

### How Java™ Technology Supports Image-Enabling with ViewWise™

#### Introduction

As the software market continues to evolve, it is increasingly more important to create software that is flexible, can be generated quickly, that supports industry standards as well as pushes the envelope and is a proven solution. Computhink selected Java as the ViewWise server solution because Java's language and architecture are the best fit using these criteria.

The Java platform on the server provides greater flexibility by allowing use in virtually any server environment, connection to most databases and client/administration connectivity from almost any location.

#### What is Java?

The Java platform is a fundamentally new way of computing, based on the power of networks and the idea that the same software should run on many different kinds of computers, consumer gadgets, and other devices.

With Java technology, you can use the same application from any kind of machine—a PC, a Macintosh computer, a network computer, or even new technologies like Internet screen phones.

The folks at Sun Microsystems have defined Java as the following: a simple, object-oriented, network-savvy, interpreted, robust, secure, architecture neutral, portable, high-performance, multithreaded, dynamic language.

#### Simple

A system that could be programmed easily without a lot of esoteric training and which leverages today's standard practice.

Most programmers working these days use C and most programmers doing object-oriented programming use C++. So even though C++ was unsuitable, Java was designed as closely to C++ as possible in order to make the system more comprehensible.

Java omits many rarely used, poorly understood, confusing features of C++ that can bring more grief than benefit. These omitted features primarily consist of operator overloading (although the Java language does have method overloading), multiple inheritance and extensive automatic coercions.

Automatic garbage collection was added, thereby simplifying the task of Java programming but making the system somewhat more complicated. A common source of complexity in many C and C++ applications is storage management - the allocation and freeing of memory. By virtue of having automatic garbage collection (periodic freeing of memory not being referenced) the Java language not only makes the programming task easier, it also dramatically cuts down on bugs.

*Developers can focus their energies on getting the job done rather than spending time trying to figure out where memory is being wasted in 20,000 lines of code.*

#### Object-Oriented



This is, unfortunately, one of the most overused buzzwords in the industry. But object-oriented design is very powerful because it facilitates the clean definition of interfaces and makes it possible to provide reusable "software ICs."

Simply stated, object-oriented design is a technique that focuses design on the data—or objects—and on the interfaces to those objects. To make an analogy with carpentry, an "object-oriented" carpenter would be mostly concerned with the chair he was building, and secondarily with the tools used to make it. A "non-object-oriented" carpenter would think primarily of his tools. Object-oriented design is also the mechanism for defining how modules "plug and play."

The object-oriented facilities of Java are essentially those of C++, with extensions from Objective C for more dynamic method resolution.

*Developers can essentially write a self-contained piece of code, add to it and re-use it many times in the development process.*

### **Network-Savvy**

Java has an extensive library of routines for coping easily with TCP/IP protocols like HTTP and FTP. This makes creating network connections much easier than in C or C++. Java applications can open and access objects across the net via URLs with the same ease that programmers are used to when accessing a local file system.

*Networking with other computers and systems either locally or world-wide is easy because Java uses the same protocols as the internet.*

### **Robust**

Java is intended for writing programs that must be reliable in a variety of ways. Java puts a lot of emphasis on early checking for possible problems, later dynamic (runtime) checking and eliminating situations that are error prone.

One of the advantages of a strongly typed language—like C++—is that it allows extensive compile-time checking so bugs can be found early. Unfortunately, C++ inherits a number of loopholes in compile-time checking from C, which is relatively lax (particularly method/procedure declarations). In Java, we require declarations and do not support C-style implicit declarations.

The linker understands the type system and repeats many of the type checks done by the compiler to guard against version mismatch problems.

The single biggest difference between Java and C/C++ is that Java has a pointer model that eliminates the possibility of overwriting memory and corrupting data. Instead of pointer arithmetic, Java has true arrays. This allows subscript checking to be performed. In addition, it is not possible to turn an arbitrary integer into a pointer by casting.

While Java doesn't make the QA problem go away, it does make it significantly easier.

Dynamic languages like Lisp, TCL and Smalltalk are often used for prototyping. One of the reasons for their success at this is that they are very robust. You don't have to worry about freeing or corrupting memory. Java programmers can be relatively fearless about dealing with memory because they don't have to worry about it getting corrupted.

Because there are no pointers in Java, programs can't accidentally overwrite the end of a memory buffer. Java programs also cannot gain unauthorized access to memory, which could happen in C or C++.

One reason that dynamic languages are good for prototyping is that they don't require you to pin down decisions too early. Java uses another approach to solve this dilemma. Java forces you to make choices explicitly because it has static typing, which the compiler enforces. Along with these choices comes a lot of assistance—you can write method invocations and if you get something wrong, you are informed about it at compile time. You don't have to worry about method invocation error.

*Java removes the inherent weakness of memory overwrites and out-of-sync versions of code by checking for potential problems and eliminating them.*

## **Secure**

Java is intended for use in networked/distributed environments. Toward that end, a lot of emphasis has been placed on security. Java enables the construction of virus-free, tamper-free systems. The authentication techniques are based on public-key encryption.

There is a strong interplay between "robust" and "secure." For example, the changes to the semantics of pointers make it impossible for applications to forge access to data structures or to access private data in objects that they do not have access to. This closes the door on most activities of viruses.

*Java is designed to make it very difficult for anyone to add to or tamper with restricted data.*

## **Architecture Neutral**

Java was designed to support applications on networks. In general, networks are composed of a variety of systems with a variety of CPU and operating system architectures. To enable a Java application to execute anywhere on the network, the compiler generates an architecture-neutral object file format—the compiled code is executable on many processors, given the presence of the Java runtime system.

This is useful not only for networks but also for single system software distribution. In the present personal computer market, application writers have to produce versions of their application that are compatible with the IBM PC and with the Apple Macintosh. With the PC market (through Windows/NT) diversifying into many CPU architectures, and Apple moving off the 680x0 toward the PowerPC, production of software that runs on all platforms becomes nearly impossible. With Java, the same version of the application runs on all platforms.

The Java compiler does this by generating byte-code instructions that have nothing to do with particular computer architecture. Rather, they are designed to be both easy to interpret on any machine and easily translated into native machine code on the fly.

*Java can run the same software on different platforms with little or no modification.*

## **Portable**

Being architecture neutral is a big chunk of being portable, but there's more to it than that. Unlike C and C++, there are no "implementation dependent" aspects of the specification. The sizes of the primitive data types are specified, as is the behavior of arithmetic on them. For example, "int" always means a signed two's complement 32 bit integer, and "float" always means a 32-bit IEEE 754 floating point number. Making these choices is feasible in this day and age because essentially all interesting CPUs share these characteristics.

The libraries that are a part of the system define portable interfaces. For example, there is an abstract Window class and implementations of it for Unix, Windows NT/95 and the Macintosh.

The Java system itself is quite portable. The compiler is written in Java and the runtime is written in ANSI C with a clean portability boundary. The portability boundary is essentially a POSIX subset.

*Java can run the same software on different platforms with little or no modification.*



## Interpreted

Java bytecodes are translated on the fly to native machine instructions (interpreted) and not stored anywhere. And since linking is a more incremental and lightweight process, the development process can be much more rapid and exploratory.

As a part of the bytecode stream, more compile-time information is carried over and available at runtime. This is what the linker's type checks are based on. It also makes programs more amenable to debugging.

*Java takes care of compile and link errors by creating code at run time instead of storing code for potential version mismatches and inconsistent procedures.*

## High Performance

While the performance of interpreted bytecodes is usually more than adequate, there are situations where higher performance is required. The bytecodes can be translated on the fly (at runtime) into machine code for the particular CPU the application is running on. For those accustomed to the normal design of a compiler and dynamic loader, this is somewhat like putting the final machine code generator in the dynamic loader.

The bytecode format was designed with generating machine codes in mind, so the actual process of generating machine code is generally simple. Efficient code is produced: the compiler does automatic register allocation and some optimization when it produces the bytecodes.

In interpreted code we're getting about 300,000 method calls per second on a Sun Microsystems SPARCStation 10. The performance of bytecodes converted to machine code is almost indistinguishable from native C or C++.

*Java runs at about the same speed as C.*

## Multithreaded

There are many things going on at the same time in the world around us. Multithreading is a way of building applications with multiple threads. Unfortunately, writing programs that deal with many things happening at once can be much more difficult than writing in the conventional single-threaded C and C++ style.

Java has a sophisticated set of synchronization primitives that are based on the widely used monitor and condition variable paradigm introduced by C.A.R. Hoare. By integrating these concepts into the language (rather than only in classes), they become much easier to use and are more robust. Much of the style of this integration came from Xerox's Cedar/Mesa system.

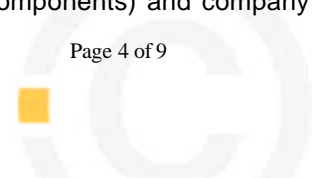
Other benefits of multithreading are better interactive responsiveness and real-time behavior. This is limited, however, by the underlying platform: stand-alone Java runtime environments have good real-time behavior. Running on top of other systems like Unix, Windows, the Macintosh, or Windows NT limits the real-time responsiveness to that of the underlying system.

*Java handles doing many things at once easily and efficiently.*

## Dynamic

In a number of ways, Java is a more dynamic language than C or C++. It was designed to adapt to an evolving environment.

For example, one major problem with C++ in a production environment is a side effect of the way that code is implemented. If company A produces a class library (a library of plug and play components) and company B



buys it and uses it in their product, then if A changes its library and distributes a new release, B will almost certainly have to recompile and redistribute their own software. In an environment where the end user gets A and B's software independently (say A is an OS vendor and B is an application vendor) problems can result. For example, if A distributes an upgrade to its libraries, then all of the software from B will break. It is possible to avoid this problem in C++, but it is extraordinarily difficult and it effectively means not using any of the language's OO features directly.

By making these interconnections between modules later, Java completely avoids these problems and makes the use of the object-oriented paradigm much more straightforward. Libraries can freely add new methods and instance variables without any effect on their clients.

An interface specifies a set of methods that an object can perform but leaves open how the object should implement those methods. A class implements an interface by implementing all the methods contained in the interface. In contrast, inheritance by sub classing passes both a set of methods and their implementations from super class to subclass. A Java class can implement multiple interfaces but can only inherit from a single super class. Interfaces promote flexibility and reusability in code by connecting objects in terms of what they can do rather than how they do it.

Classes have a runtime representation: there is a class named Class, instances of which contain runtime class definitions. If, in a C or C++ program, you have a pointer to an object but you don't know what type of object it is, there is no way to find out. However, in Java, finding out based on the runtime type information is straightforward. Because casts are checked at both compile-time and runtime, you can trust a cast in Java. On the other hand, in C and C++, the compiler just trusts that you're doing the right thing.

It is also possible to look up the definition of a class given a string containing its name. This means that you can compute a data type name and have it easily dynamically-linked into the running system.

*Making changes and populating them is much easier because Java only requires instructions on what an object can do rather than how it is to be done.*

## **What is ViewWise?**

ViewWise is an enterprise capable imaging system that allows users to capture, index, store, retrieve and search for images. It comes with a powerful set of APIs that can be customized to fit existing applications. This process is referred to as Image-enabling—a category ViewWise created through its innovative design.

An example of Image-enabling is the GroupWise imaging product that ships with Novell GroupWise 5.x packages. With GroupWise, users can use the interface they have become accustomed to and get the added benefits of imaging. This leverages their investment in the organization's existing hardware and software infrastructures, saving thousands of dollars.

ViewWise not only gives an organization industry standard imaging, but it also allows for numerous options like adding an array of plug-in technologies. MetaStorm's workflow product E-works is already image enabled by ViewWise. Some other possible efficiency-enhancing favorites include COLD, OCR/ICR, forms recognition, zonal extraction, and full text searching.

## **Java Server**

ViewWise utilizes a three-tier architecture with the Java server providing the bulk of the underlying connectivity. The Java server provides several distinct advantages over the typical client/server model.

**Pooled Connections** - In the typical client/server model each user that connects to a database application must have an individual connection to the database. The connection is reserved only for that user and is utilized only when the user is requesting/receiving data. Thus the connection is used sparingly. In the three-tier architecture model, users establish connection to the Java server and in turn the Java server establishes connection to the



database. The Java server efficiently manages the connections to the database and therefore can operate effectively on far fewer connections.

**Additional Security Layer** – One of the major concerns of the internet age is security. Hackers are constantly inventing new ways to break into systems. The three-tier model provides an additional layer of security. Clients never have direct access to the database. Instead, requests for data are submitted by the client to the Java server which then builds an SQL statement and passes it to the database. The client cannot circumvent the Java server and issue a direct SQL statement to the database.

*Through Java's additional security layer, users cannot get directly to the database. They must pass through the Java server.*

**Multiple Platform Support** – The Java server can execute on any platform that supports Java with little or no modification. The current tested and CompuThink certified supported platforms are NetWare 5.x, Windows NT 4.0, Windows 2000 Server.

*The Java server can support virtually any popular operating system.*

## RMI

Remote Method Invocation (RMI) first appeared in Java development kit 1.1. This allows Java applications to use remote method calls (rather than sockets and streams) to communicate across a network. Any Java object can be called remotely by creating a remote interface using the Java RMI API. Java RMI automatically generates the communications code needed to support distributed computing. Java RMI also generates a proxy object, which is used by the client to invoke methods on the server. The proxy object is standard Java bytecode, so it can run on any Java platform, and it can be transferred over the network.

This unique feature makes Java the optimal choice for distributed computing because it means that two applications don't need to agree on a particular interface until runtime. If one application wants to talk to another application, it can dynamically download the RMI proxy object and use it to make requests. If the second application changes at some point, you don't need to go back and change the first application. It can automatically download a new proxy object as needed to communicate with the revised application.

*Remote Method Invocation (RMI) is the glue that connects Java clients to Java servers.*

## JNDI

JNDI is an API specified in Java<sup>™</sup> that provides naming and directory functionality to applications written in Java. It is designed especially for Java by using Java's object model. Using JNDI, Java applications can store and retrieve named Java objects of any type. In addition, JNDI provides methods for performing standard directory operations, such as associating attributes with objects and searching for objects using their attributes.

JNDI is also defined independent of any specific naming or directory service implementation. It enables Java applications to access different, possibly multiple, naming and directory services using a common API. Different naming and directory service providers can be plugged in seamlessly behind this common API. This allows Java applications to take advantage of information in a variety of existing naming and directory services such as LDAP, NDS, DNS and NIS(Y.P). These providers allow Java applications to coexist with legacy applications and systems.

Using JNDI as a tool, the Java application developer can build new powerful and portable applications that not only take advantage of Java's object model but are also well-integrated with the environment in which they are deployed.



JNDI leverages your current investment in naming and directory services databases allowing Java to integrate with what you already have.

## JDBC

The ViewWise Java Server technically can support any relational database (RDBMS) that provides connectivity through Java Database Connectivity (JDBC). JDBC makes it possible to send SQL statements to a database and to process the results that are returned.

There are four general categories for JDBC drivers:

1. *The JDBC-ODBC bridge* provides JDBC API access via most ODBC drivers. Note that some ODBC binary code and in many cases database client code must be loaded on each client machine that uses this driver. This kind of driver is most appropriate on a corporate network, or for application server code written in the Java programming language in a 3-tier architecture.
2. *A native-API partly Java technology-enabled driver* converts JDBC calls into calls on the client API for Oracle, Sybase, Informix, DB2 or other DBMS. Note that, like the bridge driver, this style of driver requires that some binary code be loaded on each client machine.
3. *A net-protocol fully Java technology-enabled driver* translates JDBC API calls into a DBMS-independent net protocol which is then translated to a DBMS protocol by a server. This net server middleware is able to connect all of its Java technology-based clients to many different databases. The specific protocol used depends on the vendor. In general, this is the most flexible JDBC API alternative. It is likely that all vendors of this solution will provide products suitable for Intranet use. In order for these products to also support Internet access they must handle the additional requirements for security, access through firewalls, etc. that the web imposes.
4. *A native-protocol fully Java technology-enabled driver* converts JDBC technology calls into the network protocol used by DBMSs directly. This allows a direct call from the client machine to the DBMS server and is a practical solution for Intranet access. Since many of these protocols are proprietary the database vendors themselves will be the primary source for this style of driver.

ViewWise supported databases: Oracle 8.0 or greater, Microsoft SQL 7.0 and greater. Other databases can be supported as requested.

*JDBC is the Java protocol that connects Java to your current database.*

## RDBMS

In simplest terms, a relational database management system (RDBMS) is one that presents information in tables with rows and columns. Generally information in rows represents unique information about an object. Columns are set as categories for the object. When each row in a table is different, it is possible to use one or more columns to identify a particular row. This unique column or group of columns is called a primary key.

For example, to create information about five different people containing their first name, last name, and date of birth. The first name, last name and date of birth would be columns and each of the 5 people would be rows in the table. You would also need a unique number column or (primary key) to refer only to one person or row in the table in case you encountered two people with the same name and birth date.

Primary Key	FirstName	LastName	DateOfBirth
1	John	Smith	3/8/64



2	James	Davis	7/23/55
3	Mark	Peterson	4/6/42
4	Sara	Parker	11/10/70
5	Samantha	Jones	8/7/68

This is the fundamental design of all relational databases.

Information is retrieved from the databases using Structured Query Language (SQL). The ViewWise Java server builds the SQL statements and passes them to the RDBMS through JDBC connectivity.

*ViewWise utilizes any relational database (RDBMS) which is the current standard for industrial databases.*

## HTTP

The main protocol behind the World Wide Web (WWW) is the Hypertext Transfer Protocol (HTTP). It forms the basis for the transfer of documents between WWW servers and clients. It runs on top of the TCP/IP protocol suite. Some of the key ideas behind HTTP include persistent connections, ability to make multiplex multiple requests/responses over a single transport connection and the ability to pass URLs to a protocol in a response message.

*ViewWise utilizes HTTP, which is the current standard for internet communication.*

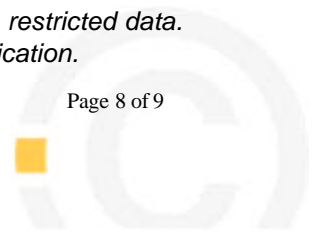
## NDS

Novell Directory Services (NDS), is a central database created by Novell that manages all resources for the network including the following: users, groups, organizations, security, network devices and applications.

*ViewWise integrates with NDS which is the corporate standard for large enterprise networks.*

## Summary of Java Benefits:

- *The Java platform on the server gives great flexibility by allowing use in virtually any server environment, connection to most databases and client/administration connectivity from almost any location.*
- *The Java platform is a fundamentally new way of computing, based on the power of networks and the idea that the same software should run on many different kinds of computers, consumer gadgets, and other devices.*
- *With Java technology, you can use the same application from any kind of machine—a PC, a Macintosh computer, a network computer, or even new technologies like Internet screen phones.*
- *Using Java, developers can focus their energies on getting the job done rather than spending time trying to figure out where memory is being wasted in 20,000 lines of code.*
- *Using Java, developers can essentially write a self-contained piece of code, add to it and re-use it many times in the development process.*
- *Networking with other computers and systems either locally or world-wide is easy because Java uses the same protocols as the internet.*
- *Java removes the inherent weakness of memory overwrites and out-of-sync versions of code by checking for potential problems and eliminating them.*
- *Java is designed to make it very difficult for anyone to add to or tamper with restricted data.*
- *Java can run the same software on different platforms with little or no modification.*



- *Java takes care of compile and link errors by creating code at run time instead of storing code for potential version mismatches and inconsistent procedures.*
- *Java runs at about the same speed as C.*
- *Java handles doing many things at once easily and efficiently.*
- *In the development process, making changes and populating them is much easier because Java only requires instructions on what an object can do rather than how it is to be done.*
- *Using pooled connections saves an end-user money because the organization doesn't need to buy a database connection for every user on the network.*
- *Through Java's additional security layer, users cannot get directly to the database. They must pass through the Java server.*
- *The Java server can support virtually any popular operating system.*
- *Remote Method Invocation (RMI) is the glue that connects Java clients to Java servers.*
- *JNDI leverages an end-user's current investment in naming and directory services databases allowing Java to integrate with what the organization already has.*
- *JDBC is the Java protocol that connects Java to the end-user's current database.*
- *ViewWise may utilize any relational database (RDBMS), which is the current standard for industrial databases.*
- *ViewWise utilizes HTTP, which is the current standard for internet communication.*
- *ViewWise integrates with NDS, which is the corporate standard for large enterprise networks.*

## **Conclusion**

The added benefits of Java make ViewWise the most state-of-the-art imaging system today. With innovative flexibility and scalability, image-enabling can save an organization hundreds of thousands of dollars in leveraged investments, saved training expenses, eliminated downtime and more.

The ViewWise Java server is an excellent fit for any organization. Whether you're large or just starting out, ViewWise leverages existing architectures and databases, is flexible and completely scalable to grow as you grow. ViewWise is also standards based, making it among the most robust and secure imaging systems on the market.

## **More information on ViewWise**

For additional information on ViewWise, please call 1-630 705 9050, or visit us on the world wide web at <http://www.computhink.com/>

