A Project Report On

Messaging System

Birla Institute of Technology
Mesra (Ranchi)

Extension Centre, Jaipur
2005

Submitted in partial fulfillment
of Final Semester Project Training

Submitted By:
Vijay Mahawar
(7MCA/5128/02)

Guided By:
Shri. Raghu Ramakrishnan
Smt. Rashmi Priyadarshi
Smt. Madhuri Kumari

Submitted to Birla Institute of Technology, Mesra, ...chi (Jaipur Campus)

Submitted to:
Birla Institute of Technology, Mesra (Ranchi)
Extension Centre, Jaipur

TO WHOMSOEVER IT MAY CONCERN

This is to certify that **Mr. VIJAY MAHAWAR**, Sixth Semester MCA Student of Birla Institute of Technology, Mesra (Ranchi), Extension Centre, Jaipur has undertaken project training under the guidance of Shri. RAGHU RAMAKRISHNAN at Tata Infotech Ltd, NSEZ, Noida from 1st Feb 2005 to 10th May 2005.

This project work is the genuine work of the candidate mentioned and has not been submitted before to the Institute.

Raghu R

Signature:

Date: 10/05/2005

TATA INFOTECH LIMITED NSEZ NOIDA

Company Seal

CERTIFICATE

This is to certify that the VIJAY MAHAWAR, student of BIT, Mesra (Ranchi), Extension Centre, Jaipur of M.C.A. VI Semester have successfully undergone his final semester Industrial Training at Tata Infotech Ltd., NSEZ, Noida.

Submitted to Birla Institute of Technology, Mesra, Ranchi (Jaipur Campus)

Sh. B. Pathak
(Faculty in-charge)

(Internal Examiner) **(External Examiner)**

ACKNOWLEDGEMENT

The following persons have provided a great deal of moral support and great help without which the project wouldn't have been possible.

1. Shri. Raghu Ramakrishnan
2. Smt. Rashmi Priyadarshi
3. Smt. Madhuri Kumari
4. Shri. Ramasamy Ramachandran

I extend my sincere thanks to them for their valuable support, cooperation and time.

I also take this opportunity to thank my friends and teammates at Tata Infotech who motivated me to learn the technologies, which were new to me.

Finally, I would like to convey my heartiest thanks to my family members for providing the necessary moral support, which was instrumental in the project development.

Submitted to Birla Institute of Technology, Mesra, Ranchi (Jaipur Campus)

Developed By **Signature**
Vijay Mahawar

About the Company

TATA INFOTECH LTD.

Established in 1977, Tata Infotech has been a leading name among the Indian IT companies. Over the last 27 years, industry experience and domain knowledge has enabled Tata Infotech to provide innovative solutions to their customers.

Tata Infotech is part of the 125-year-old Tata Group, India's most trusted and best-known industrial group. The group has a turnover of US $ 14.25 billion (equivalent to 2.6 % of India's GDP), covering 91 major companies with business operations in seven sectors - Engineering, Materials, Energy, Chemicals, Consumer Products, Services, and Communication and Information Systems.

Tata Infotech operates in the chosen areas of System Integration, Contract Hardware Manufacturing and IT Education. The Systems Integration services also include Software Development, IT Enabled Services and IT Infrastructure Management. The seven delivery centers across India and the proximity center in Chicago, US, have enabled customers across the world benefit from solutions, products and services of the company.

The company provides end-to-end application lifecycle management services using its proven methodologies based on SEI CMM Level 5 and ISO 9001 - 2000. Tata Infotech specializes in seamless distributed on-site and offshore service processes, which deliver significant savings to the customers.

Submitted to Birla Institute of Technology, Mesra, Ranchi (Jaipur Campus)

Systems Integration Services

The turnkey projects offered under the systems integration services cover a wide spectrum of solutions: ISP and Intranet setup, mission critical applications such as airline reservation systems, stock exchanges and telecom billing solutions, GIS solutions, communication software that handles more than 1000 nodes, e-governance solutions like driving license and vehicle registrations, 24x7 call centers/ helpdesk, ERP and e-biz solutions, healthcare, railroad operation & control, and complex WAN, MAN, and Campus LAN.

The company is a systems integration partner to leading technology companies like Sun, Cisco, HP-Compaq, BEA, Computer Associates, Business Objects, SAS, and MapInfo and has developed and implemented solutions covering smart card technology, imaging technology and medical equipment interface.

Software Development

Tata Infotech provides end-to-end application lifecycle management services using its proven methodologies based on SEI CMM Level 5 and ISO 9000 certified processes. Tata Infotech specializes in seamless distributed on-site and offshore service processes, which deliver significant savings to the customers. The company has developed expertise in handling projects of various kinds such as re-engineering, customized development, software integration etc. The company has been providing IT consultancy to various enterprises including Fortune 500 organizations. The enterprise has been involved in the entire value-chain of software development, right from requirement definition to post-production support, maintenance and training.

Tata Infotech is a pioneer in offshore software development. It has served clients across industries and is well suited to undertake offshore development projects.

IT Enabled Services

In the face of rapidly changing technology, and emerging markets, Tata Infotech provides state-of-the-art proven services and solutions to address the contact centre requirements of large corporations. With the clear intent of providing best of breed services to our clients, Tata Infotech has set up a joint venture with the world's leading call centre providers – SITEL. The joint venture addresses the needs of global clients through its facility with world-class infrastructure located in Mumbai, India.

The outsourcing solutions allow the customers to focus on their core businesses while Tata Infotech provides value-added customer contact solutions in support of the operational strategies. The partnership brings the advantages of credibility, client-centric approach, lifetime value of a customer, vast customer care and acquisition experience, ability to grow on a global scale, quick ramp up and implementation experience with proven operational quality and consistency.

Hardware Manufacturing Services

The manufacturing division offers its customers value-added contract manufacturing services (EMS) and hardware embedded design services for complex electro-mechanical products. Operating from a 100% export unit, current services include the manufacture and supply of automatic cheque processing systems, automatic teller machines and cash dispensers (ATM), SMT printed circuit board assemblies, computer hardware systems and sub-systems, embedded design solutions, mechatronic sub-assemblies and utility payment systems for customers worldwide.

Submitted to Birla Institute of Technology, Mesra Ranchi (Jaipur Campus)

Education Services

Education Services Division provides multi-level learning solutions across the entire spectrum of the enterprise through a customer driven approach. The learning solutions are a unique combination of learning and human resource development tools. The process comprises of skill gap identification, creating customized solutions and implementation. The entire integrated package of solutions is delivered within a pre-decided time frame, making the whole process cost effective with maximum returns on the training investments.

The offerings include solutions from SkillSoft, learning management solutions, competency management solutions, virtual classroom training, customized e-content development and blended learning solutions for enterprises.

Submitted to Birla Institute of Technology, Mesra, Ranchi (Jaipur Campus)

CONTENTS

S. No **TOPICS** **Page No.**

Submitted to Birla Institute of Technology, Mesra, Ranchi (Jaipur Campus)

Submitted to Birla Institute of Technology, Mesra, Ranchi (Jaipur Campus)

1. Introduction:

The aim of the project is to provide improved version of the Messaging System based on the Client-Server Technology with security features.

1.1. Scope:

Messaging System user interfaces shall be menu driven ergonomic desktop interfaces. The system shall provide for toolbars for all major actions. The system shall provide keyboard shortcuts for each action and an online help. The messages will be transmitted across the network via a server. The Servers will be built on cluster services with fail-safe functionality.

The Messaging System will comprise of Messaging Client and Administrator

Objective:

Reliability:

The system shall provide assured and secured messaging.

Availability:

The server part of the Messaging System shall be implemented through High Availability clustered Messaging Servers. In the event that one of the Messaging Servers shuts down or fails, the other Messaging Server shall detect the event and automatically take over without manual intervention at server end. The users must re-establish open sessions by re-authentication, and may need to initiate sending of messages interrupted by the failure.

User Authentication:

In order to prevent unauthorized use of the Messaging application, the system shall authenticate all identified Online Users, Node Administrator and Super Administrator of Messaging System using digital certificates.

User Access Rights:

The system shall provide role-based access to all Online Users, Node Administrators and Super Administrator.

Content Integrity and Non-repudiation:

The system shall ensure that messages travelling on the network are protected against alteration during transit using digital signatures. The system shall ensure non-repudiation of origin by using digital signatures.

Submitted to Birla Institute of Technology, Mesra, Ranchi (Jaipur Campus)

Content Confidentiality:

The system shall use PKI encryption to ensure content confidentiality. The local mailbox shall also be protected to ensure that there is no change to original message by unauthorised means. All traffic between client PC and Messaging Server/between two Messaging Servers shall be protected using IPSec.

Secured Storage of Private Key:

The system shall ensure secure storage of private keys in Private Key Tokens whose distribution among users shall be properly controlled. The policies regarding distribution of these keys and handling their loss and renewal is described in the

Message security labeling:

The system shall allow classification of message based on their sensitivity and handle the same as per the security policy of Messaging System. This shall be addressed by ensuring that all messages of Security Classification above Unclassified are encrypted using PKI. This option shall be based on configured policy.

Maintainability:

The system shall provide logs that shall trace the activities on client machine and server machine. These logs shall aid in easy problem identification and correction. The system shall provide command line utilities for easy server maintenance.

Portability:

The Messaging Client shall be developed in Java for portability.

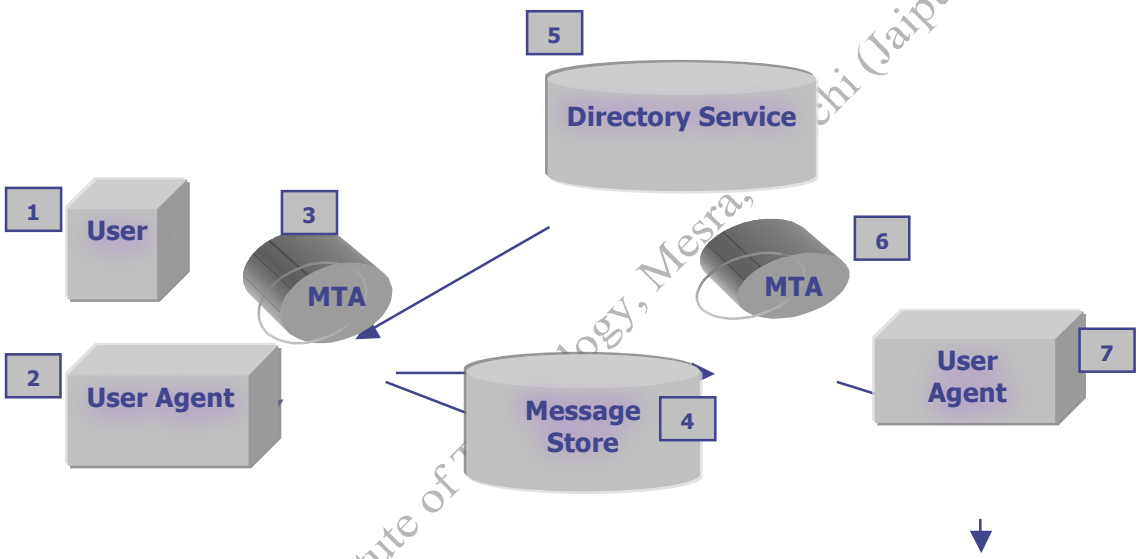
Submitted to Birla Institute of Technology, Mesra, Ranchi (Jaipur Campus)

1.2. What is Messaging?

Messaging is an exchanging of messages between an originator (message producer) and one or more recipients (message consumers).

Features Offered by Messaging System:

1. Authentication
2. Mailboxes
3. Mailing Lists
4. Read Receipts
5. Delivery Receipts
6. New Message Arrival Notification
7. Undelivered Message Notification



Routing and Delivery Model:

Message Creation

1. User creates a message using User Agent.
2. Message is put in an virtual envelope.
3. Addressee information is written on top of virtual envelope.

Message Dispatch

1. Envelope is sent to MTA in the user's message server.
2. MTA puts the timestamp when it receives virtual envelope.

Routing Decisions

1. MTA compares the Addressee information on top of the virtual envelope with Routing Directory.
2. Extract the recipient MTA details.

Message Transmission/Delivery

1. Envelope is sent to the recipient MTA.
2. Recipient MTA then stores it in the recipient's mailbox.
3. Recipient MTA puts the receipt timestamp on virtual envelope (Delivery receipt).

Message Retrieval

Submitted to Birla Institute of Technology, Mesra, Ranchi (Jaipur Campus)

1. User retrieves message from his/her mailbox, opens the virtual envelope and reads the message.
2. Read receipt is generated and sent to originator.

Messaging Standards:

RFC822

1. Allows for text only messages.
2. Allows for user defined headers.

Multipurpose Internet Mail Extension (MIME)

1. An extension of RFC822 that allows users to send binary files as attachments (graphics, photos, formatted text documents etc)
2. Allows for multiple attachments.
3. Uses an encoding mechanism like BASE64.

Simple Mail Transfer Protocol

1. Has a set of predefined commands.
2. Used to send messages from a UA to a MTA.
3. Used to exchange messages between two MTAs.

Post Office Protocol

1. Has a set of predefined commands.
2. Used to retrieve messages from a MS.

Submitted to Birla Institute of Technology, Mesra, Ranchi (Jaipur Campus)

1.3. Software Development Life Cycle:

1.3.1. Initiation Phase:

The initiation of a system (or project) begins when a business need or opportunity is identified. A Project Manager should be appointed to manage the project. This business need is documented in a Concept Proposal. After the Concept Proposal is approved, the System Concept Development Phase begins.

1.3.2. System Concept Development Phase:

Once a business need is approved, the approaches for accomplishing the concept are reviewed for feasibility and appropriateness. The Systems Boundary Document identifies the scope of the system and requires Senior Official approval and funding before beginning the Planning Phase.

1.3.3. Planning Phase:

The concept is further developed to describe how the business will operate once the approved system is implemented, and to assess how the system will impact employee and customer privacy. To ensure the products and /or services provide the required capability on time and within budget, project resources, activities, schedules, tools, and reviews are defined. Additionally, security certification and accreditation activities begin with the identification of system security requirements and the completion of a high-level vulnerability assessment.

1.3.4. Requirements Analysis Phase:

Functional user requirements are formally defined and delineate the requirements in terms of data, system performance, security, and maintainability requirements for the system. All requirements are defined to a level of detail sufficient for systems design to proceed. All requirements need to be measurable and testable and relate to the business need or opportunity identified in the Initiation Phase.

1.3.5. Design Phase:

The physical characteristics of the system are designed during this phase. The operating environment is established, major subsystems and their inputs and outputs are defined, and processes are allocated to resources. Everything requiring user input or approval must be documented and reviewed by the user. The physical characteristics of the system are specified and a detailed design is

Subject to Birla Institute of Technology, Mesra, Ranchi (Jaipur Campus)

prepared. Subsystems identified during design are used to create a detailed structure of the system. Each subsystem is partitioned into one or more design units or modules. Detailed logic specifications are prepared for each software module.

1.3.6. Development Phase:

The detailed specifications produced during the design phase are translated into hardware, communications, and executable software. Software shall be unit tested, integrated, and retested in a systematic manner. Hardware is assembled and tested.

1.3.7. Integration and Test Phase:

The various components of the system are integrated and systematically tested. The user tests the system to ensure that the functional requirements, as defined in the functional requirements document, are satisfied by the developed or modified system. Prior to installing and operating the system in a production environment, the system must undergo certification and accreditation activities.

1.3.8. Implementation Phase:

The system or system modifications are installed and made operational in a production environment. The phase is initiated after the system has been tested and accepted by the user. This phase continues until the system is operating in production in accordance with the defined user requirements.

1.3.9. Operations and Maintenance Phase:

The system operation is ongoing. The system is monitored for continued performance in accordance with user requirements, and needed system modifications are incorporated. The operational system is periodically assessed through In-Process Reviews to determine how the system can be made more efficient and effective. Operations continue as long as the system can be effectively adapted to respond to an organization's needs. When modifications or changes are identified as necessary, the system may reenter the planning phase.

1.3.10. Disposition Phase:

The disposition activities ensure the orderly termination of the system and preserve the vital information about the system so that some or all of the information may be reactivated in the future if necessary. Particular emphasis is given to proper preservation of

Submitted to Birla Institute of Technology, Mesra, Ranchi (Jaipur Campus)

the data processed by the system, so that the data is effectively migrated to another system or archived in accordance with applicable records management regulations and policies, for potential future access.

1.3.11. Documentation:

This life cycle methodology specifies which documentation shall be generated during each phase.
Some documentation remains unchanged throughout the systems life cycle while others evolve continuously during the life cycle. Other documents are revised to reflect the results of analyses performed in later phases. Each of the documents produced are collected and stored in a project file. Care should be taken, however, when processes are automated. Specifically, components are encouraged to incorporate a long-term retention and access policy for electronic processes. Be aware of legal concerns that implicate effectiveness of or impose restrictions on electronic data or records.

Submitted to Birla Institute of Technology, Mesra, Ranchi (Jaipur Campus)

2.OVERVIEW ON TECHNOLOGIES USED

Following technologies were used in the project:

☑ Java	Client
☑ C	Server
☑ PostgreSQL	Database

Java was chosen as appropriate choice for client due to its platform independence and security measures. Since the application requires a network system therefore Java was an obvious choice.

C language was chosen as appropriate choice for Server on Linux Machine due to its feature of secured low level network programming and efficient utilization of shared memory.

PostgreSQL was chosen as Database due to its compatibility with Linux and moreover being supplied by the Red Hat Package Manager in the installation Kit.

Submitted to Birla Institute of Technology, Mesra, Ranchi (Jaipur Campus)

2. Java: An Introduction

Java is a high-level, third-generation programming language, like C, Fortran, Smalltalk, Perl, and many others. It is a platform for distributed computing – a development and run-time environment that contains built-in support for the World Wide Web.

History of Java:

Java development began at Sun Microsystems in 1991, the same year the World Wide Web was conceived. Java's creator, James Gosling, did not design Java for the Internet. His objective was create a common development environment for consumer electronic devices which was easily portable from one device to another. The development team began by using C++, a high-level programming language, as their model but later altered it so as to have simplified syntax, increased robustness, better security features and greater portability across different operating system platforms. This effort evolved into a language, code named Oak and later named Java that retains much of the syntax and power of C++, but is simpler and more platform independent.

Java can be used to write computer applications that crunch numbers, process words, play games, store data or do any of the thousands of other things computer software can do. It is worth while to remember that though java is quite similar to C++. It is a different language with different characteristics.

1. Both programs are composed of one or more files with the "CLASS" extension and having machine independent Java Bytecode.
2. Both require JVM (Java Virtual Machine) to execute these Bytecodes.

Submitted to Birla Institute of Technology, Mesra, Ranchi (Jaipur Campus)

Java Platform

The evolution of the Java platform and supporting technologies has proceeded at quite a rapid pace. An indication to this quick maturation is that commercial applications have already been developed and are being deployed in Java. Some issues affecting this move to Java are:

- **Reduce Development Time and Cost**
  Java takes advantage of OOPS features. Furthermore, it has an edge over C++ for developing robust higher level software quickly.
- **Commercial class libraries**
  The Java language includes numerous built-in classes, which forms the foundation for all Java programs.
- **Component Based Applications**
  Java is considered to be the foundation to a new breed of productivity applications – such as word processors or spreadsheets – that are based on the object or component-programming model. In this model, applications can be designed as a collection of downloadable components rather than as a single shrink-wrapped software program. This in turn helps to face challenges in performance, usage monitoring and licensing.

  The problem with distributing executable programs through web pages is that computer programs are very closely tied to specific hardware and operating system they run on. A Windows programs will not run on a computer that only runs DOS. A Mac application can't run on a Unix workstation. VMS code can't be executed on an IBM mainframe, and so on. Therefore major commercial applications like Microsoft Word and Netscape Navigator have to be written almost separately for all the different platforms they run on. (A platform is loosely defined computer industry buzzword that typically means some combination of hardware and system software that will mostly run all the same software.) Netscape Navigator is one of the cross-platformest compatible of all major applications, yet it only runs on a small number of platforms. Java solves the problem of platform-independence by using byte code. The Java compiler does not produce native executable code for a particular machine like a C compiler would. Instead it produces a special format called byte code. Java byte code written in hexadecimal, byte by byte, looks like this:

Submitted to Birla Institute of Technology, Mesra, Ranchi (Jaipur Campus)

CA FE BA BE 00 03 00 2D 00 3E 08 00 3B 08 00 01 08 00 20 08

This looks a lot like machine language, but unlike machine language, Java byte code is exactly the same on every platform. This byte code fragment means the same thing on a Solaris workstation and a Macintosh PowerBook. Java programs that have been compiled into bytecode, still need an interpreter to execute them on any given platform. The interpreter reads the byte code and translates it into the native language of the host machine on the fly. The most common example of such an interpreter is Sun's program java (with a small j). Since the bytecode is completely platform independent, only the interpreter and a few native libraries need to be ported to get Java to run on a new computer or operating system. The rest of the runtime environment, including the compiler and most of the class libraries, is written in Java.

All these pieces – the Javac compiler, the Java interpreter, the Java programming language, and more are collectively referred to as Java.

Submitted to Birla Institute of Technology, Mesra, Ranchi (Jaipur Campus)

2.1. Features of the Java Language:

Having seen that Java is equally suited as a language for development both on and off the Internet, it's time to look more closely at the Java language itself. The creators of Java at Sun Microsystems have defined the Java language as "a simple, object-oriented, distributed, interpreted, robust, secure, architecture-neutral, portable, high-performance, multithreaded, and dynamic language." Well, they managed to fit all of the important 1990s buzzwords into one sentence, but we need to look more closely at Java to see if they managed to fit all of these concepts into one language.

1) SIMPLE

If you have experience with any object-oriented language, especially C++, you probably will find Java to be easier than your high school prom date. Because Java started out as C++ but has had certain features removed, it is certainly a simpler language than C++.

The simplicity of Java is enhanced by its similarities to C and C++. Because many of today's current programmers, especially those likely to consider using Java, are

experienced in at least C and probably C++, Java is instantly familiar to these programmers.

Java has simplified C++ programming by both adding features beyond those found in C++ and by removing some of the features that make C++ a complicated and difficult language to master. Java is simple because it consists of only three primitive data types-numbers, Boolean types, and arrays. Everything else in Java is a class. For example, strings are true objects, not just arrays of characters. Similarly, arrays in the Java language are first-class objects, not just memory allocations and runtime representations.

Java offers additional simplifications over C++. The ubiquitous goto statement has been removed. Operator overloading, a frequent source of confusion during the maintenance of C++ programs, is not allowed in Java. Unlike C and C++, Java has no preprocessor. This means that the concepts behind #define and typedef are not necessary in Java. Java reduces the redundancy of C++ by removing structures and unions from the language. These are both just poor cousins of a full-fledged class and are superfluous in a cohesively designed language. Of course, they were necessary in C++ because it was important for early C++ translators and then compilers to be able to correctly interpret the existing C code that relied on these features.

The most important C++ feature left out of Java is the capability to directly manipulate memory addresses through the use of pointers. Pointers are one of the cornerstones of the C and C++ languages, and it would be difficult to write many programs in these languages without using pointers. However, as any C or C++ programmer will admit, pointers are also a significant source of problems and debugging time in C and C++ programs. Pointers can accidentally be set to point to the wrong thing, causing unexpected behavior including crashes. Pointers also can be used to store allocated memory. If the allocated memory isn't freed, or released back to the operating system, then the program will gradually leak memory, until it eventually runs out. An entire set of commercial products, such as the Bounds Checker products, has come into existence to help programmers identify these types of pointer-related problems. Java simplifies this by completely removing pointers from the language and using a handle-based solution instead.

Submitted to Birla Institute of Technology, Mesra, Ranchi (Jaipur Campus)

Of course, if all Java did was remove syntax from C++, it would be a poor compiler instead of an exciting new language. Java goes well beyond C++ by adding some important features. One of the most important is automatic memory management, usually known as *garbage collection*. Garbage collection is really just a blue-collar term that means that you don't need to free memory that you allocate-the Java Virtual Machine takes care of doing this for you. If you're a C or C++ programmer, or have ever had to track down memory leaks in another language, just imagine how nice your life could be if you never have to do it again. You would have time for walks on the beach, barbecued turkey burgers on holiday weekends, and romantic evenings with your spouse.

Java goes beyond C++ in a variety of other ways, as well. For example, Java includes language-level support for writing *multithreaded* programs. A multithreaded program is one that is written such that it performs more than one task at a time. For example, consider the stock price Web page shown earlier in Figure 1.2. One thread in the program to create this page may be constantly retrieving quotes from the stock exchange while another thread searches various news databases for breaking stories about the stocks being monitored. Although you can definitely write this program in a traditional single-threaded manner, the ability to use multiple threads can make it simpler to write and maintain.

Submitted to Birla Institute of Technology, Mesra, Ranchi (Jaipur Campus)

2) OBJECT-ORIENTED

Of course, Java is object-oriented. In fact, in the mid-1990s, it's hard to imagine someone developing a new language and declaring it the greatest new thing without it being object-oriented. In its approach to object-orientation, Java follows more closely along the lines of languages such as SmallTalk than C++. Except for its primitive data types, everything in Java is an object. In contrast, C++ is much more lax in that you are entirely free to mix and match object-oriented code (classes) and procedural code (functions). In Java, this is not the case. There are no global functions in Java: all functions are invoked through an object.

Java's support for object-orientation does not include multiple inheritance. The designers of the language felt that the complexity introduced by multiple inheritance was not justified by its benefits.

Java classes are comprised of methods and variables. *Class methods* are the functions that an object of the class can respond to. *Class variables* are the data that define the state of an object. In Java, methods and variables can be declared as *private*, *protected*, or *public*. Private methods and variables are not accessible outside of the class. Protected members are accessible to subclasses of the class, but not to other classes. Finally, public methods and variables are accessible to any class.

Classes in Java can be defined as abstract. An abstract class is a class that collects generic state and behavioral information. More specific classes are defined as subclasses of the abstract class and are used to define actual, specific entities. For example, software in use at a pet store may have an abstract class named Pet. This class would store information that is common to all pets-birthdate, cost, sale price, date received, and so on. Derived from the abstract Pet class could be classes such as Dog, Cat, Bird, and Fish. Each of these classes can augment the abstract class as necessary. For example, a member variable called WaterType (salt or fresh) would be necessary in Fish. Because WaterType would be meaningless for Dogs, Cats, and Birds, it is not part of the abstract implementation of Pet.

Submitted to Birla Institute of Technology, Mesra, Ranchi (Jaipur Campus)

3) DISTRIBUTED

Java facilitates the building of distributed applications by a collection of classes for use in networked applications. By using Java's URL (Uniform Resource Locator) class, an application can easily access a remote server. Classes also are provided for establishing socket-level connections.

4) INTERPRETED

Because Java is interpreted, once the Java interpreter has been ported to a specific machine, that machine can instantly run the growing body of Java applications. As an example of the usefulness of this, imagine a hypothetical chip manufacturer, Outtel, that has just finished its newest CPU chip. This new chip, named the Zentium, serves as the foundation of a new line of computers being marketed toward Zen Buddhist monasteries. Once Outtel ports the Java interpreter to work on the Zentium, the new machine will be able to run all of the Java development utilities-the compiler, the debugger, and so on. Contrast this with a traditional language. If Outtel wants to release a C++

compiler with its new computer it must port, or create from scratch, the compiler, the debugger, the runtime library, and so on.
Also, when using an interpreter, programmers are freed from some of the concerns of intermodule dependencies. You no longer have to maintain a "make" file that is sometimes as complicated as the hardest part of your program.
Another advantage is that the time-consuming edit-compile-link-test cycle is broken. Without the compile and link steps, working in an interpreted environment is a much simpler edit-test cycle. Even with today's quick C++ compilers, it is not uncommon for a complete recompile and relink of a large program to be measured in hours and take the better part of a day. Without having to wait for lengthy compiles and links, Java promotes prototyping and easier debugging.

5) ROBUST

The designers of Java anticipated that it would be used to solve some very complex programming problems. Writing a distributed, multithreaded program that can run on a variety of operating systems with a variety of processors is not a simple task. To do it successfully, you need all the help your programming language can offer you. With this in mind, Java was created as a strongly typed language. Data type issues and problems are resolved at compile-time, and implicit casts of a variable from one type to another are not allowed.
Memory management has been simplified in Java in two ways. First, Java does not support direct pointer manipulation or arithmetic. This makes it impossible for a Java program to overwrite memory or corrupt data. Second, Java uses runtime garbage collection instead of explicit freeing of memory. In languages like C++, it is necessary to delete or free memory once the program has finished with it. Java follows the lead of languages such as LISP and SmallTalk by providing automatic support for freeing memory that has been allocated but is no longer used.

6) SECURE

Closely related to Java's robustness is its focus on security. Because Java does not use pointers to directly reference memory locations, as is prevalent in C and C++, Java has a great deal of control over the code that exists within the Java environment.

Submitted to Birla Institute of Technology, Mesra, Ranchi (Jaipur Campus)

It was anticipated that Java applications would run on the Internet and that they could dynamically incorporate or execute code found at remote locations on the Internet. Because of this, the developers of Java hypothesized the existence of a hostile Java compiler that would generate Java byte codes with the intent of bypassing Java's runtime security. This led to the concept of a byte-code verifier. The byte-code verifier examines all incoming code to ensure that the code plays by the rules and is safe to execute. In addition to other properties, the byte code verifier ensures the following:
No pointers are forged.
No illegal object casts are performed.
There will be no operand stack overflows or underflows.
All parameters passed to functions are of the proper types.
Rules regarding private, protected, and public class membership are followed.

7) ARCHITECTURE-NEUTRAL

Back in the dark ages of the early 1980s, there was tremendous variety in desktop personal computers. You could buy computers from Apple, Commodore, Radio Shack, Atari, and eventually even from IBM. Additionally, every machine came with its own very different operating system. Because developing software is such a time-consuming task, very little of the software developed for use on one machine was ever ported and then released for use on a different machine.
In many regards, this situation has improved with the acceptance of Windows, the Apple Macintosh, and UNIX variations as the only valid personal computer options. However, it is still not easy to write an application that can be used on Windows NT, UNIX, and a Macintosh. And it's getting more complicated with the move of Windows NT to non-Intel CPU architectures.
A number of commercially available source code libraries (for example, Zinc, ZApp, and XVT) attempt to achieve application portability. These libraries attempt this by focusing on either a lowest common denominator among the operating systems or by creating a common core API (Application Programming Interface).
Java takes a different approach. Because the Java compiler creates byte code instructions that are subsequently interpreted by the Java interpreter, architecture neutrality is achieved in the implementation of the Java interpreter for each new architecture.

Submitted to Birla Institute of Technology, Mesra, Ranchi (Jaipur Campus)

8) PORTABLE

In addition to being architecture-neutral, Java code is also portable. It was an important design goal of Java that it be portable so that as new architectures (due to hardware, operating system, or both) are developed, the Java environment could be ported to them.
In Java, all primitive types (integers, longs, floats, doubles, and so on) are of defined sizes, regardless of the machine or operating system on which the program is run. This is in direct contrast to languages like C and C++ that leave the sizes of primitive types up to the compiler and developer.
Additionally, Java is portable because the compiler itself is written in Java and the runtime environment is written in POSIX-compliant C.

9) HIGH-PERFORMANCE

For all but the simplest or most infrequently used applications, performance is always a consideration. It is no surprise, then, to discover that achieving high performance was one of the initial design goals of the Java developers. A Java application will not achieve the performance of a fully compiled language such as C or C++. However, for most applications, including graphics-intensive ones such as are commonly found on the World Wide Web, the performance of Java is more than adequate. For some applications, there may be no discernible difference in performance between C++ and Java.
Many of the early adopters of C++ were concerned about the possibility of performance degradation as they converted their programs from C to C++. However, many C++ early adopters discovered that, although a C program will outperform a C++ program in many cases, the additional development time and effort don't justify the minimal performance gains. Of course, because we're not all programming in assembly language, there must be some amount of performance we're willing to trade for faster development.
It is very likely that early experiences with Java will follow these same lines. Although a Java application may not be able to keep up with a C++ application, it will normally be fast enough, and Java may enable you to do things you couldn't do with C++.

10) MULTITHREADED

Writing a computer program that only does a single thing at a time is an artificial constraint that we've lived with in

Submitted to Rajasthan Institute of Technology, Veera Panchi (Jaipur Campus)

most programming languages. With Java, we no longer have to live with this limitation. Support for multiple, synchronized threads is built directly into the Java language and runtime environment.
Synchronized threads are extremely useful in creating distributed, network-aware applications. Such an application may be communicating with a remote server in one thread while interacting with a user in a different thread.

11) DYNAMIC

Because it is interpreted, Java is an extremely dynamic language. At runtime, the Java environment can extend itself by linking in classes that may be located on remote servers on a network (for example, the Internet). This is a tremendous advantage over a language like C++ that links classes in prior to runtime.
In C++, every time member variables or functions are added to a class, it is necessary to recompile that class and then all additional code that references that class. Of course, the problem is exacerbated by the fact that you need to remember to recompile the files that reference the changed class. Using make files reduces the problem, but for large, complex systems, it doesn't eliminate it.
Java addresses this problem by deferring it to runtime. At runtime, the Java interpreter performs name resolution while linking in the necessary classes. The Java interpreter is also responsible for determining the placement of objects in memory. These two features of the Java interpreter solve the problem of changing the definition of a class used by other classes. Because name lookup and resolution are performed only the first time a name is encountered, only minimal performance overhead is added.

Submitted to Birla Institute of Technology, Mesra, Ranchi (Jaipur Campus)

2.2. Client-Server architecture:

An Introduction to Sockets

The computers on the Internet are connected by the TCP/IP protocol. In the 1980s, the Advanced Research Projects Agency (ARPA) of the U.S. government funded the University of California at Berkeley to provide a UNIX implementation of the TCP/IP protocol suite. What was developed was termed the *socket interface*, although you might hear it called the Berkeley-socket interface or just Berkeley sockets. Today, the socket interface is the most widely used method for accessing a TCP/IP network.

A socket is nothing more than a convenient abstraction. It represents a connection point into a TCP/IP network, much like the electrical sockets in your home provide a connection point for your appliances. When two computers want to converse, they each use a socket. One computer is termed the server-it opens a socket and listens for connections. The other computer is termed the client; it calls the server socket to start the connection. To establish a connection, all that's needed is a destination address and a port number.

Each computer in a TCP/IP network has a unique address. *Ports* represent individual connections within that address. This is analogous to corporate mail-each person within a company shares the same address, but a letter is routed within the company by the person's name. Each port within a computer shares the same address, but data is routed within each computer by the port number. When a socket is created, it must be associated with a specific port-this is known as binding to a port.

Submitted to Lnct Institute of Technology Mesra, Ranchi (Jaipur Campus)

Socket Transmission Modes

Sockets have two major modes of operation: *connection-oriented* and *connectionless*. Connection-oriented sockets operate like a telephone; they must establish a connection and a hang up. Everything that flows between these two events arrives in the same order it was sent. Connectionless sockets operate like the mail-delivery is not guaranteed, and multiple pieces of mail may arrive in a different order than they were sent.

Which mode to use is determined by an application's needs? If reliability is important, then connection-oriented operation is better. File servers need to have all their data arrive correctly and in sequence. If some data were lost, the server's usefulness would be invalidated. Some applications-a timeserver, for example-send discrete chunks of data at regular intervals. If the data became lost, the server would not want the network to retry until the data was sent. By the time the data arrived, it would be too old to have any accuracy. When you need reliability, be aware that it does come with a price. Ensuring data sequence and correctness requires extra processing and memory usage; this extra overhead can slow down the response times of a server.
Connectionless operation uses the User Datagram Protocol (UDP). A datagram is a self- contained unit that has all the information needed to attempt its delivery. Think of it as an envelope-it has a destination and return address on the outside and contains the data to be sent on the inside. A socket in this mode does not need to connect to a destination socket; it simply sends the datagram. The UDP protocol promises only to make a best-effort delivery attempt. Connectionless operation is fast and efficient, but not guaranteed.

Connection-oriented operation uses the Transport Control Protocol (TCP). A socket in this mode needs to connect to the destination before sending data. Once connected, the sockets are accessed using a streams interface: open-read-write-close. The other end of the connection in exactly the same order it was sent receives everything sent by one socket. Connection-oriented operation is less efficient than connectionless, but it's guaranteed.

Sun Microsystems has always been a proponent of internetworking, so it isn't surprising to find rich support for sockets in the Java class hierarchy. In fact, the Java classes have significantly reduced the skill needed to create a sockets program. Each transmission mode is implemented in a separate set of Java classes. The connection-oriented classes will be discussed first.

Submitted to Birla Institute of Technology, Mesra, Ranchi (Jaipur Campus)

Introduction to TCP Sockets

Socket is an abstraction used as communication end-point. Sockets are the combinations of an IP address and the port number appended to it. The socket is that abstract end point used for communication. The socket pair for a TCP connection is the 4-tuple that defines the two endpoints of the connections:

- Local IP address
- Local TCP port
- Foreign IP address
- Foreign TCP port

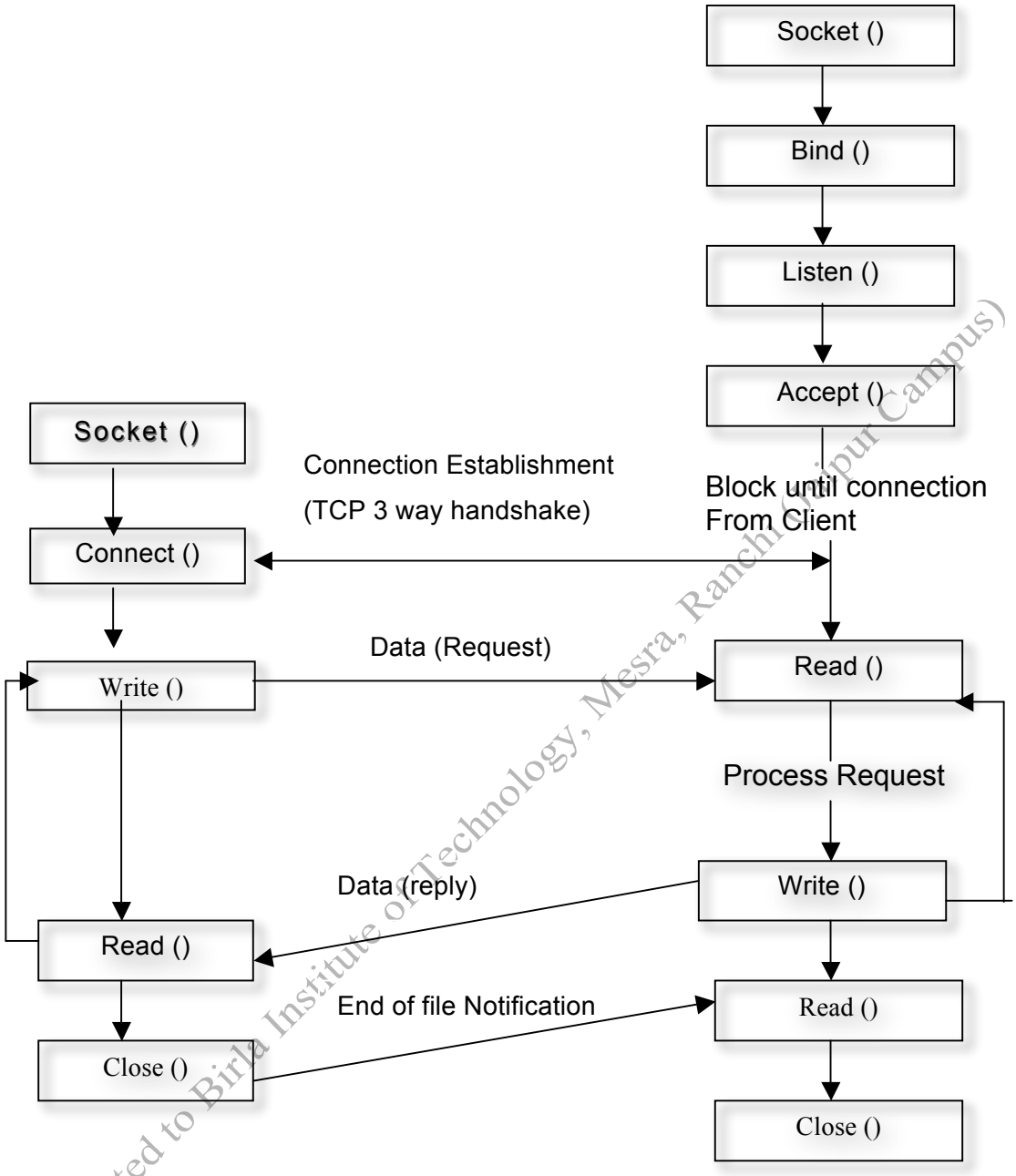
A socket pair uniquely identifies every TCP connection on an Internet using the above four attributes. The following list provides the salient features of TCP/IP mechanism transport (refer to figure 1). TCP provides connection between clients and servers. A TCP client establishes a connection with a server exchanges data with that server across the connection and then terminates the connection.

The first and foremost distinguishing feature of a TCP protocol is reliability. When TCP sends data to other end, it requires an acknowledgement in return. If an acknowledgement is not received it automatically retransmits the information and waits a longer period of time. TCP also sequences the data by associating a sequence number with every byte that it sends. If the segment arrives out of order, the receiving TCP will reorder the two segments based on their sequence numbers before passing the data to the receiving application.

TCP also provides flow control. TCP always tell its peer exactly how many bytes of data it is willing to accept from the peer. This is called the advertised window. At any time, the window is the amount of room currently available in the receiver buffer, guaranteeing that the sender cannot overflow the receiver's buffer. The window also changes dynamically over time.

Finally, TCP connection is full duplex. This means that an application can send and receive data in both directions on a given connection at any time. This also means that a TCP also keeps track of state information such as sequence numbers and window sizes for each direction of data flow: sending and receiving.

Submitted to Birla Institute of Technology, Mesra, Ranchi (Jaipur Campus)



Submitted to Birla Institute of Technology, Mesra, Ranchi (Jaipur Campus)

Figure 0-2: Socket function for TCP Client and Server

TCP/IP Client Sockets:

There are two kinds of TCP Sockets:

a. Server Socket (Designed to be a listener, which wait for client before doing anything)
b. Socket (Designed to connect to the server socket and initiate protocol exchange)

Two constructors are used to create client socket:

a. Socket (String hostName, int port):
   It creates a socket connecting the local host. The port can throw UnknownHostException or an IOException

b. Socket(Inetaddress ipaddress, int port) throws IOException

Methods:

I InetAddress getInetAddress()
Returns the InetAddress associated with the socket object.

II int getport()
Returns the remote port to which this socket object is connected.

III int getLocalport()
Returns the local port.

Submitted to Birla Institute of Technology, Mesra, Ranchi (Jaipur Campus)

2.2. Parsing XML Files In Java:

XML is becoming increasingly popular in the developer community as a tool for passing, manipulating, storing, and organizing information. If you are one of the many developers planning to use XML, you must carefully select and master the XML parser. The parser—one of XML's core technologies—is your interface to an XML document, exposing its contents through a well-specified API. Confirm that the parser you select has the functionality and performance that your application requires. A poor choice can result in excessive hardware requirements, poor system performance and developer productivity, and stability issues.

I tested a of selection of Java-based XML parsers, and this article presents the results while discussing the performance issues you should consider when selecting a parser. Your software's performance hinges on your choosing the right one.

Performance Issues

Because XML is a standardized format, it offers more developer and product support than proprietary formats, parsers, and configuration and storage schemes. Your XML project also will be easier to manage if you keep it simple. If possible, write interface code in only one or two languages (e.g., Java or C++), using as few APIs as possible (DOM, SAX, XML, and perhaps JAXP).

Minimizing technologies sounds good in theory, but it can't be done without effective tools. What makes a tool effective? Depending on the project, the following attributes can:

- Stable specifications
- Commercial vendor support
- Adequate performance
- Adequate API features

Submitted to Birla Institute of Technology, Mesra, Ranchi (Jaipur Campus)

SAX vs. DOM

At present, two major API specifications define how XML parsers work: SAX and DOM. The DOM specification defines a tree-based approach to navigating an XML document. In other words, a DOM parser processes XML data and creates an object-oriented hierarchical representation of the document that you can navigate at run-time.

The SAX specification defines an event-based approach whereby parsers scan through XML data, calling handler functions whenever certain parts of the document (e.g., text nodes or processing instructions) are found.
How do the tree-based and event-based APIs differ? The tree-based W3C DOM parser creates an internal tree based on the hierarchical structure of the XML data. You can navigate and manipulate this tree from your software, and it stays in memory until you release it. DOM uses functions that return parent and child nodes, giving you full access to the XML data and providing the ability to interrogate and manipulate these nodes. DOM manipulation is straightforward and the API does not take long to understand, particularly if you have some JavaScript DOM experience.

In SAX's event-based system, the parser doesn't create any internal representation of the document. Instead, the parser calls handler functions when certain events (defined by the SAX specification) take place. These events include the start and end of the document, finding a text node, finding child elements, and hitting a malformed element.

SAX development is more challenging, because the API requires development of callback functions that handle the events. The design itself also can sometimes be less intuitive and modular. Using a SAX parser may require you to store information in your own internal document representation if you need to rescan or analyze the information—SAX provides no container for the document like the DOM tree structure.

Is having two completely different ways to parse XML data a problem? Not really, both parsers have very different approaches for processing the information. The W3C DOM specification provides a very rich and intuitive structure for housing the XML data, but can be quite resource-intensive given that the entire XML document is typically stored in memory. You can manipulate the DOM at run-time and stream the updated data as XML, or transform it to your own format if you require.

The strength of the SAX specification is that it can scan and parse gigabytes worth of XML documents without hitting resource limits, because it does not try to create the DOM representation in memory. Instead, it raises events that you can handle as you see fit. Because of this design, the SAX implementation is generally faster and requires fewer resources. On the other hand, SAX code is frequently complex, and the lack of a document representation leaves you with the challenge of manipulating, serializing, and traversing the XML document.

Submitted to Birla Institute of Technology, Mesra, Ranchi (Jaipur Campus)

Putting the Parser to the Test

To determine the right parser for you, prioritize the importance of functionality, speed, memory requirements, and class footprint size. A few types of tests can help you evaluate them, although the performance of some depends on the specific nature and design of your software. These tests include parsing large and small XML documents, traversing and navigating the processed DOM, constructing a DOM from scratch, and evaluating the resource requirements of the parser.

You can tell quite a bit about a parser by using one or two simple XML documents. If your software will have to deal with many small files, see if the parser has some initialization overhead that slows down repeated parsing. For very large files, confirm that the parser can interpret the file in sufficient time with reasonable resource requirements. For the latter case, very large XML documents may require using a SAX parser that does not store the document in memory. You might also consider reading in parts of the document (using an appropriate DTD that allows for a partial document) and manipulating the document fragments in memory, one at a time.
In addition, new DOM parsing solutions may handle massive XML documents more effectively. Remember that the DOM API specifies only how to interact with the document, not how it must be stored. Persistent DOM (PDOM) implementations with index-based searches and retrieval are in the works, but I have not yet tested any of these.

You should also evaluate how well the parser traverses an in-memory DOM after XML data has been parsed. If you require the ability to search or scan through a post-parsed DOM using the API, you can rule out SAX—unless you are willing to create your own document model from your callback functions. For W3C DOM-compliant parsers, test the speed of scanning through the constructed DOM to see how expensive traversal of the tree can be.
Some XML parsers come with a serialization feature and are able to convert a document tree to XML data. This capability is not in all parsers, but the performance of parsers that support this ability is often proportional to the time required to navigate a given document tree using the API. Again, because SAX does not support an internal representation of the document, you would have to provide your own document and serialization functionality.
XML has arrived. Configuration files, application file formats, even database access layers make use of XML-based documents. Fortunately, several high-quality implementations of the standard APIs for handling XML are available. Unfortunately, these APIs are large and therefore provide a formidable hurdle for the beginner.

Submitted to Birla Institute of Technology, Mesra, Ranchi (Jaipur Campus)

In this article, I would like to offer an accessible introduction to the two most widely used APIs: SAX and DOM. For each API, I will show a sample application that reads an XML document and turns it into a set of Java objects representing the data in the document, a process known as XML "unmarshalling."
First, a word on style. For instructional purposes, I have kept the code as simple as possible. In order to focus on the basic usage of SAX and DOM, I completely omitted error handling and handling of XML namespaces, among other things. Furthermore, the code has not been tuned for flexibility or elegance; it may be dull, but hopefully it is also obvious.

The 60-Second XML Skinny:

For those completely new to XML, I would like to review the most important terms and concepts used with XML data.
Each XML document starts with a prologue, followed by the actual document content. The prologue begins with an XML declaration, such as:
<?xml version="1.0" standalone="yes" ?>
The declaration must be at the very beginning of the document -- not even whitespace may precede it! It is followed by the document type declaration, which in the present case only names the root element (catalog), but in a real-world application would also provide a link to a constraint as provided by a Document Type Definition (DTD) or XML Schema document:
<!DOCTYPE catalog>
This concludes the prologue. The following body of the XML document is made up of elements, which take the role of (and look like) familiar HTML tags. Every element has a name, and may have an arbitrary number of attributes:
<catalog version="1.0">...</catalog>
Here catalog is the name of the element, having one attribute named version, with value 1.0. In contrast to HTML, XML element names are case sensitive and must be closed with the appropriate closing tag. Note that there must be no space between the opening angle bracket and the element name. If the element contains neither text nor other elements, the closing tag may be merged with the start tag (a so-called empty tag):
<catalog version="1.0" />
An element may include either text, or other elements, or a combination of both. Text may include entity references, similar to those in HTML. In short, an entity reference is a placeholder for another piece of data. They are often used to include special characters, such as angle brackets: < or >. Entity references consist of a ampersand, followed by the entity name and a semicolon:

Submitted in Birla Institute of Technology, Mesra, Ranchi (Jaipur Campus)

&entityname;
XML elements have to be properly nested; in particular, the opening and closing tags of different elements must not overlap. In other words, an element's opening and end tags must reside in the same parent. This establishes a clear parent/child relationship among all elements of an XML document. Finally, the outermost element (the one following the prologue) is called the root element.
An element name may be qualified by an XML namespace prefix, yielding a qualified, or qNam. The namespace prefix is in the form of a Universal Resource Identifier (URI) and is followed by the local name after a colon:
namespace:localname
A document following these rules is syntactically well-formed. This is to be distinguished from its validity, which refers to adherence to the constraint laid out in the DTD or XML Schema document. Note that for a document that does not specify a constraint (such as the example document below), the concept of validity makes no sense.

The XML Document and Data Objects:

The document to read describes the catalog of a library. The catalog may contain an arbitrary number of books and magazines. Each book has a title and exactly one author. Each magazine has a name and may contain an arbitrary number of articles. Finally, each article has a headline and a starting page.

```
<?xml version="1.0"?>

<catalog library="somewhere">

  <book>
    <author>Author 1</author>
    <title>Title 1</title>
  </book>

  <book>
    <author>Author 2</author>
    <title>His One Book</title>
  </book>

  <magazine>
    <name>Mag Title 1</name>

    <article page="5">
      <headline>Some Headline</headline>
    </article>

    <article page="9">
      <headline>Another Headline</headline>
    </article>
  </magazine>

  <book>
    <author>Author 2</author>
```

Submitted to Birla Institute of Technology, Mesra, Ranchi (Jaipur Campus)

```
    <title>His Other Book</title>
  </book>

  <magazine>
    <name>Mag Title 2</name>

    <article page="17">
      <headline>Second Headline</headline>
    </article>
  </magazine>

</catalog>
```

Note that the starting page is encoded as an attribute of the article element. This is done primarily to demonstrate the use of attributes, although it can be argued that this design decision is actually semantically justified, since the starting page of an article is information *about* the article, but not part of the article itself.

In the example text, the following elements (called "complex elements" for the purpose of this article) may contain other elements:

- <catalog>
- <book>
- <magazine>
- <article>

The "simple" elements are those that contain only text:

- <author>
- <title>
- <name>
- <headline>

There are no elements that contain both text and child elements simultaneously.

The complex elements are represented in the application code by classes, whereas the simple elements are java.lang.String member variables of these classes. Since the sole purpose of these classes is to bundle the data read from the document, their interface has been kept minimal: they can be instantiated, their data members can be set, and finally, they override the toString() method, so as to allow access to the data inside.

```
class Catalog {
    private Vector books;
    private Vector magazines;

    public Catalog() {
        books = new Vector();
        magazines = new Vector();
    }

    public void addBook( Book rhs ) {
        books.addElement( rhs );
```

Submitted to Birla Institute of Technology, Mesra, Ranchi (Jaipur Campus)

```
    }
    public void addMagazine( Magazine rhs ) {
        magazines.addElement( rhs );
    }

    public String toString() {
        String newline = System.getProperty( "line.separator" );
        StringBuffer buf = new StringBuffer();

        buf.append( "--- Books ---" ).append( newline );
        for( int i=0; i<books.size(); i++ ){
            buf.append( books.elementAt(i) ).append( newline );
        }

        buf.append( "--- Magazines ---" ).append( newline );
        for( int i=0; i<magazines.size(); i++ ){
            buf.append( magazines.elementAt(i) ).append( newline );
        }

        return buf.toString();
    }
}

// ---------------------------------------------------------------

class Book {
    private String author;
    private String title;

    public Book() {}

    public void setAuthor( String rhs ) { author = rhs; }
    public void setTitle(  String rhs ) { title  = rhs; }

    public String toString() {
        return "Book: Author='" + author + "' Title='" + title + "'";
    }
}

// ---------------------------------------------------------------

class Magazine {
    private String name;
    private Vector articles;

    public Magazine() {
        articles = new Vector();
    }

    public void setName( String rhs ) { name = rhs; }

    public void addArticle( Article a ) {
        articles.addElement( a );
    }

    public String toString() {
        StringBuffer buf = new StringBuffer(
        "Magazine: Name='" + name + "' ");
        for( int i=0; i<articles.size(); i++ ){
            buf.append( articles.elementAt(i).toString() );
        }
```

Submitted to Birla Institute of Technology, Mesra, Ranchi (Jaipur Campus)

```
        return buf.toString();
    }
}

// ---------------------------------------------------------------

class Article {
    private String headline;
    private String page;

    public Article() {}

    public void setHeadline( String rhs ) { headline = rhs; }
    public void setPage(     String rhs ) { page     = rhs; }

    public String toString() {
        return "Article: Headline='" + headline + "' on page='" +
page + "' ";
    }
}
```

The classes have not been declared public, therefore they have package visibility. The primary consequence of this is that all of them can be defined in the same source file. (To remove possible confusion: the variable name rhs used in the setter methods stands for right-hand-side -- a very convenient naming convention for assignments!)

Unmarshalling with SAX

SAX, the Simple API for XML, is a traditional, event-driven parser. It reads the XML document incrementally, calling certain callback functions in the application code whenever it recognizes a token. Callbacks events are generated for the beginning and the end of a document, the beginning and end of an element, etc. They are defined in the interface org.xml.sax.ContentHandler, which every SAX-based document handler class must implement. It is the responsibility of the application programmer to implement these callback functions. Often, the application may not care about certain events reported by the SAX parser. For these cases, there exists a convenience class, org.xml.sax.helpers.DefaultHandler, which provides empty implementations for all functions defined in ContentHandler; custom classes simply extend DefaultHandler and need only override those callbacks in which they are specifically interested. This is done in the code below.

At the heart of a program (or class) utilizing the SAX parser typically lies a stack. Whenever an element is started, a new data object of the appropriate type is pushed onto the stack. Later, when the element is closed, the topmost object on the stack has been

Submitted to Birla Institute of Technology, Mesra, Ranchi (Jaipur Campus)

finished and can be popped. Unless it has been the root element (in which case the stack will be empty after it has been popped), the most recently popped element will have been a child element of the object that now occupies the top position of the stack, and can be inserted into its parent object. This process corresponds to the shift-reduce cycle of bottom-up parsers. Note how the requirement that XML elements must not overlap is crucial for the proper functioning of this idiom.

Example 1. Unmarshalling with SAX.

```
class SaxCatalogUnmarshaller extends DefaultHandler {
    private Catalog catalog;

    private Stack stack;
    private boolean isStackReadyForText;

    private Locator locator;

    // -----

    public SaxCatalogUnmarshaller() {
        stack = new Stack();
        isStackReadyForText = false;
    }

    public Catalog getCatalog() { return catalog; }

    // ----- callbacks: -----

    public void setDocumentLocator( Locator rhs ) { locator = rhs; }

    // -----

    public void startElement( String uri, String localName, String
qName,
                              Attributes attribs ) {

        isStackReadyForText = false;

        // if next element is complex, push a new instance on the
stack
        // if element has attributes, set them in the new instance
        if( localName.equals( "catalog" ) ) {
            stack.push( new Catalog() );

        }else if( localName.equals( "book" ) ) {
            stack.push( new Book() );

        }else if( localName.equals( "magazine" ) ) {
            stack.push( new Magazine() );

        }else if( localName.equals( "article" ) ) {
            stack.push( new Article() );
            String tmp = resolveAttrib( uri, "page", attribs,
"unknown" );
            ((Article)stack.peek()).setPage( tmp );
```

Submitted to Birla Institute of Technology, Mesra, Ranchi (Jaipur Campus)

```
        }
        // if next element is simple, push StringBuffer
        // this makes the stack ready to accept character text
        else if( localName.equals( "title" ) || localName.equals(
"author" ) ||
                 localName.equals( "name"  ) || localName.equals(
"headline" ) ) {
            stack.push( new StringBuffer() );
            isStackReadyForText = true;
        }
        // if none of the above, it is an unexpected element
        else{
            // do nothing
        }
    }

    // -----

    public void endElement( String uri, String localName, String
qName ) {

        // recognized text is always content of an element
        // when the element closes, no more text should be expected
        isStackReadyForText = false;

        // pop stack and add to 'parent' element, which is next on
the stack
        // important to pop stack first, then peek at top element!
        Object tmp = stack.pop();

        if( localName.equals( "catalog" ) ) {
            catalog = (Catalog)tmp;

        }else if( localName.equals( "book" ) ) {
            ((Catalog)stack.peek()).addBook( (Book)tmp );

        }else if( localName.equals( "magazine" ) ) {
            ((Catalog)stack.peek()).addMagazine( (Magazine)tmp );

        }else if( localName.equals( "article" ) ) {
            ((Magazine)stack.peek()).addArticle( (Article)tmp );
        }
        // for simple elements, pop StringBuffer and convert to
String
        else if( localName.equals( "title" ) ) {
            ((Book)stack.peek()).setTitle( tmp.toString() );

        }else if( localName.equals( "author" ) ) {
            ((Book)stack.peek()).setAuthor( tmp.toString() );

        }else if( localName.equals( "name" ) ) {
            ((Magazine)stack.peek()).setName( tmp.toString() );

        }else if( localName.equals( "headline" ) ) {
            ((Article)stack.peek()).setHeadline( tmp.toString() );
        }
        // if none of the above, it is an unexpected element:
        // necessary to push popped element back!
        else{
            stack.push( tmp );
        }
```

Submitted to Birla Institute of Technology, Mesra, Ranchi (Jaipur Campus)

```
    }

    // -----

    public void characters( char[] data, int start, int length ) {

        // if stack is not ready, data is not content of recognized
element
        if( isStackReadyForText == true ) {
            ((StringBuffer)stack.peek()).append( data, start, length
);
        }else{
        // read data which is not part of recognized element
        }
    }

    // -----

 private String resolveAttrib
(String uri, String localName, Attributes attribs,
 String defaultValue )
    {

        String tmp = attribs.getValue( uri, localName );
        return (tmp!=null)?(tmp):(defaultValue);
    }
}
```

Of the various callback methods declared in the ContentHandler interface, only four are implemented here. In unmarshalling a document, we are primarily interested in the contents that are encoded in it. Therefore, the relevant events are the beginning and end of an element, and the occurrence of raw character data inside an element. We also implement the setDocumentLocator() method. Although not used in the application code, it can be very helpful in debugging. The org.xml.sax.Locator interface acts like a cursor, pointing to the position in the XML document where the last event occurred. It provides useful methods such as getLineNumber() and getColumnNumber().

Start of Element

When the startElement() function is called, the SAX parser passes it a number of arguments. The first three are (in order): the namespace URI, the local name, and the fully qualified name of the element. By default, only the URI and the local name need to be supplied, while the qualified name is optional. Since the catalog document does not introduce any XML namespaces, we only use the local name in the present application.
The last argument holds the attributes of the present element (if any) in a specific container, which allows retrieval of the attributes by their names, as well as iteration over all attributes using an integer index.

Submitted to Birla Institute of Technology, Mesra, Ranchi (Jaipur Campus)

Elements are recognized by their local names. If the current element is a complex element, an object of the appropriate type is instantiated and pushed onto the stack. If the current element is simple, a new StringBuffer is pushed onto the stack instead, ready to accept character data.
Finally, the <article> element has an attribute, which is read from the attribs argument and inserted into the newly created article object on top of the stack. The attribute is extracted using the convenience function resolveAttrib(), which returns the attribute value or a default text, if the attribute is missing.

End of Element

The endElement() function is called with essentially the same arguments as the startElement() function; only the list of attributes is missing. In any case, the topmost element on the stack is popped, converted to the proper type, and inserted into its parent, which now occupies the top of the stack. Only the root element, which has no parent, is treated differently.

Raw Text

Finally, the callback function named characters() is called when the parser encounters raw text. It is passed a char array, containing the actual data, as well as a position at which to start reading and the length of data to be read from the array. Of course, it is illegal to access the data array outside of those boundaries. The implementation of the callback method inserts the data into the StringBuffer on the stack.
The way the characters() function is called by the underlying SAX parser often leads to some initial confusion, for two reasons. Firstly, there is no guarantee that a stretch of contiguous data results in only a single call to characters() -- it would be perfectly legal for the parser to invoke the callback function for each individual character of text! Although this is certainly an extreme scenario, it is quite common for text with embedded entity references to result in several calls to characters(): one for the text before the reference, a separate call for the entity itself, and finally, one for the remaining text. This is the reason that a StringBuffer is pushed on the stack if a simple element is encountered when reading the example document. (In fact, using a StringBuffer with the characters() callback function is a common idiom when using the SAX API.)
The second reason that characters() can lead to confusion results from the fact that it is called for *all* text characters encountered by the parser, including whitespace, even the whitespace between element tags (such as newlines and tabs). This is surprising, since ContentHandler defines a special callback method ignorableWhitespace(), taking the same arguments as characters().

Submitted to Birla Institute of Technology, Mesra, Ranchi (Jaipur Campus)

However, without a DTD or XML Schema, this method is never called, since there is no way for the parser to distinguish whether some whitespace is ignorable or not. In the present example program, the boolean flag isStackReady serves to distinguish between the two. The stack only becomes ready to accept text when a simple element has started and before it has ended.

Unmarshalling with DOM

The Document Object Model (DOM) describes an XML document as a tree-like structure, with every XML element being a node in the tree. A DOM-based parser reads the *entire* document, and (at least in principle) forms the corresponding document tree in memory. The DOM tree is formed from classes that all implement the org.w3c.dom.Node interface. This interface provides functions to walk or modify the tree (such as getChildNodes(), or appendChild() and removeChild()), and, of course, methods to query each node for its name and value.

The present unmarshalling code does not need to modify the DOM tree. The tree traversal itself is essentially recursive: the root node is unmarshalled, then each of its child nodes (which are either of type book or magazine), and, in the case of the magazine, its children (article). Whenever a child node has been unmarshalled, the resulting object representation of that node is inserted into the parent object.

Example 2. Unmarshalling with DOM.

Submitted to Birla Institute of Technology, Mesra, Ranchi (Jaipur Campus)

```
class DomCatalogUnmarshaller {

    public DomCatalogUnmarshaller() { }

    //------

    public Catalog unmarshallCatalog( Node rootNode ) {
        Catalog c = new Catalog();

        Node n;
        NodeList nodes = rootNode.getChildNodes();

        for( int i=0 ; i<nodes.getLength(); i++ ){
            n = nodes.item( i );

            if( n.getNodeType() == Node.ELEMENT_NODE ){

                if( n.getNodeName().equals( "book" ) ) {
                    c.addBook( unmarshallBook( n ) );
```

```
            }else if( n.getNodeName().equals( "magazine" ) ){
                c.addMagazine( unmarshallMagazine( n ) );

            }else{
                // unexpected element in Catalog
            }
        }else{
            // unexpected node-type in Catalog
        }
    }
    return c;
}

// -----

private Book unmarshallBook( Node bookNode ) {
    Book b = new Book();

    Node n;
    NodeList nodes = bookNode.getChildNodes();

    for( int i=0 ; i<nodes.getLength(); i++ ){
        n = nodes.item( i );

        if( n.getNodeType() == Node.ELEMENT_NODE ){

            if( n.getNodeName().equals( "author" ) ){
                b.setAuthor( unmarshallText( n ) );

            }else if( n.getNodeName().equals( "title" ) ){
                b.setTitle( unmarshallText( n ) );

            }else{
                // unexpected element in Book
            }
        }else{
            // unexpected node-type in Book
        }
    }
    return b;
}

// -----

private Magazine unmarshallMagazine( Node magazineNode ) {
    Magazine m = new Magazine();

    Node n;
    NodeList nodes = magazineNode.getChildNodes();

    for( int i=0 ; i<nodes.getLength(); i++ ){
        n = nodes.item( i );

        if( n.getNodeType() == Node.ELEMENT_NODE ){

            if( n.getNodeName().equals( "name" ) ) {
                m.setName( unmarshallText( n ) );

            }else if( n.getNodeName().equals( "article" ) ) {
                m.addArticle( unmarshallArticle( n ) );
```

Submitted to Birla Institute of Technology, Mesra, Ranchi (Jaipur Campus)

```
                }else{
                    // unexpected element in Magazine
                }
            }else{
                // unexpected node-type in Magazine
            }
        }
        return m;
    }

    // -----

    private Article unmarshallArticle( Node articleNode ) {
        Article a = new Article();

        if( articleNode.hasAttributes() == true ) {
            a.setPage( unmarshallAttribute( articleNode, "page",
"unknown" ) );
        }

        Node n;
        NodeList nodes = articleNode.getChildNodes();

        for( int i=0 ; i<nodes.getLength(); i++ ){
            n = nodes.item( i );

            if( n.getNodeType() == Node.ELEMENT_NODE ){

                if( n.getNodeName().equals( "headline" ) ) {
                    a.setHeadline( unmarshallText( n ) );

                }else{
                    // unexpected element in Article
                }
            }else{
                // unexpected node-type in Article
            }
        }
        return a;
    }

    // -----

    private String unmarshallText( Node textNode ) {
        StringBuffer buf = new StringBuffer();

        Node n;
        NodeList nodes = textNode.getChildNodes();

        for( int i=0; i<nodes.getLength(); i++ ){
            n = nodes.item( i );

            if( n.getNodeType() == Node.TEXT_NODE ) {
                buf.append( n.getNodeValue() );
            }else{
                // expected a text-only node!
            }
        }
        return buf.toString();
    }
```

Submitted to Birla Institute of Technology, Mesra, Ranchi (Jaipur Campus)

```
    // -----

    private String unmarshallAttribute( Node node,
        String name, String defaultValue ){
        Node n = node.getAttributes().getNamedItem( name );
        return (n!=null)?(n.getNodeValue()):(defaultValue);
    }
}
```

There are subtypes of the Node interface representing elements, text, comments, entities, and many others. The tree model, by which each part of the document is represented as a Node, is followed very consistently. Character data, for instance, is considered a child of its enclosing Element and is represented by its own Text instance, which has to be queried using getNodeValue() to find the actual string.
The Node supertype offers getNodeName(), getNodeValue(), and getAttributes() to provide access to information about a Node instance without having to downcast it.
Not all three of these methods make sense for every node type, however. For instance, only an Element can have attributes; for all other Node subtypes the corresponding function returns null. For Element nodes, getNodeName() returns the tag name, but getNodeValue() returns null. In contrast, for a Text node, getNodeValue() returns the character data, while getNodeName() returns the fixed string "#TEXT". The www.w3.org DOM specification contains a table detailing the behavior of all three functions for every possibly node type.
In the present program, we are only interested in three kinds of nodes: those representing elements, text, and attributes. All of the unmarshalling functions are very similar to each other. They accept the topmost node of the subtree they are to unmarshall as an argument. Then they create an object representing the current node and iterate over its child nodes, unmarshalling each in turn. If a child node describes a complex element, the node is passed on to the appropriate unmarshalling function, depending on the element name. A child node of type TEXT_NODE describes a simple element, and the node value is simply the character data.
Nodes describing attributes are a bit different, since attributes are not really part of the document's tree structure: attributes are not proper children of the elements in which they are contained. They can therefore not be reached by tree-walking operations; instead, the Node class provides a getAttributes() function, which returns a collection of key/value-pairs, containing the attributes. Again, we provide a convenience function that returns a default value in case no attribute can be found for the given name.

Submitted to Birla Institute of Technology, Mesra, Ranchi (Jaipur Campus)

The Driver:

Finally, we need a driver class, containing static void main(). The main() function reads the API to use (SAX or DOM) and the name of the XML file from the command line. It creates a org.xml.sax.InputSource from the filename. This class is acceptable to both SAX and DOM as an encapsulation of an XML document. Then it creates instances of the the appropriate parser and unmarshaller classes and passes the input file to them. Finally, it prints the contents of the created objects to standard output.

Example 3. Driver class.

```
public class Driver {

    public static void main( String[] args ) {
        Catalog catalog = null;

        try {
            File file = new File( args[1] );
            InputSource src = new InputSource( new FileInputStream(
file ) );

            if( args[0].equals( "SAX" ) ) {
                System.out.println( "--- SAX ---" );

                SaxCatalogUnmarshaller saxUms = new
SaxCatalogUnmarshaller();

                XMLReader rdr = XMLReaderFactory.
                    createXMLReader(
"org.apache.xerces.parsers.SAXParser" );
                rdr.setContentHandler( saxUms );
                rdr.parse( src );

                catalog = saxUms.getCatalog();

            }else if( args[0].equals( "DOM" ) ) {
                System.out.println( "--- DOM ---" );

                DomCatalogUnmarshaller domUms = new
DomCatalogUnmarshaller();

                org.apache.xerces.parsers.DOMParser prsr =
                    new org.apache.xerces.parsers.DOMParser();
                prsr.parse( src );
                Document doc = prsr.getDocument();

                catalog = domUms.unmarshallCatalog(
                doc.getDocumentElement() );

            }else{
                 System.out.println( "Usage: SAX|DOM filename" );
                 System.exit(0);
             }

            System.out.println( catalog.toString() );
```

Submitted to Birla Institute of Technology, Mesra, Ranchi (Jaipur Campus)

```
        }catch( Exception exc ) {
            System.out.println( "Usage: SAX|DOM filename" );
            System.err.println( "Exception: " + exc );
        }
    }
}
```

SAX and DOM are interface specifications. Implementations of these interfaces are available from various sources (both commercial and free), and it is part of the driver's responsibility to load the specific parser class. The code above uses the Apache Xerces implementations of the SAX and DOM specifications; these are freely available, open source, high-quality implementations. Be sure that the corresponding classes are included in your CLASSPATH.
The SAX specification contains a factory class that can be used to select which SAX parser implementation will be used. After instantiating the XMLReader class, we need to register with it our SAX unmarshaller as application-specific content handler. Finally, we can retrieve the unmarshalled objects from the unmarshaller instance.
As opposed to SAX, the DOM specification covers only the tree representation of the XML document. Instantiating and using the parser is not actually covered by DOM itself, and the specific implementation must be named directly in the application code. After the input document has been parsed, the resulting DOM tree can be retrieved from the parser using the getDocument function, which returns a Document instance. The Document interface extends the Node interface and represents the root node of the document. It is then used with the appropriate unmarshaller class, similar to the SAX case.

Conclusion:

It bears repeating that the code above is for instructional purposes *only*. It ignores many XML structures (such as namespaces, entities, and, of course, constraints), as well as more advanced features of the parser classes (such as additional SAX callback handlers, or more powerful ways to walk and modify a DOM tree). But the most immediate omission concerns the handling of unexpected elements and similar errors. The locations in the code where these conditions should be handled are clearly marked. It can be enlightening to insert some logging code and then observe the behavior of the program after some "errors" (such as unexpected elements) have been introduced into the XML document. Finally, the document structure has been hard-coded into program. A real-world application would need greater flexibility, or at least better diagnostics.
I hope to have demonstrated how to use either API to parse a simple XML document and turn its data into a set of Java objects.

Submitted to Birla Institute of Technology, Mesra, Ranchi (Main Campus)

The example application is simple, but it should be enough to get you started. The references contain additional resources.

Submitted to Birla Institute of Technology, Mesra, Ranchi (Jaipur Campus)

2.3. Steps in Java Native Interface:

Step 1: Make the list of native functions to be used in Java.

```
getApp(Extension);
getAppPath(AppicationName);
getAppListName(String_Array_To_Store_All_Application_Name);
getUserDSN();
```

Step 2: Write the java code with the prototype of native function to be used in Java. Using the native keyword.

```
public native String getUserDSN();
```

Step 3: Save and Compile the java file(ReadRegistry.java) and create the class file.

Step 4: Create the Java Header File using the Javah command. To this we have to take care to include the package name as well while creating the Header file. using javah <Package_Name>.<ClassFileName>

```
javah xyz.abc.utility.ReadRegistry
```

Step 5: Select the Project type as Dynamic Link Library while creating the C/C++ Project in VC++.

Step 6: Copy the Java Header File(ReadRegistry.h) in the VC++/C Project folder.

Step 7: Write the C/C++ code. For this we have to include the Java Header File and also the include the "jni.h" header file.

```
#include "jni.h"
#include "ReadRegistry.h"
```

Step 8: Copy the Prototype of the native methods from the Java Header File in to the C/C++ code.

Step 9: Write the implementation code for native methods in C/C++ compile it and build the dll.

Step 10: Copy the dll created to the System32 folder, which is located generally in C:\Winnt or C:\Windows folder

Submitted to Birla Institute of Technology, Mesra, Ranchi (Jaipur Campus)

Step 11: Come back to the java code and write the code for using the native methods, also write the code for loading Library.

System.loadLibrary("ReadRegistry");

Example:

```
#include <windows.h>
#include<jni.h>
#include "ReadRegistry.h"
#include<string.h>
#include<stdio.h>

#define BUFSIZE 80
#define MAX_KEY_LENGTH 255
#define MAX_VALUE_NAME 16383
#define ARRSIZE 50
#define VARNAME "a"

JNIEXPORT jint JNICALL Java_xyz_abc_utility_ReadRegistry_getUserDSN
(JNIEnv *env, jobject obj, jobjectArray objArr)
{
    HKEY     hKey;
    LONG     lreturnStatus;
    //buffer for class name
    TCHAR    TBufferName[2000] = TEXT("");
    //buffer for subkey name
    TCHAR    tAchKey[MAX_KEY_LENGTH] = TEXT("");
    //size of name string
    DWORD    dSizeName;
    //buffer for class name
    TCHAR    tAchClass[MAX_PATH] = TEXT("");
    //size of class string
    DWORD    dwSizeClassName = MAX_PATH;
    //number of subkeys
    DWORD    dwSubKeys=0;
    //longest subkey size
    DWORD    dwMaxSubKey;
    //longest class string
    DWORD    dwMaxClass;
    //number of values for key
    DWORD    dwNoOfValues;
    //longest value name
    DWORD    dwMaxValueName;
    //longest value data
    DWORD    dwMaxValueData;
    //size of security descriptor
    DWORD    dwSizeSecurityDescriptor;
    //last write time
    FILETIME ftLastWriteTime;

    //size of class string
    long lSizeNameLength = 2000;
    unsigned int iLsubscript=0;

    lreturnStatus = RegOpenKeyEx(
       HKEY_CURRENT_USER,
       "Software\\ODBC\\ODBC.INI",
       0L,
       KEY_QUERY_VALUE|KEY_READ, &hKey
       );
       if(lreturnStatus == ERROR_SUCCESS)
```

Submitted to Birla Institute of Technology, Mesra, Ranchi (Jaipur Campus)

```
        {
            printf("ODBC Exists\n");
            //lreturnStatus = RegQueryMultipleValues
            lreturnStatus = RegQueryInfoKey(
             // key handle
            hKey,
             // buffer for class name
            tAchClass,
             // size of class string
            &dwSizeClassName,
             // reserved
            NULL,
             // number of subkeys
            &dwSubKeys,
             // longest subkey size
            &dwMaxSubKey,
             // longest class string
             &dwMaxClass,
             // number of values for this key
            &dwNoOfValues,
             // longest value name
            &dwMaxValueName,
             // longest value data
            &dwMaxValueData,
             // security descriptor
            &dwSizeSecurityDescriptor,
             // last write time
            &ftLastWriteTime
             );
            if(lreturnStatus == ERROR_SUCCESS)
            {
               for(iLsubscript=0; iLsubscript<dwSubKeys; iLsubscript++)
               {
                   dSizeName = MAX_KEY_LENGTH;
                   lreturnStatus = RegEnumKey(hKey,
                     iLsubscript,
                     tAchKey,
                   dSizeName);
                   if (lreturnStatus == ERROR_SUCCESS)
                   {
                       //printf("%d:%s\n",iLsubscript,(char *)tAchKey);
                       (*env)->SetObjectArrayElement(env, objArr,iLsubscript,
                       (*env)->NewStringUTF(env,
                       (char *)tAchKey));
                   }
               }
            }
        }
        else
        {
            Printf("Opening Software Failed,
            Code is:%d\n",lreturnStatus);
        }
        return iLsubscript;
}
```

Submitted to Birla Institute of Technology, Mesra, Ranchi (Jaipur Campus)

2.4. Definitions & Acronyms:

Transmission Control Protocol/Internet Protocol reserve lower 1024 ports for specific protocols and the higher are free and can be used for the other purposes. Like port no. 21 is reserved for FTP. Port no. 23 for telnet Port no. 25 for email Port no. 119 for net users. It is up to protocol to determine how a client should interact with port.

◎ HTTP:

It is a protocol that web users and servers use to transfer Hyper Text Pages and images.

◎ Proxy Server:

A proxy server speaks the client side of a protocol to another server. A proxy server has a additional ability to filter certain requests or cache the results for specific requests that may come in future.

◎ Threads:

Thread are one of the most important aspects of our application. Since our application demand simultaneous processing of several tasks and so the threads are used. Moreover, java provides a very good mechanism of creating threads.

Ways to create a new thread:

Method 1:
Extend the thread class in your class and override the method run.

```
public class mythread extends thread
{
      public      void run()
      {
                  ……
                  ……
      }
}
```

Submitted to Birla Institute of Technology, Mesra, Ranchi (Jaipur Campus)

Method 2:

Implementing the runnable interface (the class thread itself is an implementation of runnable) and use that class in thread construction.

```
public class mythread implements Runnable
{
    public void run()
     {
                ……
                ……
     }
}
```

Thread t1= new Thread(new mythread);

The general thread methods used are:
wait();
notify();
notifyall();

All these methods can be called only with in a synchronized method.

Options:

ThreadGroup(String groupName)
ThreadGroup(ThreadGroup parentobject,String groupName)

Example for Timeout Listener:

```
public class TimeoutListener
    implements MouseListener, MouseMotionListener,
            KeyListener, ActionListener {

public static long longGlastTime;

/**
 * Boolean to stop the dialog from displaying more than once
 */
private static boolean boolGTimedOut = false;
long longGtimeout;
Object ObjGlock;
private Logger objGLogger = Logger.getLogger(AwanIdentityMain.class);

public TimeoutListener(long longPtimeout) {
        ObjGlock = new int[1];
        longGtimeout = longPtimeout;
        AwanTimeoutListener.longGlastTime =
System.currentTimeMillis();
}
public void actionPerformed(ActionEvent actionPEvent) {
       synchronized(ObjGlock) {
       if(!AwanTimeoutListener.boolGTimedOut) {
       // We really shouldn't exit, but cleanup gracefully instead
              if((System.currentTimeMillis()
```

Submitted to Birla Institute of Technology, Mesra, Ranchi (Jaipur Campus)

```
                            - AwanTimeoutListener.longGlastTime)
                        >= longGtimeout) {
                    AwanTimeoutListener.boolGTimedOut = true;
                    objGLogger.debug("Timed out! "
                        + System.currentTimeMillis());
                    AwanCommonLib.informUser(new JFrame("Timed
Out!!"),
                        AwanUserFacConstIF.TIMEOUT_ERR,
                        AwanUserFacConstIF.TIMEOUT);
                    //System.exit(0);
                } else {
                    //Application is still active and is in use
                    objGLogger.debug("Still active. Last Time:"
                        + longGlastTime);
                }
            }    // end if
        }    // end synchronized
    }    // end method actionPerformed

    public void keyPressed(KeyEvent keyPEvent) {
        recordLastEvent(keyPEvent);
    }

    public void keyReleased(KeyEvent keyPEvent) {
        recordLastEvent(keyPEvent);
    }

    public void keyTyped(KeyEvent keyPEvent) {
        recordLastEvent(keyPEvent);
    }

    public void mouseClicked(MouseEvent mousePEvent) {
        recordLastEvent(mousePEvent);
    }

    public void mouseDragged(MouseEvent mousePEvent) {
        recordLastEvent(mousePEvent);
    }

    public void mouseEntered(MouseEvent mousePEvent) {
        recordLastEvent(mousePEvent);
    }

    public void mouseExited(MouseEvent mousePEvent) {
        recordLastEvent(mousePEvent);
    }

    public void mouseMoved(MouseEvent mousePEvent) {
        recordLastEvent(mousePEvent);
    }

    public void mousePressed(MouseEvent mousePEvent) {
        recordLastEvent(mousePEvent);
    }

    public void mouseReleased(MouseEvent mousePEvent) {
        recordLastEvent(mousePEvent);
    }

    private void recordLastEvent(InputEvent inputPEvent) {
        synchronized(ObjGlock) {
            AwanTimeoutListener.longGlastTime =
System.currentTimeMillis();
            System.out.println("Last Event occurred at: " +
longGlastTime);
        }
    }
}    // end class TimeoutListener
```

Submitted to Birla Institute of Technology, Mesra, Ranchi (Jaipur Campus)

Using the Class TimeoutListener:

```
/**
 * Variable to decide the allowable idle time by user in milliseconds
*/
public static final long TIMEOUT =5*60*1000;
AwanTimeoutListener ObjLtimeoutListener;
Timer Ltimer;

ObjLtimeoutListener = new AwanTimeoutListener(TIMEOUT);
this.addMouseListener(ObjLtimeoutListener);
this.addMouseMotionListener(ObjLtimeoutListener);
this.addKeyListener(ObjLtimeoutListener);


//We poll at half the timeout interval to reduce the fudge factor
//introduced by events delivered at just under the timeout value.
Ltimer = new Timer((int)TIMEOUT/2, ObjLtimeoutListener);
Ltimer.setRepeats(true);
Ltimer.start();
```

Submitted to Birla Institute of Technology, Mesra, Ranchi (Jaipur Campus)

◎ InetAddress class:

It is used to encapsulate both the numerical IP address and the domain name for that address. The InetAddress class has no visible constructors. We have to use the factory methods.

Factory Methods:

They are merely a convention where by static method in a class return a instance of that class.

static InetAddress getLocalHost()
static InetAddress getByName(String hostName)
static InetAddress getByName(String hostName)

Note: The above three methods throw UnknownHostException

Instance Methods:
boolean equals()
byte getAddress()
String getHostAddress()
String getHostName()
int hashcode()
boolean is MulticastAddress()
String toString()

Submitted to Birla Institute of Technology, Mesra, Ranchi (Jaipur Campus)

3. ECPG Language: Introduction

3.1. ecpg - Embedded SQL in C:

Embedded SQL has some small advantages over other ways to handle SQL queries. It takes care of all the tedious moving of information to and from variables in your C program. Many RDBMS packages support this embedded language.

There is an ANSI-standard describing how the embedded language should work. ecpg was designed to meet this standard as much as possible. So it is possible to port programs with embedded SQL written for other RDBMS packages to Postgres and thus promoting the spirit of free software.

3.2. The Concept

You write your program in C with some special SQL things. For declaring variables that can be used in SQL statements you need to put them in a special declare section. You use a special syntax for the SQL queries.

Before compiling you run the file through the embedded SQL C preprocessor and it converts the SQL statements you used to function calls with the variables used as arguments. Both variables that are used as input to the SQL statements and variables that will contain the result are passed.

Then you compile and at link time you link with a special library that contains the functions used. These functions (actually it is mostly one single function) fetches the information from the arguments, performs the SQL query using the ordinary interface (libpq) and puts back the result in the arguments dedicated for output.

Then you run your program and when the control arrives to the SQL statement the SQL statement is performed against the database and you can continue with the result.

Submitted to Birla Institute of Technology, Mesra, Ranchi (Jaipur Campus)

3.3. How To Use ecpg

This section describes how to use the ecpg tool.

3.3.1. Preprocessor

The preprocessor is called ecpg. After installation it resides in the Postgres bin/ directory.

3.3.2. Library

The ecpg library is called libecpg.a or libecpg.so. Additionally, the library uses the libpq library for communication to the Postgres server so you will have to link your program with -lecpg -lpq.
The library has some methods that are "hidden" but that could prove very useful sometime.

- ECPGdebug(int *on*, FILE **stream*) turns on debug logging if called with the first argument non-zero. Debug logging is done on *stream*. Most SQL statement logs its arguments and result.

  The most important one (ECPGdo) that is called on almost all SQL statements logs both its expanded string, i.e. the string with all the input variables inserted, and the result from the Postgres server. This can be very useful when searching for errors in your SQL statements.

- ECPGstatus() This method returns TRUE if we are connected to a database and FALSE if not.

3.3.3. Error handling

To be able to detect errors from the Postgres server you include a line like

```
exec sql include sqlca;
```

in the include section of your file. This will define a struct and a variable with the name *sqlca* as following:

```
struct sqlca
{
 char sqlcaid[8];
 long sqlabc;
 long sqlcode;
 struct
 {
  int sqlerrml;
  char sqlerrmc[70];
 } sqlerrm;
 char sqlerrp[8];
```

Submitted to Birla Institute of Technology, Mesra, Ranchi (Jaipur Campus)

```
 long sqlerrd[6];
 /* 0: empty                                        */
 /* 1: OID of processed tuple if applicable         */
 /* 2: number of rows processed in an INSERT, UPDATE */
 /*    or DELETE statement                          */
 /* 3: empty                                        */
 /* 4: empty                                        */
 /* 5: empty                                        */
 char sqlwarn[8];
 /* 0: set to 'W' if at least one other is 'W'      */
 /* 1: if 'W' at least one character string       */
 /*    value was truncated when it was              */
 /*    stored into a host variable.                 */
 /* 2: empty                                        */
 /* 3: empty                                        */
 /* 4: empty                                        */
 /* 5: empty                                        */
 /* 6: empty                                        */
 /* 7: empty                                        */
 char sqlext[8];
} sqlca;
```

If an error occured in the last SQL statement then sqlca.sqlcode will be non-zero. If sqlca.sqlcode is less that 0 then this is some kind of serious error, like the database definition does not match the query given. If it is bigger than 0 then this is a normal error like the table did not contain the requested row.
sqlca.sqlerrm.sqlerrmc will contain a string that describes the error. The string ends with the line number in the source file.
List of errors that can occur:

-12, Out of memory in line %d.
Does not normally occur. This is a sign that your virtual memory is exhausted.

-200, Unsupported type %s on line %d.
Does not normally occur. This is a sign that the preprocessor has generated something that the library does not know about. Perhaps you are running incompatible versions of the preprocessor and the library.

-201, Too many arguments line %d.
This means that Postgres has returned more arguments than we have matching variables. Perhaps you have forgotten a couple of the host variables in the **INTO :var1,:var2**-list.

-202, Too few arguments line %d.
This means that Postgres has returned fewer arguments than we have host variables. Perhaps you have too many host variables in the **INTO :var1,:var2**-list.

-203, Too many matches line %d.
This means that the query has returned several lines but the variables specified are no arrays. The **SELECT** you made probably was not unique.

-204, Not correctly formatted int type: %s line %d.

Submitted to Birla Institute of Technology, Mesra, Ranchi (Jaipur Campus)

This means that the host variable is of an int type and the field in the Postgres database is of another type and contains a value that cannot be interpreted as an int. The library uses strtol for this conversion.

-205, Not correctly formatted unsigned type: %s line %d.
This means that the host variable is of an unsigned int type and the field in the Postgres database is of another type and contains a value that cannot be interpreted as an unsigned int. The library uses strtoul for this conversion.

-206, Not correctly formatted floating point type: %s line %d.
This means that the host variable is of a float type and the field in the Postgres database is of another type and contains a value that cannot be interpreted as an float. The library uses strtod for this conversion.

-207, Unable to convert %s to bool on line %d.
This means that the host variable is of a bool type and the field in the Postgres database is neither 't' nor 'f'.

-208, Empty query line %d.
Postgres returned PGRES_EMPTY_QUERY, probably because the query indeed was empty.

-220, No such connection %s in line %d.
The program tries to access a connection that does not exist.

-221, Not connected in line %d.
The program tries to access a connection that does exist but is not open.

-230, Invalid statement name %s in line %d.
The statement you are trying to use has not been prepared.

-400, Postgres error: %s line %d.
Some Postgres error. The message contains the error message from the Postgres backend.

-401, Error in transaction processing line %d.
Postgres signalled to us that we cannot start, commit or rollback the transaction.

-402, connect: could not open database %s.
The connect to the database did not work.

100, Data not found line %d.
This is a "normal" error that tells you that what you are quering cannot be found or we have gone through the cursor.

Submitted to Birla Institute of Technology, Mesra, Ranchi (Jaipur Campus)

3.4. Limitations

What will never be included and why or what cannot be done with this concept.

Oracle's single tasking possibility

Oracle version 7.0 on AIX 3 uses the OS-supported locks on the shared memory segments and allows the application designer to link an application in a so called single tasking way. Instead of starting one client process per application process both the database part and the application part is run in the same process. In later versions of Oracle this is no longer supported.
This would require a total redesign of the Postgres access model and that effort can not justify the performance gained.

3.5. Porting From Other RDBMS Packages

The design of ecpg follows SQL standard. So porting from a standard RDBMS should not be a problem. Unfortunately there is no such thing as a standard RDBMS. So ecpg also tries to understand syntax additions as long as they do not create conflicts with the standard.
The following list shows all the known incompatibilities. If you find one not listed please notify Michael Meskes. Note, however, that we list only incompatibilities from a precompiler of another RDBMS to ecpg and not additional ecpg features that these RDBMS do not have.
Syntax of FETCH command

The standard syntax of the FETCH command is:
FETCH [direction] [amount] IN|FROM *cursor name*.
ORACLE, however, does not use the keywords IN resp. FROM. This feature cannot be added since it would create parsing conflicts.

3.6. For the Developer

This section is for those who want to develop the ecpg interface. It describes how the things work. The ambition is to make this section contain things for those that want to have a look inside and the section on How to use it should be enough for all normal questions. So, read this before looking at the internals of the ecpg. If you are not interested in how it really works, skip this section.

Submitted to Birla Institute of Technology, Mesra, Patna (Jaipur Campus)

3.6.1. ToDo List

This version the preprocessor has some flaws:

Library functions

to_date et al. do not exists. But then Postgres has some good conversion routines itself. So you probably won't miss these.

Structures and unions

Structures and unions have to be defined in the declare section.

Missing statements

The following statements are not implemented thus far:
exec sql allocate
exec sql deallocate
SQLSTATE

message 'no data found'

The error message for "no data" in an exec sql insert select from statement has to be 100.

sqlwarn[6]

sqlwarn[6] should be 'W' if the PRECISION or SCALE value specified in a SET DESCRIPTOR statement will be ignored.

3.6.2. The Preprocessor

The first four lines written to the output are constant additions by ecpg. These are two comments and two include lines necessary for the interface to the library.
Then the preprocessor works in one pass only, reading the input file and writing to the output as it goes along. Normally it just echoes everything to the output without looking at it further.
When it comes to an **EXEC** SQL statements it intervenes and changes them depending on what it is. The **EXEC** SQL statement can be one of these:
Declare sections

Declare sections begins with
exec sql begin declare section;

and ends with
exec sql end declare section;

In the section only variable declarations are allowed. Every variable declare within this section is also entered in a list of variables indexed on their name together with the corresponding type.
In particular the definition of a structure or union also has to be listed inside a declare section. Otherwise ecpg cannot handle these types since it simply does not know the definition.
The declaration is echoed to the file to make the variable a normal C-variable also.
The special types VARCHAR and VARCHAR2 are converted into a named struct for every variable. A declaration like:
VARCHAR var[180];

is converted into
struct varchar_var { int len; char arr[180]; } var;

Include statements

An include statement looks like:

Submitted to Birla Institute of Technology, Mesra, Ranchi (Jaipur Campus)

exec sql include filename;

Note that this is NOT the same as
#include <filename.h>

Instead the file specified is parsed by ecpg itself. So the contents of the specified file is included in the resulting C code. This way you are able to specify EXEC SQL commands in an include file.

Connect statement

A connect statement looks like:
exec sql connect to *connection target*;

It creates a connection to the specified database.
The *connection target* can be specified in the following ways:
dbname[@server][:port][as *connection name*][user *user name*]
tcp:postgresql://server[:port][/dbname][as *connection name*][user *user name*]
unix:postgresql://server[:port][/dbname][as *connection name*][user *user name*]
character variable[as *connection name*][user *user name*]
character string[as *connection name*][*user*]
default
user
There are also different ways to specify the user name:
userid
userid/*password*
userid identified by *password*
userid using *password*
Finally the userid and the password. Each may be a constant text, a character variable or a chararcter string.

Disconnect statements

A disconnect statement looks loke:
exec sql disconnect [*connection target*];

It closes the connection to the specified database.
The *connection target* can be specified in the following ways:
connection name
default
current
all

Open cursor statement

An open cursor statement looks like:
exec sql open *cursor*;

and is ignore and not copied from the output.

Submitted to Birla Institute of Technology, Mesra, Ranchi (Jaipur Campus)

Commit statement

A commit statement looks like
exec sql commit;

and is translated on the output to
ECPGcommit(__LINE__);

Rollback statement

A rollback statement looks like
exec sql rollback;

and is translated on the output to
ECPGrollback(__LINE__);

Other statements

Other SQL statements are other statements that start with exec sql and ends with ;. Everything inbetween is treated as an SQL statement and parsed for variable substitution.
Variable substitution occur when a symbol starts with a colon (:). Then a variable with that name is looked for among the variables that were previously declared within a declare section and depending on the variable being for input or output the pointers to the variables are written to the output to allow for access by the function.
For every variable that is part of the SQL request the function gets another ten arguments:

The type as a special symbol.

A pointer to the value or a pointer to the pointer.

The size of the variable if it is a char or varchar.

Number of elements in the array (for array fetches).

The offset to the next element in the array (for array fetches)

The type of the indicator variable as a special symbol.

A pointer to the value of the indicator variable or a pointer to the pointer of the indicator variable.

0.

Number of elements in the indicator array (for array fetches).

The offset to the next element in the indicator array (for array fetches)

Submitted to Birla Institute of Technology, Mesra, Ranchi (Jaipur Campus)

3.6.3. A Complete Example

Here is a complete example describing the output of the preprocessor of a file foo.pgc:

```
exec sql begin declare section;
int index;
int result;
exec sql end declare section;
...
```

```
exec sql select res into :result from mytable where index = :index;
```

is translated into:

```
/* Processed by ecpg (2.6.0) */
/* These two include files are added by the preprocessor */
#include <ecpgtype.h>;
#include <ecpglib.h>;

/* exec sql begin declare section */

#line 1 "foo.pgc"

 int index;
 int result;
/* exec sql end declare section */
...
ECPGdo(__LINE__, NULL, "select  res  from mytable where index = ?
",
        ECPGt_int,&(index),1L,1L,sizeof(int),
         ECPGt_NO_INDICATOR, NULL , 0L, 0L, 0L, ECPGt_EOIT,
         ECPGt_int,&(result),1L,1L,sizeof(int),
         ECPGt_NO_INDICATOR, NULL , 0L, 0L, 0L, ECPGt_EORT);
#line 147 "foo.pgc"
```

(the indentation in this manual is added for readability and not something that the preprocessor can do.)

3.6.4. The Library

The most important function in the library is the ECPGdo function. It takes a variable amount of arguments. Hopefully we will not run into machines with limits on the amount of variables that can be accepted by a vararg function. This could easily add up to 50 or so arguments.

The arguments are:

A line number

This is a line number for the original line used in error messages only.

A string

This is the SQL request that is to be issued. This request is modified by the input variables, i.e. the variables that where not known at compile time but are to be entered in the request. Where the variables should go the string contains ";".

Input variables

As described in the section about the preprocessor every input variable gets ten arguments.

ECPGt_EOIT

An enum telling that there are no more input variables.

Output variables

Submitted to Birla Institute of Technology, Mesra, Ranchi (Jaipur Campus)

As described in the section about the preprocessor every input variable gets ten arguments. These variables are filled by the function.

ECPGt_EORT

An enum telling that there are no more variables.

All the SQL statements are performed in one transaction unless you issue a commit transaction. To get this auto-transaction going the first statement or the first after statement after a commit or rollback always begins a transaction. To disable this feature per default use the -t option on the commandline.

To be completed: entries describing the other entries.

Submitted to Birla Institute of Technology, Mesra, Ranchi (Jaipur Campus)

4. PostgreSQL: An Introduction

PostgreSQL is a highly scalable, SQL compliant, open source object-relational database management system. With more than 15 years of development history, it is quickly becoming the de facto database for enterprise level open source solutions.

PostgreSQL is an object-relational database management system (ORDBMS) based on POSTGRES, Version 4.2, developed at the University of California at Berkeley Computer Science Department. POSTGRES pioneered many concepts that only became available in some commercial database systems much later.

PostgreSQL is an open-source descendant of this original Berkeley code. It supports SQL92 and SQL99 and offers many modern features:

- complex queries
- foreign keys
- triggers
- views
- transactional integrity
- multiversion concurrency control

Additionally, PostgreSQL can be extended by the user in many ways, for example by adding new

- data types
- functions
- operators
- aggregate functions
- index methods
- procedural languages

And because of the liberal license, PostgreSQL can be used, modified, and distributed by everyone free of charge for any purpose, be it private, commercial, or academic.

Submitted to Birla Institute of Technology, Mesra, Ranchi (Jaipur Campus)

PostgreSQL: History

A Brief History of PostgreSQL

The object-relational database management system now known as PostgreSQL is derived from the POSTGRES package written at the University of California at Berkeley. With over a decade of development behind it,

PostgreSQL is now the most advanced open-source database available anywhere.

The Berkeley POSTGRES Project

The POSTGRES project, led by Professor Michael Stonebraker, was sponsored by the Defense Advanced Research Projects Agency (DARPA), the Army Research Office (ARO), the National Science Foundation (NSF), and ESL, Inc.

The implementation of POSTGRES began in 1986. The initial concepts for the system were presented in The design of POSTGRES and the definition of the initial data model appeared in The POSTGRES data model. The design of the rule system at that time was described in The design of the POSTGRES rules system. The rationale and architecture of the storage manager were detailed in The design of the POSTGRES storage system.

POSTGRES has undergone several major releases since then. The first "demoware" system became operational in 1987 and was shown at the 1988 ACM-SIGMOD Conference. Version 1, described in The implementation of POSTGRES, was released to a few external users in June 1989. In response to a critique of the first rule system (A commentary on the POSTGRES rules system), the rule system was redesigned (On Rules, Procedures, Caching and Views in Database Systems) and Version 2 was released in June 1990 with the new rule system. Version 3 appeared in 1991 and added support for multiple storage managers, an improved query executor, and a rewritten rule system. For the most part, subsequent releases until Postgres95 focused on portability and reliability.

POSTGRES has been used to implement many different research and production applications. These include: a financial data analysis system, a jet engine performance monitoring package, an asteroid tracking database, a medical information database, and several geographic information systems. POSTGRES has also been used as an educational tool at several universities. Finally, Illustra Information Technologies (later merged into Informix, which is now owned by IBM.) picked up the code and commercialized it. In late 1992, POSTGRES became the primary data manager for the Sequoia 2000 scientific computing project.

The size of the external user community nearly doubled during 1993. It became increasingly obvious that maintenance of the prototype code and support was taking up large amounts of time that should have been devoted to database research. In an effort to reduce this support burden, the Berkeley POSTGRES project officially ended with Version 4.2.

Postgres95

In 1994, Andrew Yu and Jolly Chen added a SQL language interpreter to POSTGRES. Under a new name, Postgres95 was subsequently released to

Submitted to Birla Institute of Technology, Mesra, Ranchi (Jaipur Campus)

the web to find its own way in the world as an open-source descendant of the original POSTGRES Berkeley code.
Postgres95 code was completely ANSI C and trimmed in size by 25%. Many internal changes improved performance and maintainability. Postgres95 release 1.0.x ran about 30-50% faster on the Wisconsin Benchmark compared to POSTGRES, Version 4.2. Apart from bug fixes, the following were the major enhancements:

- The query language PostQUEL was replaced with SQL (implemented in the server). Subqueries were not supported until PostgreSQL (see below), but they could be imitated in Postgres95 with user-defined SQL functions. Aggregate functions were re-implemented. Support for the GROUP BY query clause was also added.
- In addition to the monitor program, a new program (psql) was provided for interactive SQL queries, which used GNU Readline.
- A new front-end library, libpgtcl, supported Tcl-based clients. A sample shell, pgtclsh, provided new Tcl commands to interface Tcl programs with the Postgres95 server.
- The large-object interface was overhauled. The inversion large objects were the only mechanism for storing large objects. (The inversion file system was removed.)
- The instance-level rule system was removed. Rules were still available as rewrite rules.
- A short tutorial introducing regular SQL features as well as those of Postgres95 was distributed with the source code
- GNU make (instead of BSD make) was used for the build. Also, Postgres95 could be compiled with an unpatched GCC (data alignment of doubles was fixed).

PostgreSQL

By 1996, it became clear that the name "Postgres95" would not stand the test of time. We chose a new name, PostgreSQL, to reflect the relationship between the original POSTGRES and the more recent versions with SQL capability. At the same time, we set the version numbering to start at 6.0, putting the numbers back into the sequence originally begun by the Berkeley POSTGRES project.
The emphasis during development of Postgres95 was on identifying and understanding existing problems in the server code. With PostgreSQL, the emphasis has shifted to augmenting features and capabilities, although work continues in all areas.

Submitted to Birla Institute of Technology, Mesra, Ranchi (Jaipur Campus)

PostgreSQL: Advantages

PostgreSQL offers many advantages for your company or business over other database systems.

Immunity to over-deployment

Over-deployment is what some proprietary database vendors regard as their #1 licence compliance problem. With PostgreSQL, no-one can sue you for breaking licensing agreements, as there is no associated licensing cost for the software.
This has several additional advantages:

- More profitable business models with wide-scale deployment.
- No possibility of being audited for license compliance at any stage.
- Flexibility to do concept research and trial deployments without needing to include additional licensing costs.

Better support than the proprietary vendors

In addition to our strong support offerings, we have a vibrant community of PostgreSQL professionals and enthusiasts that your staff can draw upon and contribute to.

Significant saving on staffing costs

Our software has been designed and created to have much lower maintenance and tuning requirements than the leading proprietary databases, yet still retain all of the features, stability, and performance.
In addition to this our training programs are generally regarded as being far more cost effective, manageable, and practical in the real world than that of the leading proprietary database vendors.

Legendary reliability and stability

Unlike many proprietary databases, it is extremely common for companies to report that PostgreSQL has never, ever crashed for them in several years of high activity operation. Not even once. It just works.

Extensible

The source code is available to all at no charge. If your staff have a need to customise or extend PostgreSQL in any way then they are able to do so with a minimum of effort, and with no attached costs. This is complemented by the community of PostgreSQL professionals and enthusiasts around the globe that also actively extend PostgreSQL on a daily basis.

Submitted to Birla Institute of Technology, Mesra Ranchi (Jaipur Campus)

Cross platform

PostgreSQL is available for almost every brand of Unix (34 platforms with the latest stable release), and Windows compatibility is available via the Cygwin framework. Native Windows compatibility is also available with version 8.0 and above.

Designed for high volume environments

We use a multiple row data storage strategy called MVCC to make PostgreSQL extremely responsive in high volume environments. The leading proprietary database vendor uses this technology as well, for the same reasons.

GUI database design and administration tools

Several high quality GUI tools exist to both administer the database (pgAdmin, pgAccess) and do database design (Tora, Data Architect).

Submitted to Birla Institute of Technology, Mesra, Ranchi (Jaipur Campus)

Technical Features

- Fully ACID compliant.
- ANSI SQL compliant.
- Referential Integrity.
- Replication (non-commercial and commercial solutions) allowing the duplication of the master database to multiple slave machines.
- Native interfaces for ODBC, JDBC, C, C++, PHP, Perl, TCL, ECPG, Python, and Ruby.
- Rules.
- Views.
- Triggers.
- Unicode.
- Sequences.
- Inheritance.
- Outer Joins.
- Sub-selects.
- An open API.
- Stored Procedures.
- Native SSL support.
- Procedural languages.
- Hot stand-by (commercial solutions).
- Better than row-level locking.
- Functional and Partial indexes.
- Native Kerberos authentication.

- Support for UNION, UNION ALL and EXCEPT queries.
- Loadable extensions offering SHA1, MD5, XML, and other functionality.
- Tools for generating portable SQL to share with other SQL-compliant systems.
- Extensible data type system providing for custom, user-defined datatypes and rapid development of new datatypes.
- Cross-database compatibility functions for easing the transition from other, less SQL-compliant RDBMS.

Installation

PostgreSQL is made available through Red Hat Linux Support. For installing Postgres Server the rpm files must be installed. The list of rpm files is give below:

Name	Summary
rh-postgresql-7.3.8-2	PostgreSQL client programs and libraries
rh-postgresql-contrib-7.3.8-2	Contributed source and binaries distributed with PostgreSQL
rh-postgresql-devel-7.3.8-2	PostgreSQL development header files and libraries
rh-postgresql-docs-7.3.8-2	Extra documentation for PostgreSQL
rh-postgresql-jdbc-7.3.8-2	Files needed for Java programs to access a PostgreSQL database.
rh-postgresql-libs-7.3.8-2	The shared libraries required for any PostgreSQL clients
rh-postgresql-pl-7.3.8-2	The PL procedural languages for PostgreSQL
rh-postgresql-server-7.3.8-2	The programs needed to create and run a PostgreSQL server
rh-postgresql-tcl-7.3.8-2	A Tcl client library for PostgreSQL
rh-postgresql-test-7.3.8-2	The test suite distributed with PostgreSQL

Table 1: List and Descriptions Of Postgres rpm's

Submitted to Birla Institute of Technology, Mesra, Ranchi (Jaipur Campus)

The steps for PostgreSQL installation is given below:

1. Copy all the rpm in /tmp directory
2. login as root
3. Check if postgres is already installed
   rpm -qa|grep postgres
4. If filenames are shown by the above command then remove the installed files by the following command:
   rpm -e < FILE_NAMES_LISTED_BY_ABOVE_COMMAND

   While removing the files the filenames should be specified without

suffixing the version. For example rpm –e rh-postgresql-libs.

5. To install PostgreSQL run the foolowing command one by one:
    a. rpm -ivh rh-postgresql-libs-7.3.8-2.i386.rpm
    b. rpm -ivh rh-postgresql-devel-7.3.8-2.i386.rpm
    c. rpm -ivh rh-postgresql-tcl-7.3.8-2.i386.rpm
    d. rpm -ivh rh-postgresql-7.3.8-2.i386.rpm
    e. rpm -ivh rh-postgresql-contrib-7.3.8-2.i386.rpm
    f. rpm -ivh rh-postgresql-server-7.3.8-2.i386.rpm
    g. rpm -ivh rh-postgresql-docs-7.3.8-2.i386.rpm
    h. rpm -ivh rh-postgresql-jdbc-7.3.8-2.i386.rpm
    i. rpm -ivh rh-postgresql-pl-7.3.8-2.i386.rpm
    j. rpm -ivh rh-postgresql-test-7.3.8-2.i386.rpm
    k. rpm -ivh postgresql-odbc-7.3.8-2.i386.rpm
    l. rpm -ivh rh-postgresql-python-7.3.8-2.i386.rpm

    If any dependencies are found while installing or removing any file then
    --nodeps option should be used with the above commands.

Submitted to Birla Institute of Technology, Mesra, Ranchi (Jaipur Campus)

4.1. Starting PostgreSQL from scratch

1) Log in as any Operating System user.

This user will be the database super user and it can't be the Operating System "root" user. Typically this user is called as 'postgres' or 'pgsql'. But it can be any other user as well.

Linux RH8 Note: su - postgres

2) Initialise the database template.

Export the environment variable "PGDATA", pointing to the directory where the database is to be located, then call "initdb" to create an initial database template.

e.g.
$ export PGDATA=/mnt1/dbs/postgresql
$ initdb

This will create a template database in /mnt1/dbs/postgresql. For all practical purposes, consider this directory opaque unless you know what you are doing.
When initdb is finished, it will tell you how to start the PostgreSQL server as the last part of the messages it gives.

Success. You can now start the database server using:

```
/usr/bin/postmaster -D /mnt1/dbs/postgresql
```

or

```
/usr/bin/pg_ctl -D /mnt1/dbs/postgresql -l logfile start
```

So you should now start the database server as indicated by this last message. Using pg_ctl is the simplest way of doing that.

If you do not want to set the data directory every time, exporting PGDATA variable from .profile of postgresql user is a common practice.

3) Create a database to start with

$ createdb test
CREATE DATABASE
$

Submitted to Birla Institute of Technology, Mesra, Ranchi (Jaipur Campus)

4) Start using it with PostgreSQL terminal client "psql"

```
$ psql test
Welcome to psql, the PostgreSQL interactive terminal.
Type: \copyright for distribution terms
      \h for help with SQL commands
      \? for help on internal slash commands
      \g or terminate with semicolon to execute query
      \q to quit

test=#
```

At this point, you can use all the commands shown below.

Submitted to Birla Institute of Technology, Mesra, Ranchi (Jaipur Campus)

4.2. Some useful tips

1) Shutting down the database

- Log in as database super user. It's the OS user who ran initdb etc. in above steps
- Use pg_ctl to stop the database

$ pg_ctl -D mnt1/dbs/postgresql stop

2) Start PostgreSQL with networking

PostgreSQL by default does not listen on the network. To make it do so, you need to pass the '-i' option to it. The pg_ctl command will be modified to look like:

$ /usr/bin/pg_ctl -D /mnt1/dbs/postgresql -l logfile -o -i start

3) Tuning PostgreSQL

Many paramters that PostgreSQL uses can be modified using configuration file "$PGDATA/postgresql.conf". The configuration file is well documented. Read the administrators guide as well.

If you are looking for any heavy duty work, keep in mind that the PostgreSQL defaults are very conservative and are not meant for any heavy duty work. Please tune the system before you put any load on it.

4) Users in PostgreSQL

You can create databases and users in PostgreSQL. Unlike some other RDBMS though, users in PostgreSQL are global i.e. same username/password combination can be used to connect to any database in the system. Rights granted to any users can be tuned, but there is nothing like a user for a particular database.

Also the PostgreSQL super user is also a database user by default. It is advised that after you create your first database, change the password of the PostgreSQL super user by using the 'alter user' command.

Authentication options for PostgreSQL can be set in $PGDATA/pg_hba.conf. The file is self explanatory.

Submitted to Birla Institute of Technology, Mesra, Ranchi (Jaipur Campus)

5) Making changes to PostgreSQL options effective

If you change PostgreSQL options, they will take effect the next time you restart it. If you can not take down the system, look at reload option of pg_ctl. However not all settings can be reloaded on the fly. e.g. settings like shared buffers need a postmaster restart

Submitted to Birla Institute of Technology, Mesra, Ranchi (Jaipur Campus)

4.3. PostgreSQL Commands Guide

ABORT — Rolls back changes made during a transaction block.
ALTER GROUP — Modifies the structure of a user group.
ALTER TABLE — Modifies table, row, and column attributes.
ALTER USER — Modifies user account properties and permissions.
BEGIN — Starts a chained-mode transaction block.
CLOSE — Closes a previously defined cursor object.
CLUSTER — Provides the backend server with clustering information about a table.
COMMENT — Adds a comment to an object within the database.
COMMIT — Ends the current transaction block and finalizes changes made within it.
COPY — Copies data between files and tables
CREATE AGGREGATE — Defines a new aggregate function within the database.
CREATE CONSTRAINT TRIGGER — Creates a trigger for use with a constraint.
CREATE DATABASE — Creates a new database on the system.
CREATE FUNCTION — Defines a new function within the database.
CREATE GROUP — Creates a new user group within the database.
CREATE INDEX — Constructs an index on a table.
CREATE LANGUAGE — Defines a new language to be used by functions.
CREATE OPERATOR — Defines a new operator within the database.
CREATE RULE — Defines a new rule on a table.
CREATE SEQUENCE — Creates a new sequence number generator.
CREATE TABLE — Creates a new table.
CREATE TABLE AS — Creates a new table built from data retrieved by a SELECT.
CREATE TRIGGER — Creates a new trigger.
CREATE TYPE — Defines a new data type for use in the database.
CREATE USER — Creates a new user account.
CREATE VIEW — Creates a view on a table.
createdb — Creates a new database from the command line.
createlang — Defines a new programming language for use in writing database functions.
createuser — Adds a new user account to a database.
SQL_CURRENT_DATE — Returns the current date.
SQL_CURRENT_TIME — Returns the current time.

Submitted to Birla Institute of Technology, Mesra, Ranchi (Jaipur Campus)

SQL_CURRENT_TIMESTAMP — Returns the current date and time.
SQL_CURRENT_USER — Returns the current database username.
DECLARE — Defines a new cursor.
DELETE — Removes rows from a table.
DROP AGGREGATE — Removes the definition of an aggregate function from the database.
DROP DATABASE — Removes a database from the system.
DROP FUNCTION — Removes a user-defined function.
DROP GROUP — Removes a user group from the database.
DROP INDEX — Removes an index from a database.
DROP LANGUAGE — Removes the definition of a procedural language.
DROP OPERATOR — Removes an operator from the database.
DROP RULE — Removes a rule from a database.
DROP SEQUENCE — Removes a sequence from a database.
DROP TABLE — Removes a table from a database
DROP TRIGGER — Removes the definition of a trigger from a database.
DROP TYPE — Removes a type from the system catalogs.
DROP USER — Removes a user account from a database.
DROP VIEW — Removes an existing view from a database.
dropdb — Removes a database from the system.
droplang — Removes the definition of a procedural language from a database.
dropuser — Removes a user account from a database.
ecpg — The embedded SQL C preprocessor.
END — Ends the current transaction block and finalizes its modifications.
EXPLAIN — Shows the statement execution plan for a supplied query.
FETCH — Retrieves rows from a cursor.
GRANT — Grants access privileges to a user, a group, or to all users in the database.
initdb — Creates a new database cluster.
initlocation — Create a secondary PostgreSQL database storage area
INSERT — Inserts new rows into a table.
ipcclean — Cleans shared memory resources left behind by aborted backend processes.
LISTEN — Listen for a notification event.
LOAD — Dynamically loads object files into a database.
LOCK — Explicitly locks a specified table within the current transaction.
MOVE — Repositions the cursor to another row.

Submitted to Birla Institute of Technology, Mesra, Ranchi (Jaipur Campus)

NOTIFY — Signals all backends that are listening for the specified notify event.
pg_dump — Exports a database to a script file.
pg_dumpall — Exports all databases on the system to a script file.
pg_ctl — Starts, stops, and restarts postmaster.
pgtclsh — The TCL database client.
pgtksh — A graphical TCL/TK database client.
postgres — Runs a single-user backend process.
postmaster — Executes the PostgreSQL multi-user backend process.
psql — The PostgreSQL interactive terminal.
REINDEX — Rebuilds indices on tables.
RESET — Restores run-time variables to their default settings.
REVOKE — Revokes access privileges from a user, a group, or all users.
ROLLBACK — Aborts the current transaction block and abandons any modifications it would have made.
SELECT — Retrieves rows from a table or view.
SELECT INTO — Construct a new table from the results of a SELECT.
SET — Set run-time variables.
SET CONSTRAINTS — Sets the constraint mode for the current transaction block.
SET TRANSACTION — Sets the transaction isolation level for the current transaction block.
SHOW — Displays the values of run-time variables.
TRUNCATE — Empties the contents of a table.
UNLISTEN — Stops the backend process from listening for a notification event.
UPDATE — Modifies the values of column data within a table.
VACUUM — Cleans and analyzes a database.
vacuumdb — Cleans and analyzes a database.

Submitted to Birla Institute of Technology, Mesra, Ranchi (Jaipur Campus)

\? -- Help on PSQL Commands

\a toggle between unaligned and aligned output mode
\c [onnect] [DBNAME|- [USER]]
Connect to new database (currently "TESTDB")
\C [STRING] set table title, or unset if none
\cd [DIR] change the current working directory
\copy ... perform SQL COPY with data stream to the client host
\copyright show PostgreSQL usage and distribution terms
\d [NAME] describe table, index, sequence, or view
\d{t|i|s|v|S} [PATTERN] (add "+" for more detail)

list tables/indexes/sequences/views/system tables
\da [PATTERN] list aggregate functions
\dd [PATTERN] show comment for object
\dD [PATTERN] list domains
\df [PATTERN] list functions (add "+" for more detail)
\do [NAME] list operators
\dl list large objects, same as \lo_list
\dp [PATTERN] list table access privileges
\dT [PATTERN] list data types (add "+" for more detail)
\du [PATTERN] list users
\e [FILE] edit the query buffer (or file) with external editor
\echo [STRING] write string to standard output
\encoding [ENCODING] show or set client encoding

\f [STRING] show or set field separator for unaligned query output
\g [FILE] send query buffer to server (and results to file or |pipe)
\h [NAME] help on syntax of SQL commands, * for all commands
\H toggle HTML output mode (currently off)
\i FILE execute commands from file
\l list all databases
\lo_export, \lo_import, \lo_list, \lo_unlink
large object operations
\o FILE send all query results to file or |pipe
\p show the contents of the query buffer
\pset NAME [VALUE] set table output option
(NAME:=
{format|border|expanded|fieldsep|null|recordsep|
tuples_only|title|tableattr|pager})
\q quit psql
\qecho [STRING] write string to query output stream (see \o)
\r reset (clear) the query buffer
\s [FILE] display history or save it to file
\set [NAME [VALUE]] set internal variable, or list all if no parameters
\t show only rows (currently off)
\T [STRING] set HTML <table> tag attributes, or unset if none
\timing toggle timing of commands (currently off)
\unset NAME unset (delete) internal variable

\w [FILE] write query buffer to file
\x toggle expanded output (currently off)
\z [PATTERN] list table access privileges (same as \dp)

Submitted to Birla Institute of Technology, Mesra, Ranchi (Jaipur Campus)

\! [COMMAND] execute command in shell or start interactive shell

Submitted to Birla Institute of Technology, Mesra, Ranchi (Jaipur Campus)

5. References & Sources:

5.1. Books:

- Java Complete Reference……Herbert Schildt
- Java Application Development……STG

5.2. Websites:

- *http://www.postgresql.org/*
  (The Site provides technical stuff related to PostgreSQL Database).
- *http://www.devx.com/*
  (The site provides good tutorial on XML Parsing)
- *http://www.openldap.org/*
  (Open source LDAP software including servers, clients, and SDKs.)
- *http://java.sun.com/*
  (Official site of the Sun MicroSystems)

Submitted to Birla Institute of Technology, Mesra, Ranchi (Jaipur Campus)