implemented Device Control class

4. Retail Device Services

These include the base Device Service Interface, and for each retail device category:

- a. A derived JavaPOS v1.3 Device Server Interface .

JavaPOS: Architectural Overview 2

JavaPOS defines a multi-layered retail architecture. At first glance it appears to be a two level model: applications sitting on top of devices. However a closer look reveals a bit more complexity.

The device layer is actually composed of two defined sublayers (controls and services), each with its own set of interfaces. An overview of the various layers within the JavaPOS architecture follows.

Applications

Both Java retail applications and applets sit at the top layer of the JavaPOS architecture. They use the set of Java Device Interfaces defined by JavaPOS to access and control the retail device hardware present on the platform on which they execute.

By conforming to this specified set of interfaces, the Java retail application becomes independent of any vendor-specific requirements associated with the POS devices. It also becomes independent of any POS terminal hardware and software operating system specifics as well.

Thus the Java retail application programmer can write it once, and it runs everywhere.

Device Controls

The topmost layer in the Device hierarchy is the Device Control.

Purpose

This layer is responsible for supporting the retail application's use of the Device Interface, and it decouples the application from the lower device layers.

Each Device Control defines a unique retail device category in terms of a group of properties, events and methods which parallel the OPOS definitions for that category (ex: "Scanner" or "Cash Drawer").

The JavaPOS committee will continue to define and make available a complete set of Java language Device Control classes with each subsequent release of the JavaPOS specification.

Packaging

The JavaPOS standard supplies these controls to conform with the standard Java language component model. As a result, any Device Control may be packaged as a JavaBean component.

This allows developers to use standard Application Builders from a variety of vendors when constructing their retail applications.

Versioning

A given Device Control will only change when a new release of the JavaPOS standard redefines its interface. As discussed above, each new JavaPOS release will include a standard set of Device Control Java source files, which can in most cases be incorporated directly into a retail application.

A version number, corresponding to the first JavaPOS release in which a given control interface was defined, will be part of the name of that interface. For example, a Device Control object implementing an interface called "ScannerControl12 " implies the object will support all control requirements imposed on the scanner device, as specified in version 1.2 of the JavaPOS standard.

Isolation

The Device Control layer isolates the retail application from the Device Service software which is associated with the platform rather than the application, and which operates directly on the device.

A typical Device Control can be considered as a fairly thin wrapper for its corresponding service. It has the following responsibilities within the JavaPOS architecture:

1. Properties

Make Device Service properties visible to the application, and notify the service if/when the application changes them.

2. Events

Propagate Device Service generated events up to the application level.

3. Methods

Forward all application requests for a device directly to the Device Service, and return status or repost any device exception directly back to the application.

Device Services

The Device Service occupies the lower layer in the JavaPOS Device hierarchy.

Purpose

This layer decouples the Device Control (and hence the application) from the specifics of the connected POS peripherals. The Device Service interface is used by the controls, and in many respects it mirrors the same interfaces these controls present to the retail application.

It is the Device Service layer which identifies the particular serial or parallel line or USB port over which the retail peripheral hardware it services is physically connected to the POS terminal.

The Device Service uses this port to implement the set of properties, methods and events defined by the JavaPOS standard, through the series of escape sequences and data it exchanges with the hardware peripheral.

Multiple Device Services can share classes between themselves to more efficiently implement services for complex retail peripherals. This capability would normally be used to support a POS Printer with a Cash Drawer kickout or a Bar Code Scanner with an integrated Scale.

Packaging

For every Device Control there is an equivalent Device Service. Unlike the control objects, which are really direct extensions of the retail application, Device Service objects are akin to device drivers, and can be thought of as being part of the system platform itself.

They are loaded into the POS terminal only when “connected to” by a Device Control, which in turn has been previously “opened” by the application.

Versioning

The version number of a Device Service is completely analogous to that of a Device Control. For example a service object implementing the “ScannerService12” interface implies the object will support all requirements of the Scanner Device Service as defined in version 1.2 of the JavaPOS standard.

Since Device Services are part of the platform while Device Controls are typically associated with the retail application, the possibility of a version mismatch can arise.

In general, the Device Control will “know” the minimum version of the Device Service it requires to support its functionality. If the service object has a lower version, the control must either return an appropriate exception when asked to initialize itself, or scale down its requirements to conform to the capabilities of the service object.

Utilizing this second approach might require replacing the corresponding Device Control class implementation which accompanies the JavaPOS standard, with one more customized to the needs of the particular application.

Platform API Requirements

3

The JavaPOS architecture described above depends upon the following set of Java APIs being available on the POS terminal platform.

Java Communications Interface

The Java Communications API allows the Device Service to identify those communication ports present on the physical POS Terminal platform (serial, parallel, etc.) which are connected to the retail peripheral devices (ex: Scanner).

Support for this interface on both the Windows and Solaris platforms is freely available from Java Software at:

<http://java.sun.com/products/javacomm>

and such support also comes bundled with the latest release of JavaOS for Business. It is composed of the following 2 classes.

CommPortIdentifier

A static object which is used to obtain the collection of available ports on the system (ex: “Serial A”, “Parallel A”, “Com 1”, “Com 2”, etc.). Some of the more important functionality supported by this object allows the client to:

- Get an enumerated list of all port names
- Select a specific port by name
- Get the type of a port (serial, parallel, USB, ...)
- Declare ownership of a specified port

CommPort

This object represents the port selected via the `CommPortIdentifier`. It can be accessed to obtain standard Java `InputStream` / `OutputStream` objects,

The Device Service uses these I/O streams to send command sequences to the retail peripheral device connected to the port, and to receive incoming data from it.

Java Service Loader (JSL)

The Java Service Loader can be considered an enhancement to the normal Java Class Loader. It decouples the name used to request instantiation of a Java object of a new class, from the network location, actual class name and server class path.

When supplied with the name of a retail device (ex: “Scanner”, “NCRScanner”, or “Scanner12”) by a Device Control, the JSL loads the corresponding Device Service object into the memory of the POS terminal and returns the reference to that object.

More detailed information on the JSL specification is posted on the JavaPOS website.

Availability

This 100% pure Java API is currently available on the JavaOS platform, and a beta version for Windows95/NT platforms will be posted to the JavaPOS website early in Q1 99. JSL support is also being incorporated into several additional operating systems.

Java System Database (JSD)

The Java System Database supports the naming requirements of POS Terminal platform configuration. It retrieves and stores configuration information on behalf of applications and Device Services.

Although originally designed to meet the specific configuration and management needs of the retail industry, the JSD API is proving general enough to be applicable to other device-centric platforms such as Banking ATMs and airline checkin, restaurant and hotel desk stations.

Availability

This 100% pure Java API is currently available on the JavaOS platform, and a beta version for Windows95/NT platforms will be posted to the JavaPOS website early in Q1 99. JSL support is also being incorporated into several additional operating systems.

Comparison with NT registry

Conceptually the JSD can be thought of as a platform independent equivalent to the NT Registry, although it provides a considerably richer interface. For example, JSD parameter values are objects instead of strings, and a rich device name aliasing capability is supported.

On a thick client Windows95/98 or Windows NT system, the NT Registry may in fact be used as a basis for the JSD's persistent device configuration information.

In addition, the JSD optionally provides in-store server management and control of the configuration information of multiple thin client POS terminals.

Database Population

The JSD is populated in one of two ways.

1. Dynamic Population

JSD updates are made during POS terminal start-up, to add any peripheral “plug and play” devices which are automatically self-discovered.

2. Static Population

The JSD API includes a set of population routines, typically invoked via a platform-specific “device configuration” script.

For thin POS terminals running JavaOS, this script is executed on the in-store server.

For POS terminals running Windows95/98 or Windows NT, this script would normally be written to reuse configuration data maintained in the NT Registry.

Database Internals

Regardless of how the JSD is populated, it creates and supports a complete “in-memory” configuration database for the POS terminal.

This database is structured as a hierarchical collection of entries. Each entry consists of a string name, a list of properties, and an associated Java object.

For example, the entry for the Scanner Device Service might have:

- Name = “Scanner Service”
- Properties = “Version: 1.3” and “Port: Comm A”
- Object = the Java class code for the specified Scanner Device Service

Usage

The capabilities of the JSD are used at several points within the JavaPOS architecture. The primary ones are shown below:

1. Java Service Loader

The JSD enables the JSL to identify which Device Service to load, given the service name specified by the Device Control.

2. Device Service

The JSD enables the Device Service to determine which CommPort to connect to, based upon the name of the actual retail device specified by the service.

3. General

The JSD supports a “temporary” area, which can be used to store and exchange data between cooperating threads in one or more levels of the architecture.

Summary

A complete platform for JavaPOS-compliant retail applications and device services requires only the presence of a Java Virtual Machine (JVM), support for the Java Communication API classes, and the presence of the 100% pure Java classes which comprise the JSL and JSD.

JavaPOS-compliant applications and device services then become completely independent of the proprietary nature of the retail peripherals, POS terminal hardware, and POS terminal operating system on which they reside.

Until support for the JSL and JSD APIs appear on the Windows95/98 platform (current estimate: early Q1 99), JavaPOS-compliant retail applications may be prototyped by using an OPOS bridge, available from several vendors, to access existing OPOS compliant device services.

One example of such a bridge is JOWE, offered by Sieman's Nixdorf. To get it:

1. Go to the JavaPOS website (<http://www.javapos.com>)
2. Click on "JavaPOS Europe"
3. Click on the Siemens Nixdorf logo

JavaPOS Futures

4

The JavaPOS committee plans to expend efforts investigating the following issues in the months ahead. The results of this research may influence future versions of the JavaPOS standard.

Additional Security Hooks

The additional security features of the JDK 2 release may be brought forward to allow assignment of security constraints to Java applications and applets. This would be particularly important with respect to managing device usage.

From a security point of view then, all JavaPOS applications would cease to be "identical" in terms of the authorization each is given to access POS devices.

Cross-JVM Communication Capabilities

At present, there are several different intercommunication mechanisms that may be used to connect Java applets or applications executing under separate JVMs (ex: RMI, JINI).

Standardization on a single intercommunication mechanism for retail, wrapped in an easy-to-use interface, is anticipated to be a deliverable feature of a future version of JavaPOS.

Tracking Future OPOS Releases

With the issuance of JavaPOS v1.3, complete Java mapping equivalents now exist for all seventeen OPOS v1.3 devices. JavaPOS v1.4 will map the additional device (Credit Authorization Terminal) introduced by OPOS v1.4.

After v1.4, all new retail POS device specifications will be defined first by the UnifiedPOS committee, and then mapped simultaneously by JavaPOS and OPOS.

A whitepaper is being developed which will provide an OPOS to JavaPOS migration strategy for existing retail systems.

Configuration Management & Administration

A future JavaPOS whitepaper is planned to address the issues involved in administrating thin client POS terminals in a retail setting.

Worldwide Standard

JavaPOS is a worldwide standard. A group of system suppliers, ISVs and retailers have formed the nucleus of the JavaPOS-Japan committee, and a similar JavaPOS-Europe group is also meeting. For further information on either, please refer to the appropriate sub-page off of

<http://www.javapos.com>.

Two key differences distinguish these committees from their OPOS counterparts:

Internationalization

Using their international perspective, the JavaPOS-E and JavaPOS-J committees are expected to significantly aid design efforts to map the new Java I18N capabilities onto the retail POS application space.

Retailer Involvement

Actively involving end-users from inception proved extremely beneficial in creating the JavaPOS specification here in the United States.

Initial membership in the JavaPOS-Europe and JavaPOS-Japan committees therefore also consist of leading retailers as well as system suppliers and ISVs .