

J/eXtensions for Financial Services (J/XFS) for the Java Platform

Frequently Asked Questions

March 28, 2000



**Developed by the members of the J/XFS Forum and agreed in the
CEN J/XFS Workshop**

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 J/XFS.

What is J/XFS?

J/XFS is the registered trademark for the "J/eXtensions for Financial Services for the Java platform" standard. Putting it another way, J/XFS is a platform-independent Java language standard, used to communicate with and control financial devices.

Essentially, it was recognized early on that the emergence of the Java language on the computing scene offered several advantages to the developer of financial applications.

The J/XFS standards was invented by a group of leading system suppliers (De La Rue, IBM, NCR, Wincor Nixdorf and Sun Microsystems), to examine the ways in which these Java advantages could best be exploited in the financial environment. The committee has worked on producing a J/XFS standard to define the interface between a 100% pure Java application and a wide range of financial devices.

Publically available since March 1999, the work on the J/XFS standard has been transferred in May 1999 to the European Committee for Standardization (CEN) and its Information Society Standardization System (ISSS) in Brussels, where a workshop has been established with additional companies and interested parties joining the work on the J/XFS standard (more information can be obtained by visiting the web at <http://www.cenorm.be/iss/Workshop/J-XFS/default.htm>). Since the standard will be based upon the Java platform, J/XFS-compliant financial solutions will be capable of being hosted on a variety of operating systems and hardware offerings, including both thick and thin client configurations.

What are the advantages that J/XFS brings to financial application development?

J/XFS provides six (6) main advantages to the developer of financial applications.

Java Language Advantages

The Java language supports an automated garbage collection facility which eliminates the necessity of freeing all unused memory, and the often difficult task of freeing it correctly. There are no pointers in the Java language, and all array references are "bounded", eliminating yet another common source of application instability.

As a result, experience has shown that applications written in Java tend to be more robust than those written in other languages, which is often a critical requirement for financial systems.

Java also includes a powerful multithreading capability built directly into the language. On large server systems, each application thread might be running under control of a separate CPU, thereby optimizing performance without requiring a change to the program code.

Reduced Client Configuration 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 financial client to a system capable of supporting a single JVM, the J/XFS standard enables financial applications to be run on thin client platforms, which are potentially less costly than more traditional thick client configurations. The minimal configuration requirements of such thin financial clients also allow the possibility of reusing existing bank hardware.

Reduced Client Administration Costs

The capability of the Java language platform to reside on thin clients offers additional cost savings to those sites that run J/XFS-compliant applications. If the client portion of a financial application is written as a Java applet or loadable application, all of the application code resides on the in-bank server. This means that installing or upgrading the J/XFS software on the server results in the automatic loading of the new software into each local client, when the client is next booted. The system administrator need only install a financial application once, and it becomes installed everywhere in the bank. Fix the application once, and it is fixed everywhere. The absence of persistent storage on a thin client also eliminates the need to perform client data and backup recovery. For sites with large numbers of such clients, these advantages can result in a considerable savings in system administrative overhead costs.

Device Independence

The J/XFS standard abstracts the vendor-specific data sequences used to control financial devices, into a set of properties, methods and events (PME) unique to each device type (Card Reader, Pin Pad, etc.). By conforming to these PME models, J/XFS-compliant financial applications essentially become independent of the underlying device hardware which they access. This in turn allows such hardware to be upgraded or replaced entirely, without affecting the application layer.

Location Independence

The Communication layer of the J/XFS standard ensures identical application access to financial devices, whether local or remote. This enables multiple teller stations to share centralized resources such as passbook printers and document scanners. These devices continue to operate correctly no matter how and where they are physically deployed. This device sharing is transparently provided by a J/XFS implementation, so neither the banking application nor a J/XFS Device Service need to take care of this functionality.

Platform Independence

The J/XFS standard utilizes the JVM as its financial 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 any J/XFS-compliant financial application will run equally well on a thick client system or on a thin client system. Furthermore, financial Independent Hardware Vendors (IHVs) can write a J/XFS-compliant device driver once, and have it identically access their hardware device whenever the driver is installed on Windows or any other J/XFS compliant O/S. The only remaining proprietary element in such a financial application is then the application code itself, or: "Write a J/XFS-compliant application once, run it anywhere™"

What is the relationship between the J/XFS and WOSA/XFS standards?

WOSA/XFS was created specifically to exploit Microsoft's Windows OS. The primary advantage of this standard was that it permitted financial application developers to be independent of the proprietary details (ex: special escape sequences) of the financial peripheral devices they accessed.

J/XFS will map the current set of defined WOSA/XFS financial devices to the Java platform. It will attempt to reuse the WOSA/XFS architecture (defined by a set of device specific properties, methods and events) whenever possible. By thus sharing a common device architecture with WOSA/XFS, J/XFS will reduce implementation costs for vendors to support both standards, as well as provide a clear migration path to the Java environment for financial client-side applications.

Since many financial application developers already have experience using the WOSA/XFS APIs, this approach should reduce the learning curve for the very audience that J/XFS is targeted at.

J/XFS also provides the added benefit that conforming financial applications will be independent of the proprietary details of BOTH the peripheral devices they access AND the O/S and machine hardware on which they run. For example, the J/XFS standard eliminates the dependency on the NT Registry.

What is the relationship between the J/XFS and JavaPOS standards?

The Java Point Of Sale (JavaPOS) standard is the O/S neutral Java standard created to support access to, and control and management of, retail Point Of Sale devices. JavaPOS-compliant solutions can be deployed on both thick and thin client POS terminals. The standard includes a set of Java extensions contained in the Java Device Drivers Kit (JDDK) to specify:

1. The interfaces used by retail POS applications to control POS devices.
2. The interfaces supported by the Java device drivers which service requests from these applications.
3. The way in which these Java device drivers locate and access their respective peripherals.

The basic JavaPOS tri-level architecture (Application / Device Control / Device Service) has been incorporated directly into the J/XFS standard, along with the following additional components:

1. A Device Communication layer, which supports shared application access to remote financial peripherals such as a passbook printer, from multiple client stations.
2. A Device Manager which abstracts the functionality provided in the JDDK, so that alternative technologies which deliver device access, control and management, may be transparently supported.
3. A centralized error logging and function tracing capability.
4. A persistent repository, where configuration data is stored and used by applications and devices.

This approach allows J/XFS to provide applications with a "device view" that is nearly identical to that provided by JavaPOS. As a result, J/XFS helps to blur the distinction between retail and financial devices on the Java platform.

What is the relationship with the OpenCard Framework ?

OpenCard Framework (OCF) is a standard framework announced by an Industry consortium that provides inter-operable smart card solutions across many hardware and software platforms. The OpenCard Framework is an open standard providing an architecture and a set of APIs that enable application developers and service providers to build and deploy smart card aware solutions in any OpenCard-compliant environment. More information on OCF can be obtained from the web at <http://www.opencard.org>.

Through their combination of mobility and security, smart cards are playing an increasingly important role in the rapidly developing areas of electronic commerce and online information services. The OpenCard Framework is based on a network-centric view of computing where resources are located throughout the network and access to them is ubiquitous. Authenticated access and secure transactions are indispensable prerequisites for the secure functioning and continued growth of the information economy. Smart card aware applications and services will be the means by which this will be achieved. The OCF Card Services provides high level APIs to read and write data from and to card files, generate key pairs, import keys, sign data.

OCF may run on top of J/XFS, using J/XFS Device Controls for access to card readers and communication with smart cards. But also J/XFS can take advantage from OCF implementations, because a J/XFS Device Service may be programmed using an existing OCF CardTerminal implementation to access the smart card device.

What about deploying J/XFS compliant applications on Windows/NT?

It is true that to date, Microsoft has refused to support 100% pure Java applications on its Windows/NT platforms. In response both Sun Microsystems and IBM (among others) have developed powerful Java Virtual Machines designed specifically for Windows/NT, so that any 100% pure Java application can run on these operating system without change. Sun's JVM (plus associated compilers and debuggers) can be downloaded from the JavaSoft website free of charge at:

http://www.java.sun.com/products/OV_jdkProduct.html

IBM's JVM (plus a peer to peer socket benchmark program) can be downloaded from the IBM website free of charge at:

<http://www.ibm.com/java/jdk/download>

Any Java incompatibilities contained in the native Windows/NT JVM will therefore not be an issue for developers or deployers of J/XFS-compliant financial applications.

What are the actual J/XFS Deliverables?

The J/XFS standards committee was chartered with producing the following set of deliverables, which can be downloaded from the J/XFS website at the following URL: <http://www.jxfs.com/documentation.html>

J/XFS Conceptual Overview Whitepaper and Presentation

These documents describe the philosophy and architecture of the J/XFS standard, and serve as an introduction to the other documentation. They are targeted at financial technical managers, architects and developers.

J/XFS Architectural Overview Presentation

This technical presentation consists of a series of slides which highlight the J/XFS architecture, and is targeted specifically at financial application and system developers.

J/XFS Architecture Guide for Application and Device Programmers

This document defines the J/XFS standard. It is directed at two main audiences. Financial application developers can find application-level Java interfaces for the set of all defined financial peripheral devices. System developers, who write J/XFS Device Services, can find the device driver-level Java interfaces for the set of all defined financial peripheral devices.

J/XFS Device Class Interface Specifications

- X Printers (Receipt, Journal, Passbook, Document and Scanner)
- X Cash Dispensers/Cash Recyclers and ATMs
- X Pin Pads
- X Chip Card and Magnetic Stripe
- X Alarms
- X Text I/O
- X Depository Units
- X Check Readers and Check Scanners
- X Sensors and Indicators
- X Cameras

Specifications for Sorters and Counters as well as other banking devices will be covered in subsequent versions of the specifications, based on industry and customer requirements.

J/XFS Frequently Asked Questions (FAQ) List

Answers to a set of frequently asked questions concerning the J/XFS standard. You are reading this FAQ now.

Is a J/XFS thin client solution inherently less robust than a thick client one ?

Interestingly in either case a Java-based application is likely to be more robust than its C or C++ counterpart. This is due to the features of the Java language itself; in particular Java's support for automatic garbage collection, elimination of pointers and bounded arrays.

However assume that the "robustness" referred to here will be measured by the ability of an application to keep operating when communication has been lost with the in-bank server. To answer this question, both types of clients must be examined.

Thick Client

Financial applications running on a thick client can operate somewhat independently of the in-bank server because of information stored on local disks. This might include the financial application itself, account information, device drivers and peripheral configuration data.

The in-bank server would be accessed primarily to post transactions, and for retrieval of the global information necessary to support such features as interest calculation.

This relative independence comes at the cost of increased administrative overhead. This is due to the need to keep the local persistent information (both applications and data) current. Typically this requires some level of manual system maintenance to be performed directly upon each such client system in the bank.

Thin Client

Financial applications running on thin clients generally get all the information they require to perform their functions at boot time. This means that all data updates are made once to the in-bank server (ex: interest rate change), and automatically reflected in each client the next time it is booted.

However once the boot process is completed, there is no reason that a thin client application need be any less robust than its thick client equivalent.

Necessary information can be stored dynamically in flash memory or in a local caching disk. The same transaction buffering capability typically provided by financial application software works identically on both thick and thin clients. The J/XFS thin client advantage is that it eliminates the need to consistently perform disk backup and restore or disk upgrade administrative tasks on a thick client financial terminal.

Does the J/XFS standard make use of JavaBeans and/or Enterprise JavaBeans?

JavaBeans are client side Java components, with a well defined property, method and event model ("PME") that has been applied to the J/XFS Device Control interface. As a result, standard JavaBean application builders can be used in constructing J/XFS-compliant financial applications.

Enterprise JavaBeans (EJBs) on the other hand, are server-side Java components which normally contain the "business logic" for a multi-tiered application.

There is no direct connection between EJBs and the J/XFS standard at this time.

What are the J/XFS Dependencies?

The main objective of the J/XFS standard is the decoupling of financial applications from the I/O device subsystems by providing real hardware- and platform-independence. Therefore any requirement for platform-specific dependencies would be a contravention. There are only two pre-requisites for a J/XFS in any JVM implementation on any operating system:

The J/XFS standard is based upon JDK 1.1.6 and above. It will also require the availability of functionality equivalent to the Java Communication API, which allows "pure" Java programs to access, control and manage devices. Both are available on a wide range of operating system platforms.

What does compliance mean ?

Compliance means that any Java-based banking application can transparently work with any J/XFS implementation and any J/XFS Device Service for particular banking peripherals, regardless of the specific manufacturer of this device. While implementing a solution for banking applications and banking peripherals based on J/XFS, the requirements for compliance should be clearly defined. Based on such a definition, the various components (e.g. application, middleware, device driver etc.) for the desired solution can be chosen from different vendors (hardware manufacturers, ISVs, service providers etc.).

We distinct between two different kinds of compliance:

Application compliance

Application compliance means, that a Java based banking application can make a J/XFS API call to a specific banking device and the J/XFS implementation as well as the appropriate J/XFS Device Service (normally provided by the manufacturer) will understand the call correctly and all defined functions as defined in the J/XFS specifications are supported. Note that specific functions, which are generally not supported by that particular device, will generate a „Function not supported“ error message. The banking application itself must not care about the real hardware access, the transparency is achieved by the J/XFS implementation in conjunction with the J/XFS Device Service.

Device compliance

Device compliance means, that a J/XFS Device Service written in Java is able to understand the appropriate J/XFS API calls for that particular device (i.e. a statement printer) correctly and to support all functions as defined in the J/XFS specifications (as long as these functions are supported by the device itself). The J/XFS Device Service itself will handle the real hardware access in a transparent manner to the J/XFS implementation and the banking application making the J/XFS API call, regardless of the specific operating system platform the JVM runs on.

The above compliance definitions has been approved by the CEN/ISSS J/XFS Workshop in December 1999.

What is the relationship between the CEN workshop and the J/XFS Forum ?

The mission of the European Committee for Standardization (CEN) is to promote technical harmonization in Europe in conjunction with worldwide bodies and partners, because harmonization diminishes trade barriers, promotes safety, allows interoperability of products, systems and services, and promotes common technical understanding. This is achieved by respecting the principles of openness and transparency as well as the principle of consensus. The objective of a workshop under the auspices of CEN is to publish a CEN Workshop Agreement (CWA), which is a consensus-based specification drawn up in an open environment, with a minimum of bureaucratic rules, as a first attempt to become a European Standard.

There is a clear distinction between the CEN/ISSS J/XFS Workshop and the J/XFS Forum:

CEN/ISSS J/XFS Workshop

The objectives of the CEN/ISSS J/XFS Workshop is to work on specifications for the J/XFS standard. Although the J/XFS Forum members invented the standard, now the Workshop owns it and is responsible for the standard and its future.

Commercial and non-commercial institutions are members of the CEN/ISSS Workshop, but the workshop itself does not have a commercial interest. The deliverables of the workshop will be technical specifications, published as a CWA on the CEN/ISSS web site.

J/XFS Forum

The objective of the J/XFS Forum now is the development of a J/XFS implementation. The members of the Forum (currently DeLaRue, IBM, NCR, Wincor Nixdorf and Sun; other manufacturers and vendors are endorsed to join the group) own the source code and are responsible for future enhancements and maintenance.

The deliverables of the J/XFS Forum will be product quality code and the binaries will be made freely available on the Web.

Each member himself is responsible to develop and deliver the appropriate Device Service for its devices. Every other banking device manufacturer can do the same and we encourage hardware vendors to support the J/XFS standard by providing J/XFS compliant Device Services.

What are the Timeframes for the J/XFS Deliverables?

All J/XFS deliverables as produced by the original inventors are publically and free of charge available on the web at <http://www.jxfs.com/documentation.html>. The original inventors have transferred the J/XFS specifications to the European Committee for Standardization (CEN) and its Information Society Standardization System in Brussels, Belgium. A workshop with several participants has been initiated in May 1999 and has agreed to publish the J/XFS architecture and specifications as a CEN Workshop Agreement (CWA) in September 1999. Information about the member companies of this workshop can be obtained from the following URL: <http://www.cenorm.be/iss/Workshop/J-XFS/participws-jxfs.htm>.

Basically, as technical specifications these deliverables will have the same content as the documents mentioned above, but they will be available from the CEN/ISSS web site. The CEN/ISSS J/XFS Workshop will continue to work on the future of the J/XFS APIs. For more information on the CEN/ISSS J/XFS Workshop see the web at <http://www.cenorm.be/iss/Workshop/J-XFS/default.htm>

How does J/XFS provide a migration path ?

The J/XFS standard itself deals with the API specifications for new Java-based banking applications (ie. the call for a specific device is made in banking application written in Java) and device services complying to a device interface (i.e. a device driver written in Java working with a J/XFS implementation).

The J/XFS architecture itself allows the implementation of wrappers to existing Client/Server-based financial I/O subsystems (e.g. WOSA/XFS, LanDP or others) on operating system platforms where a JVM is present. By doing this, the usage of currently installed banking device equipment and supporting client/server infrastructure is also ensured as the co-existence between older client/server-based banking applications and newly developed applications based on Java. Such wrappers could be developed by manufacturers, ISV's or any other development organization familiar with J/XFS and the existing financial I/O subsystem.

When does a J/XFS implementation become available ?

Members of the J/XFS Forum have jointly worked on the development of an implementation, tentatively called J/XFS Kernel. This implementation will function as a kind of middleware between the banking application and the banking device itself and will implement all functions as described in the base architecture document for J/XFS. General availability for the J/XFS Kernel was March 2000. For news please check the J/XFS website at <http://www.jxfs.com/new.html>.

Despite this, an appropriate J/XFS compliant Device Service (aka device driver) for the specific banking device must be available, e.g. from the manufacturer or other service providers. Obviously, the banking applications itself (e.g. self-service or teller application) must use J/XFS API calls for accessing banking peripherals. Both, application developers as well as banking hardware manufacturers are encouraged to endorse the J/XFS standard by using its API in Java banking applications as well developing J/XFS compliant device driver.

What does a J/XFS implementation provide ?

The J/XFS Kernel fully implements the J/XFS standard as defined in the architecture and API specification documents. It also provides a exchangeable communication layer, which enables the sharing of devices in a network.

The J/XFS Server has a server-based repository, where configuration data can be stored. As administration component it will provide a Basic Configuration Tool (BCT).

The complete J/XFS Kernel is developed in 100 % pure Java and, by doing this, this implementation of J/XFS provides full platform independence. Nevertheless, the primary target operating system platforms are Windows NT, OS/2 and Linux and their Java Virtual Machine implementations. On these platforms extensive testing has been done.

The supported scenarios cover the environments we find in todays bank branches as well as new hardware supporting the 'thin client' model.

The core functions supported by the J/XFS Kernel are:

- Asynchronous operation
- Event driven architecture
- DeviceControls as Java Beans
- Access to devices locally attached to other client machines (remote device access).
- Central configuration database
- Flexible Logging Concept
- Exchangeable communication layer

What are the J/XFS Kernel Deliverables?

The J/XFS Kernel deliverables provide the following components:

- Basic client side services incl. Device Manager
- Device Communication
- Device Controls
- J/XFS Repository
- J/XFS Logger
- Basic Configuration Tool and Log Viewer
- Installation Routines
- Sample Test Application and Sample (generic) Device Service