

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

## **Conceptual Overview**

---

March 28, 2000



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

## Preface

This document describes the concept of the J/XFS standard and is the result of a joint effort of leading system suppliers to define an architecture for financial enterprise computing in Java™.

The intention of this document is to give a conceptual overview of a Java based I/O-subsystem for banking peripherals called J/eXtensions for Financial Services (J/XFS) for the Java™ platform.

Utilizing the Java language, J/XFS provides a standard for a banking device subsystem with real platform and hardware independence which enables the access to banking peripherals for new Java banking applications. It also provides a migration path for current financial I/O subsystems and ensures co-existence between current Client/Server and new Java banking applications, so customer investments in banking device infrastructure are protected.

J/XFS enables full transparency between the application and the device level while providing a flexible and extensible solution for the network computing paradigm.

The information provided in this document was contributed by members of the J/XFS Forum and represents its current views on the issues discussed as of the date of publication. It is furnished for informational purposes only and is subject to change without notice.

The J/XFS Forum and its members makes no warranty, expressed or implied, with respect to this document.

The members of the J/XFS Forum encourage banks and other financial service companies, independent software vendors, and banking device manufacturers worldwide to get acquainted with the J/XFS standard. To submit questions and comments for the J/XFS specifications visit the J/XFS web site:

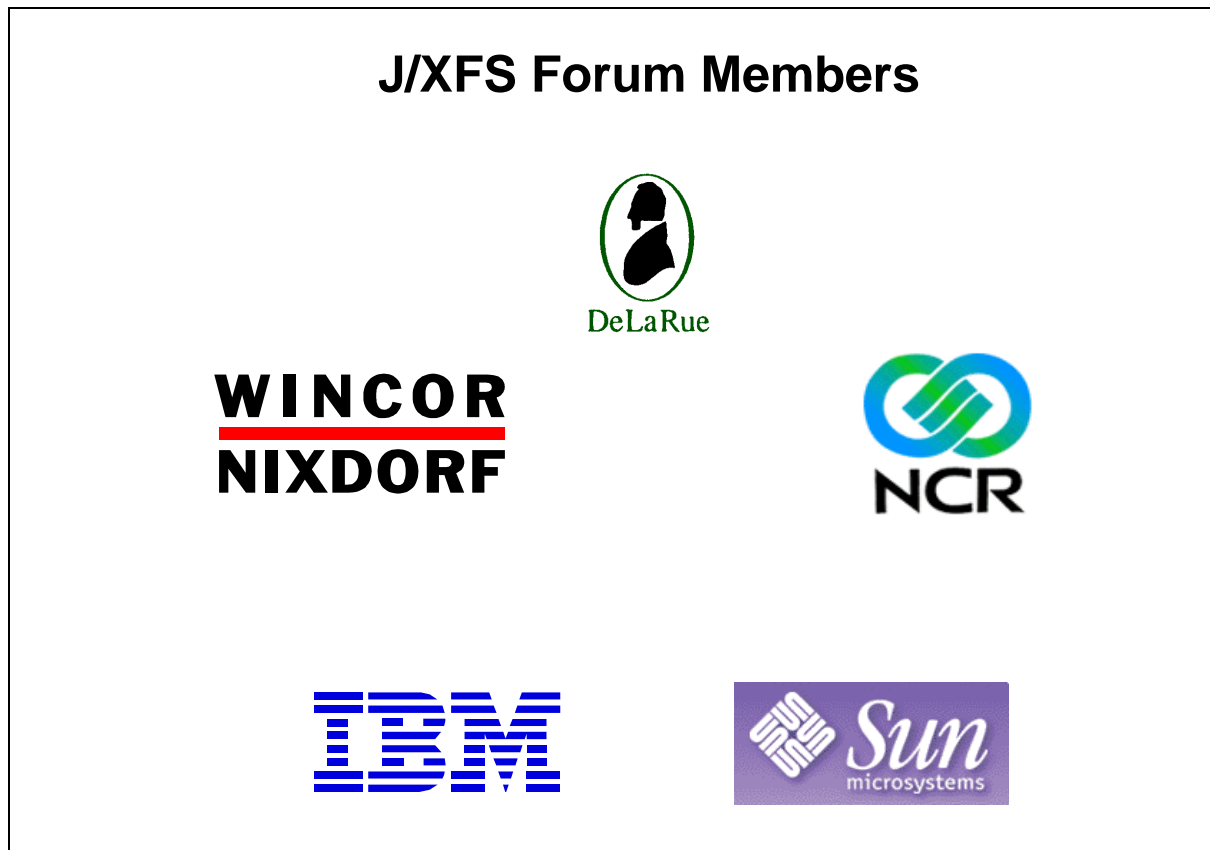
<http://www.jxfs.com>

Java and all Java-based trademarks and logos are trademarks of Sun Microsystems, Inc. The Java Trademark Guidelines are currently available on the web at [http://java.sun.com/nav/business/trademark\\_guidelines.html](http://java.sun.com/nav/business/trademark_guidelines.html).

All other trademarks are trademarks of their respective owners.

## Introduction

Recognizing the advantages of the Java language for developing finance industry applications, representatives of Sun Microsystems, Siemens Nixdorf Retail and Banking Systems, NCR, DeLaRue and IBM have embarked on an effort to develop a set of standard Java interfaces in support of the unique input and output devices used in the finance industry at various access channels including Branch Teller and Platform, Self Service and Call Center. This new standard is called J/eXtensions for Financial Services (J/XFS) for the Java™ platform.



The mission of this initiative was to produce a finance industry standard for financial I/O devices that supports 100% pure Java applications while leveraging existing standards. It was and remains to be our intention to bring together representatives from hardware vendors (network computers and I/O devices), software vendors and financial institutions (primarily banks) to help to develop and maintain the standard. The J/XFS specifications have been transferred to the European Committee for Standardization (CEN) and its Information Society Standardization System (ISSS) in Brussels. A workshop with several participants has been initiated in May 1999 and the J/XFS standard has been passed as a CEN Workshop Agreement in September 1999. The Workshop will continue its work on the future of the J/XFS standard and its APIs. For more information see the web at <http://www.cenorm.be/iss/Workshop/J-XFS/default.htm>.

## Advantages

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, given the application developer the advantage of being able to „write once, run anywhere". Therefore financial applications written in Java can be deployed at multiple access channels with minimal changes. In addition, Java applications will potentially run on platforms that are less costly initially and less costly to maintain.

In the „thin client" or network computing environment to which Java is targeted, the application can be written as a Java applet or loadable application, maintained at the server, and downloaded to the user machines when used. Thus the cost of maintenance and administration will be greatly reduced.

The advantages of Java applications in general will be carried over the support of finance industry specific devices, since J/XFS will be based on an open, nonproprietary architecture. Device drivers (device services or service providers) can be written once and run anywhere. In addition, just like Java applications, the Java device services can be kept and maintained at the server and downloaded as needed by the application.

J/XFS utilizes the Java language as the finance industry platform. This language-based platform is decoupled from any hardware or operating systems specifics. Therefore any J/XFS conformant financial application will run equally well on any client supporting Java, the only remaining proprietary element in such a financial application is the application code itself. Thus by being a 'vertical' solution based on the 'horizontal' Java approach, J/XFS provides real platform- and hardware independence.

J/XFS is a flexible and extensible solution providing full transparency between the application and the hardware level. The J/XFS standard provides device independence by abstracting 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. The Communication layer of the J/XFS standard provides location independence ensuring 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.

But the J/XFS solution will also provide a migration from existing Client/Server-based financial I/O subsystems by providing the ability to write wrappers (e.g. for WOSA/XFS solutions), which ensure the co-existence of 'old' and 'new' applications on the client as well as protecting the customer investments in hard- and software because the hardware can be attached from both applications.

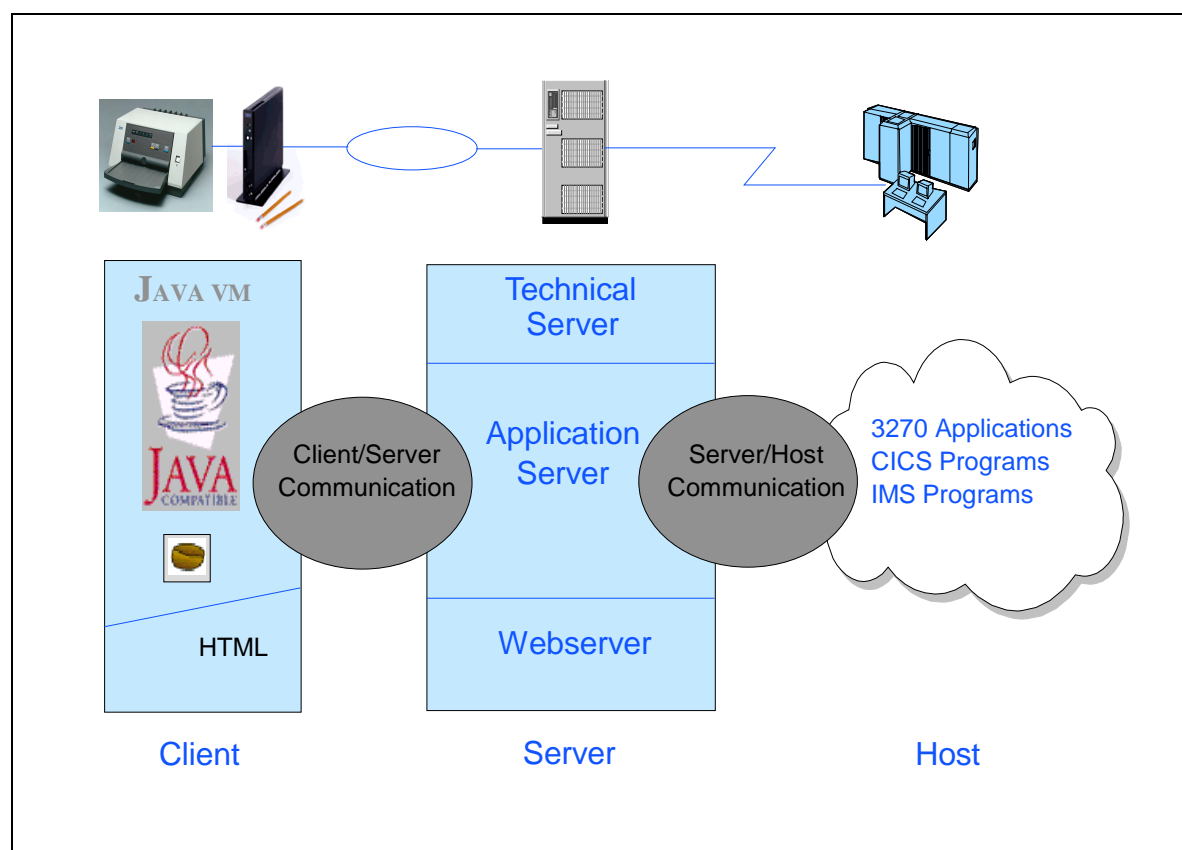
## Objectives

Currently an implementation supporting financial I/O devices requires the presence of a specific software and its prerequisites (controller, Intel-PCs with proprietary operating systems) and also a specific device driver for the particular I/O device. All these solutions are focused on specific hard- and software (and therefore mostly incompatible) coupling the financial applications with the I/O device subsystem.

Decoupling the financial applications from the I/O device subsystem and to provide hardware- and platform-independence is the overall goal and highly required by financial institutions.

The objectives of the J/XFS initiative are:

- Definition of an architecture for financial I/O device access from Java applications.
- Definition of a set of financial I/O device interfaces (APIs).
- Derive the J/XFS APIs from existing standards (e.g. WOSA/XFS and JavaPOS).
- Provide for an architected method for the sharing of peripheral devices within the J/XFS solution.
- Definition of the architecture and the APIs in such a way that they are platform independent but specific to the Java programming language.
- Lay the groundwork for endorsement and/or administration by a recognized standards body.



The scenario above shows an overview of the three-tier model leveraging the network computing approach.

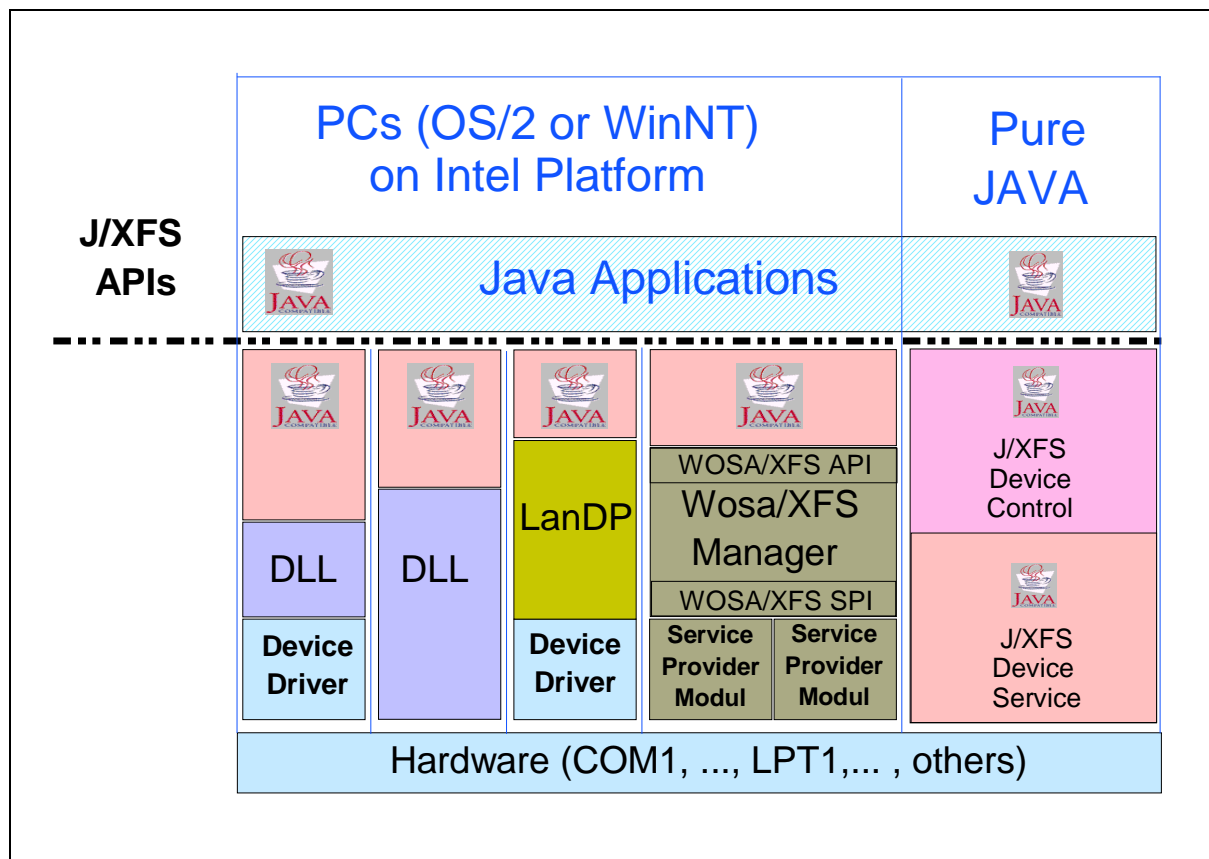
On the client side only Java and HTML will be used to enable a real thin client environment. There is no prerequisite for a specific, perhaps proprietary operating system. The communication between the client and the server is based on TCP/IP and may use HTTP, IIOP or RMI. The server acts as the Web, application, and technical server (note that this is a logical view and doesn't mean that all these functions have to run on one specific machine). The communication between the server and the host

system is based on standard architecture like TCP/IP, SNA or Corba and depends on the specific kind of connection to the given enterprise host servers (e.g. IMS or CICS, 3270 applications, etc.)

## J/XFS solution

Because of the need to provide an evolutionary way of migrating from various existing client/server solutions to the proposed new standard we need both a pragmatic approach to support these existing solutions as well as the need to build a 100% pure Java solution. This enables application programmers to focus on the development of new Java-based banking applications without any impact from the underlying infrastructure.

Below an overview is given to show both, the wrapping solution as well as the 100% pure Java-based solution for J/XFS:

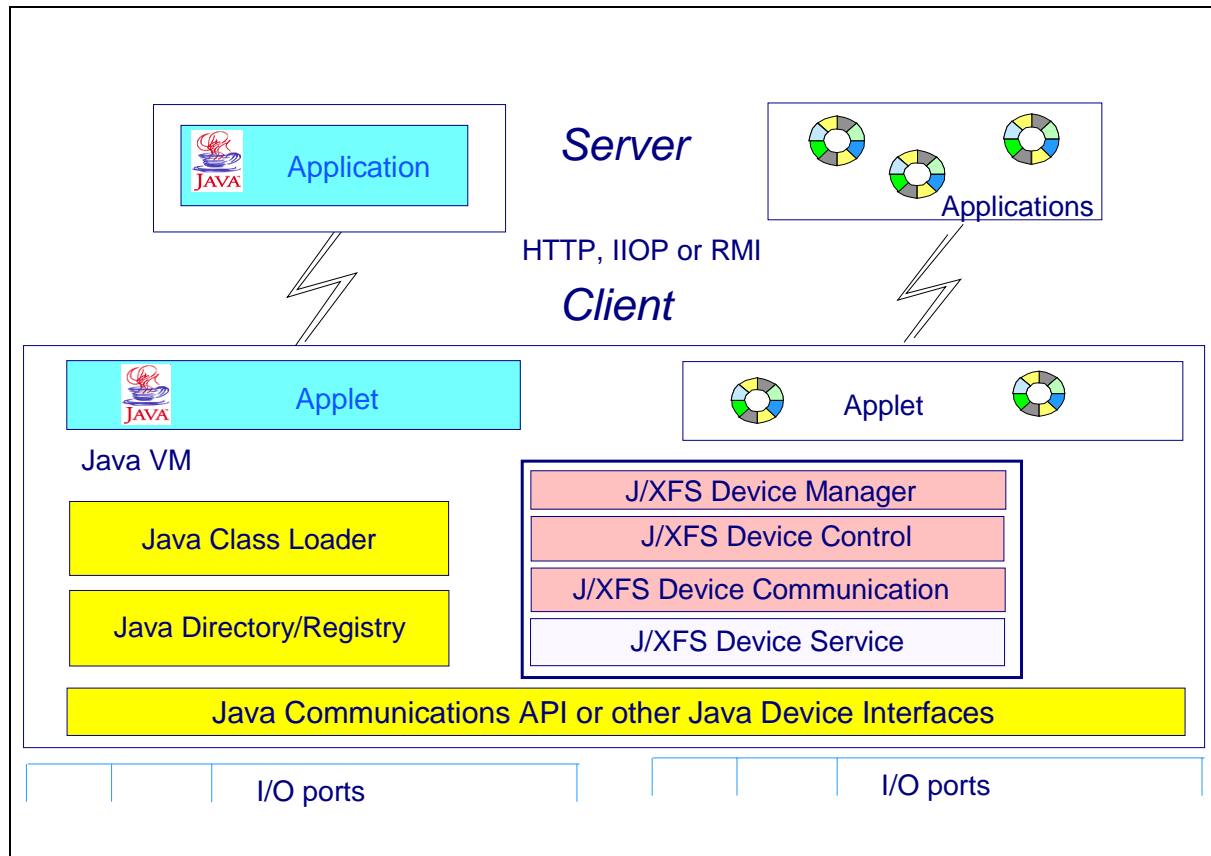


J/XFS defines the interface to a certain number of standard financial devices which are accessed by applets written in the Java programming language. The applets run in a Java VM environment (possibly embedded in a web browser) which allows multiple threads in the program but not multiple programs. Thus, the J/XFS layers must run in the same process context as the application. The J/XFS API's are provided for each Java banking application, regardless what the particular operating system platform actually is. By doing this, co-existence between older client/server and newly java-based banking applications can be ensured.

The J/XFS architecture for financial devices has the following defined layers:

- J/XFS Device Control and Device Manager
- J/XFS Device Communication
- J/XFS Device Service

The following picture gives an overview of this solution:



Both connections between the server and the client shown above will use TCP/IP. It might be possible to use HTTP as the protocol between the application running at the server and the applet running at the client. RMI can also be used. Using object oriented technology, the usage of IIOP between objects on the server and objects on the client is recommended to provide platform independence.

The J/XFS Kernel implementation provides the first three layers while the J/XFS compliant Device Service is provided by the manufacturer of the banking device (or any service provider capable to do this). The J/XFS Kernel implementation will use standard features or extensions of the actual Java Virtual Machine implementation, i.e. the Java Comm API and a class loader. Configuration data is stored in a directory or registry.

The Device Control API defines the way a Java application can communicate with a specific device. Additionally, the Device Control layer uses the central J/XFS Device Manager class which organizes access and location of the services. It is the central coordinating instance in any JVM which wants to access financial devices.

The Device Communication layer is the layer which resolves the sharing of devices. Its presence is transparent to the application, Device Control and Device Service.

The Device Service is the layer supplied by the device manufacturer for use with J/XFS. It has a defined API which allows the Device Control and Device Communication Layer to request actions from the device and translate them into the device specific commands which are then sent to the locally attached device (by using a Java Device Interface like the Java Comm API).

## J/XFS Device Manager

The J/XFS Device Manager is a central controlling instance where device requests are routed through. Within a given Java Virtual Machine there is exactly one Device Manager present.

Its main duties are:

- Keep lists of devices, services and communications
- Connect controls, comms and services in a transparent manner
- Query centrally configured data
- Keep controls and services from using non-standard Java parts
- Make controls and services more simple and straightforward to program

For each device, the J/XFS Device Manager keeps track of whether a service object already exists for the specified device, and returns this for each additional request. The J/XFS Device Manager shields all other J/XFS components from class loaders and configuration data storage.

## J/XFS Device Control

The J/XFS Device Control is the topmost layer in the device hierarchy and is responsible for supporting the financial application's use of the device interface. This interface specifies each financial device category in terms of a group of properties, methods and events. The J/XFS standard will map these entities in such a way as to match equivalent entities within the standard Java language component model. As a result, any Device Control may be packed as a Java Bean component. The Device Control layer isolates the financial application from the device service software, which is associated with the device rather than the application and which operates directly on the device.

### Properties

A control makes Device Service properties visible to the application, and notifies the service if the application changes them.

### Events

A control propagates Device Service generated events up to the application level.

### Methods

A control forwards all application service requests directly to the Device Service, and returns status or reposts exceptions to the application.

## J/XFS Device Communication

The optional Device Communication layer is a transparent layer between Device Control and Device Service. It provides a Device Service-like API to the top (i.e. to the Device Control) and a Device Control-like API to the bottom (i.e. to the local Device Service). Thus, it serves as an additional indirection layer to hide any network communication required between the Control and Service objects.

For the simpler case of local device access the Device Communication layer may be omitted and the J/XFS Device Manager, J/XFS Configuration and J/XFS Server are reduced to interfacing to a registry (JSD, a file or other available storages).

## J/XFS Device Service

This layer decouples the Device Control from the specifics of the hardware peripheral it must control. It implements the Device Service interface used by the controls, which in many respects mirror the device interface presented to the financial application. It is the Device Service which identifies the particular communication port over which the financial I/O device it services (e.g. magnetic stripe reader, passbook printer, etc.) is connected to the terminal. The Device Service then uses this port to implement the set of properties, methods and events defined by the J/XFS standard, via a series of escape sequences and data exchanges with the hardware peripheral.

Within the J/XFS architecture support for shared devices (e.g. teller assist units) will also be implemented.

## Java API requirements

J/XFS needs availability of at least a JDK 1.1.6 functionality. The additional security and internationalization functionality of JDK 2.0 is not reflected in the J/XFS specifications. These could be used by a future J/XFS implementation when JDK 2.0 and subsequent releases becomes widely available.

The J/XFS implementation in a pure Java environment the existence of the following Java APIs or equivalent layers being available on the various platforms:

### Java Communication API

This interface is an extension to Java (JDK 1.1.4 and above) and it allows the Device Service to identify those financial peripheral devices present on the physical platform which are connected to the communication ports. Instead of the javax.comm API J/XFS can use also other Java Device Interfaces as they become available.

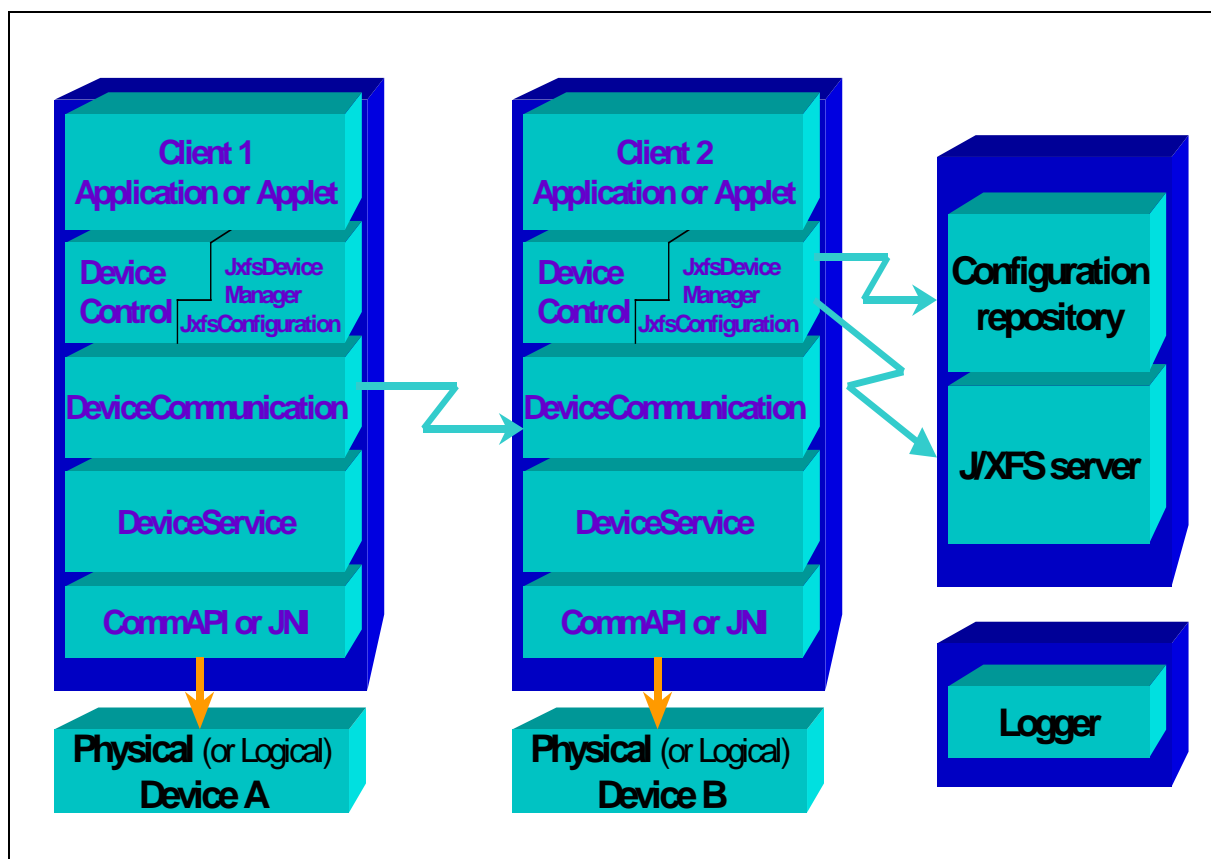
### Java Class Loader

The normal Java Class Loader 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. The Device Manager can use it to instantiate the Device Control.

To allow the sharing of devices on different machines, some efforts have to be done to ensure the localization and availability of these devices. This is the main reason why central administration processes and an additional indirection layer is also defined by the architecture. If only local access to devices is needed, an implementation may omit this communication layer.

Due to the nature of network computers which are supported as clients, it is not possible to guarantee that local persistent storage possibilities exist on each client although local caching storage may be present. Therefore, any configuration information may transparently be kept locally or stored on a central server.

The following diagram illustrates the basic architecture:



The development of a J/XFS Kernel implementation has started at the IBM Boeblingen Laboratory (Germany) in July 1999. The design and all reviews has been done jointly between all companies involved. The general availability of the J/XFS Kernel was March 2000.

The J/XFS Kernel fully implements the J/XFS standard as defined in the architecture and API specification documents. It Iso 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 today's bank branches as well as new hardware supporting the 'thin client' model.

J/XFS is targeted for all channels banks and other financial institutions are interested in.

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

The J/XFS Kernel deliverables provides 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

## Compliance

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.

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 JXFS API call, regardless of the specific operating system platform the JVM runs on.

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.

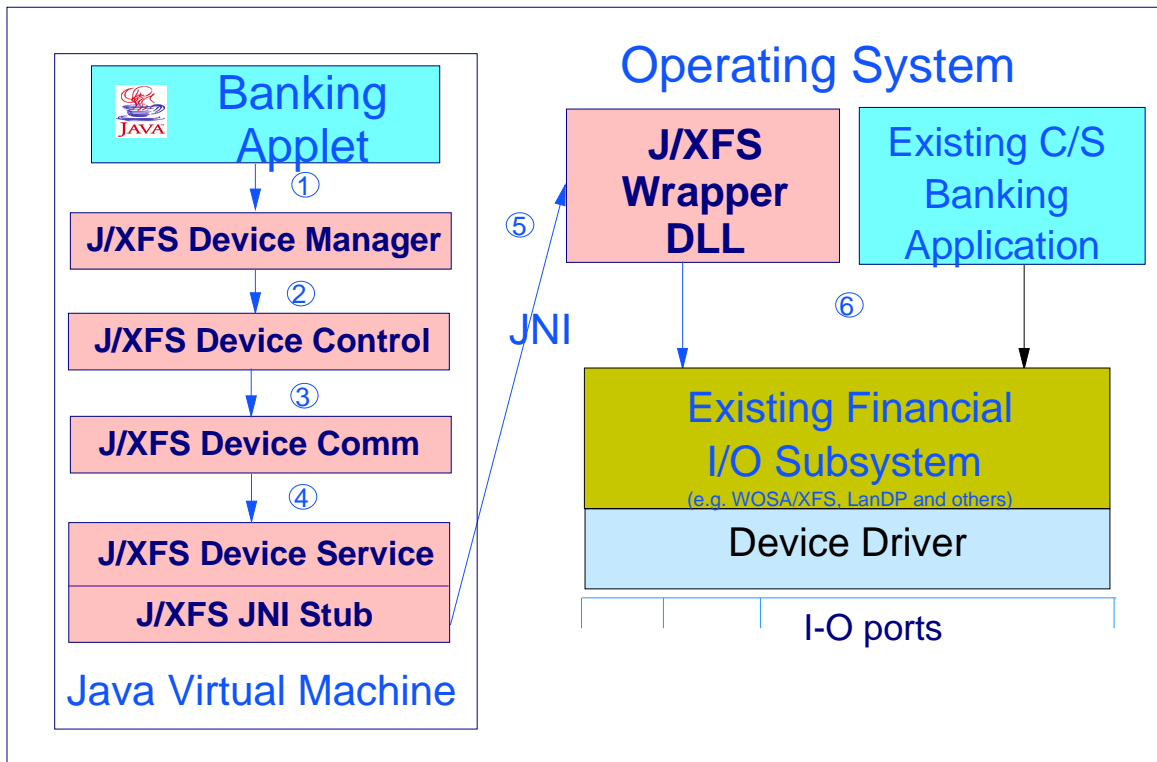
The above compliance definitions has been approved by the CEN/ISSS J/XFS Workshop in December 1999.

## Migration

Ensuring a migration path for currently installed banking device equipment as well as providing the co-existence between older banking applications and newly developed applications based on Java is done by providing J/XFS compliant wrappers. 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).

But the J/XFS architecture 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. These wrappers could be provided as an add-on to existing products by the vendors or can be developed within the scope of actual customer projects. 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.

A description of the concept of co-existence between existing client/Server banking applications and new Java banking applications is shown below:



- 1) A Java application or applet requests a Device Control from the J/XFS Device Manager.
- 2) Based on the configuration information which is available upon initial load, the J/XFS Device Manager loads the appropriate Device Control for the requested peripheral.
- 3) The J/XFS Device Control passes all requests to a corresponding J/XFS Device Service.
- 4) The J/XFS Device Communication layer will only be used if a connection to a remote client is needed.
- 5) The initialized J/XFS DeviceService is a shadow bean which behaves as a Device Service towards the J/XFS Kernel, and uses a JNI stub to pass requests through the Java Native Interface to a corresponding J/XFS Wrapper DLL, which runs under the control of the operating system.
- 6) This Wrapper DLL maps the J/XFS call to the appropriate I/O subsystem call, which is expected from the existing implementation running under the control and using the hardware device interface of the operating system.

We have decided to use two standard efforts as a starting point for the J/XFS standard, WOSA/XFS and JavaPOS. The first provides a starting point for the definitions of the requirements in the finance

industry needed by this standard while the second represents a starting point for how a set of similar requirements were met in another industry (i.e. the retail industry) under the Java environment.

## Relationship to WOSA/XFS

Several software vendors founded the Banking Solutions Vendor Council (BSVC) and agreed on the Windows™ Open Services Architecture/eXtensions for Financial Services (WOSA/XFS) which is currently available in a second version. WOSA/XFS defines a set of Application Programming Interfaces (APIs), Service Provider Interfaces (SPIs), and supporting services for providing access to financial services and unique finance industry peripheral devices. Such solutions are implemented in the finance industry, but they are following the client/server paradigm with all its problems in respect to distribution, deployment and maintenance.

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.

## Relationship to JavaPOS

The Java for Point of Sale (JavaPOS) standards committee was formed by a collection of retail vendors (e.g. SUN, NCR, SNI, Epson, and IBM), ISVs and end users to examine the ways in which advantages using Java could best be exploited. JavaPOS is a standard that defines an architecture for Java-based POS device access and a set of POS device interfaces (APIs) sufficient to support a range of POS solutions (for more information see <http://www.javapos.com> )

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.

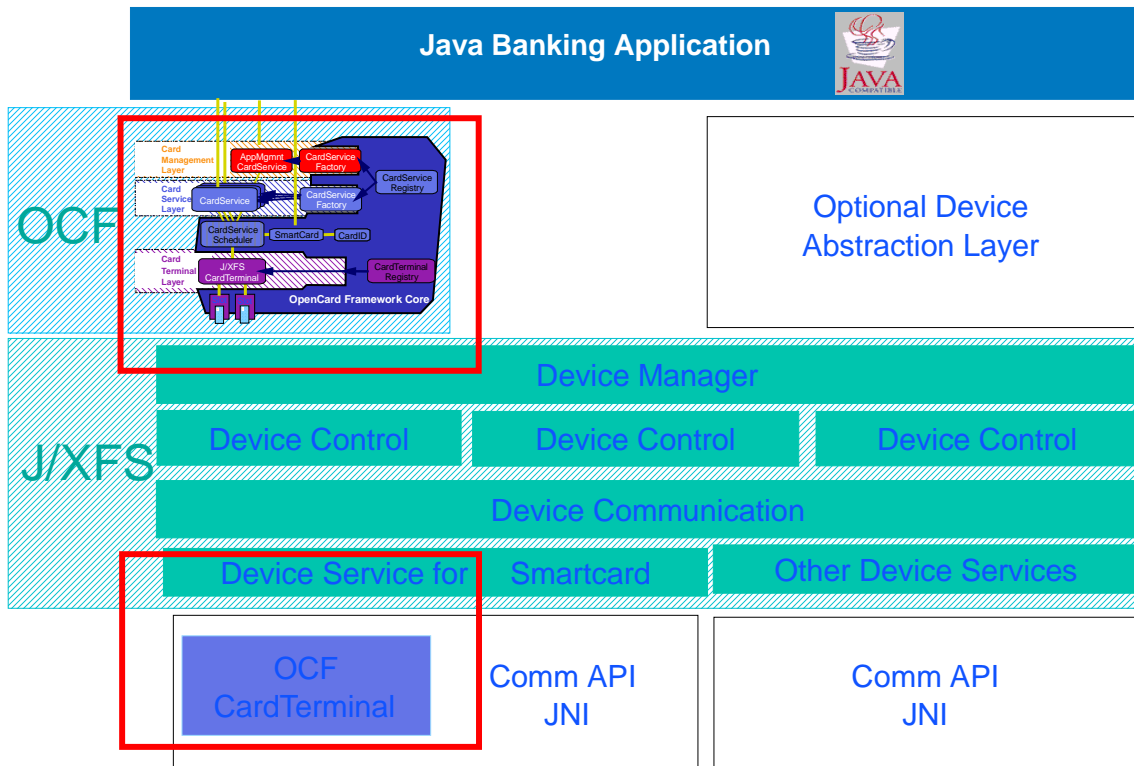
With the advent of Java as a platform-independent solution promising a real „thin"-client approach all advantages of Java could also be used for the finance industry to be able to avoid the high cost of deploying and maintaining client/server-based solutions.

### Relationship to Open Card 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 the J/XFS Device Manager and 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.



## Summary

The J/XFS Forum members provide financial institutions, Independent Software Vendors as well as banking hardware manufacturers this solid and extensible Application Programming Interface.

The J/XFS specifications cover the following device types:

- Printer Devices
  - Receipt Printer
  - Journal Printer
  - Passbook Printer
  - Document Printer
  - Scanner
- Cash Dispenser/Recycler and ATM
- Pin Pad Device
- ID Card
  - Chip Card Device
  - Magnetic Stripe Device
- Text I/O Device
- Alarm Device
- Depository Units
- Check Readers and Scanners
- Sensors and Indicators
- 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.

For more information visit the J/XFS website at <http://www.jxfs.com>.