UNIT III

Web Publishing with Java

Why Java?

* Why use Java for Distributed Internet and Intranet Applications?
* Why use Java for Database Connectivity?
* Why use Java When you could use ActiveX?
* Why not use Java When you could use ActiveX?

Why Java?

Java has created a huge amount of excitement within the internet community. Java is a standard, consistent, and predictable way of presenting or manipulating data. Platform independence alone provides enough incentive for many people to choose Java over proprietary languages and tools.

The mocha:

* **Platform Independence**

The concept of Write-once-run-anywhere (known as the Platform independent) is one of the important key feature of java language that makes java as the most powerful language. Not even a single language is idle to this feature but java is more closer to this feature. The programs written on one platform can run on any platform provided the platform must have the JVM.

* **Simple**There are various features that makes the java as a simple language. Programs are easy to write and debug because java does not use the pointers explicitly. It is much harder to write the [java programs](http://www.roseindia.net/java/java-introduction/java-features.shtml) that can crash the system but we can not say about the other programming languages. Java provides the bug free system due to the strong memory management. It also has the automatic memory allocation and deallocation system.
* **Object-Oriented**To be an Object Oriented language, any language must follow at least the four characteristics.
* Inheritance           :       It is the process of creating the new classes and using the behavior of the existing classes by extending them just to reuse  the existing code and adding the additional features as needed.
* Encapsulation:      :       It is the mechanism of combining the information and providing the abstraction.
* Polymorphism:     :       As the name suggest one name multiple form, Polymorphism is the way of providing the different functionality by the functions  having the same name based on the signatures of the methods.
* Dynamic binding  :       Sometimes we don't have the knowledge of objects about their specific types while writing our code. It is the way of providing the maximum functionality to a program about the specific type at runtime.
* **Robust**Java has the strong memory allocation and automatic garbage collection mechanism. It provides the powerful exception handling and type checking mechanism as compare to other programming languages. Compiler checks the program whether there any error and interpreter checks any run time error and makes the system secure from crash. All of the above features makes the java language robust.
* **Distributed**The widely used protocols like HTTP and [FTP](http://www.roseindia.net/java/java-introduction/java-features.shtml) are developed in java. Internet programmers can call functions on these protocols and can get access the files from any remote machine on the internet rather than writing codes on their local system.
* **Portable**The feature Write-once-run-anywhere  makes the java language portable provided that the system must have interpreter for the JVM. Java also have the standard data size irrespective of operating system or the [processor](http://www.roseindia.net/java/java-introduction/java-features.shtml). These features makes the java as a portable language.
* **Dynamic**While executing the java program the user can get the required files dynamically from a local drive or from a computer thousands of miles away from the user just by connecting with the Internet.
* **Secure**Java does not use memory pointers explicitly. All the programs in java are run under an area known as the sand box. Security manager determines the accessibility options of a class like reading and writing a file to the local disk. Java uses the public key encryption system to allow the java applications to transmit over the [internet](http://www.roseindia.net/java/java-introduction/java-features.shtml) in the secure encrypted form. The bytecode Verifier checks the classes after loading.
* **Performance**Java uses native code usage, and lightweight process called  threads. In the beginning interpretation of bytecode resulted the [performance](http://www.roseindia.net/java/java-introduction/java-features.shtml) slow but the advance version of JVM uses the adaptive and just in time compilation technique that improves the performance.
* **Multithreaded**  
  As we all know several features of Java like Secure, Robust, Portable, dynamic etc; you will be more delighted to know another feature of Java which is **Multithreaded.**Java is also a Multithreaded programming language. Multithreading means a single program having different threads executing independently at the same time. Multiple threads execute instructions according to the program code in a process or a program. Multithreading works the similar way as multiple processes run on one computer.    
  Multithreading programming is a very interesting concept in Java. In multithreaded programs not even a single thread disturbs the execution of other thread. Threads are obtained from the pool of available ready to run threads and they run on the system CPUs. This is how Multithreading works in Java which you will soon come to know in details in later chapters.
* **Interpreted**We all know that Java is an interpreted language as well. With an interpreted language such as Java, programs run directly from the source code.   
  The interpreter program reads the source code and translates it on the fly into computations. Thus, Java as an interpreted language depends on an interpreter program.  
  The versatility of being **platform independent** makes Java to outshine from other languages. The source code to be written and distributed is platform independent.    
  Another advantage of Java as an interpreted language is its error debugging quality. Due to this any error occurring in the program gets traced. This is how it is different to work with Java.
* **Architecture-Neutral**The term architectural neutral seems to be weird, but yes Java is an architectural neutral language as well. The growing popularity of networks makes developers think distributed. In the world of network it is essential that the applications must be able to migrate easily to different computer systems. Not only to computer systems but to a wide variety of hardware architecture and Operating system architectures as well.  The Java compiler does this by generating byte code instructions, to be easily interpreted on any machine and to be easily translated into native machine code on the fly.The compiler generates an architecture-neutral object file format to enable a Java application to execute anywhere on the network and then the compiled code is executed on many processors, given the presence of the Java runtime system.Hence Java was designed to support applications on network. This feature of Java has thrived the programming language.

**Why use Java for Distributed Internet and Intranet Applications?**

Some of the reasons for using java on the internet include:

* State Awareness

Because HTTP is a stateless protocol, keeping track of a chain of request requires the passing of a unique identifier with each request.

This additional overhead is avoided by implementing an applet or servlet in Java.

In the applet case, state information can be maintained on the client side, while in the servlet case, state information can be maintained on the server side.

Java offers a third, distributed approach – using RMI or CORBA – which eliminates the need to distinguish between the server and the client, and state information exists on both side.

* Reuse

The availability of servlet API allows java functionality developed for the client side to be effectively reused on the server side.

* Input Validation

Passing a user’s input on the client side often makes more sense than passing the input back to the server for validation and re-prompting the user to correct an individual input.

For example, rather than prompting for a value in an html form input, that value could be entered into a form in an applet that validates the input amount as the user enters it on the client side.

* Efficiency

Applets can display data without information about how the data should be formatted. For this reason, applets can sometimes perform tasks more efficiently than html.

Servlets, provide an even greater performance advantage when used in place of the traditional CGI mechanism.

* Agents

Software agents roam throughout computer networks delivering messages or gathering information.

Because Java runs on any platform, agents implemented in Java can roam further afield than agents implemented in other languages.

* Commerce Applications

The Java language supports the level of security required by commerce applications and provides an excellent dynamic infrastructure on which to build the distributed commerce application of the future.

* Concurrent Applications

Java provides a highly integrated, multithreaded programming environment that is relatively easy to use.

Distributed internet applications usually must do more than one thing at the same time.

**Why use Java for Database Connectivity?**

* Java provides an excellent tool for database connectivity.
* Almost any application, whether on the client or server side, needs to access a database at some point.
* Java abstracts a database into a series of classes and interfaces, that effectively hide the implementation of the database drivers.
* The specification is called Java Database Connectivity (JDBC), and drivers already exist for most applications.
* Java cod written to access a database runs on any platform and with any database.

**Why use Java When you could use ActiveX?**

* Java runs on any machine with Java Virtual Machine or a java – enabled browser. ActveX runs on any 32-bit windows machine, in IE, in a Netscape Navigator with an Active-X enabling plug-in on a 32-bit Windows machine, or on an operating system with a DCOM implementation.
* ActiveX controls are larger than compiled java byte code.
* ActiveX is less secure than Java in an untrusted environment.
* Finally, Java’s component model, JavaBeans, is gaining industry- wide acceptance.

**Why not use Java When you could use ActiveX?**

* There are many reasons, to use Java on the internet and within a corporate intranet – especially if the intranet is sufficiently heterogeneous mixture of platforms.
* If the intranet consists of all Microsoft platforms, then it is better to use ActiveX.

The Java Language

* Classes and Objects
* Creating Classes and Objects
* A java Application
* Instance and Class variables and Methods
* Packages
* Overloading Constructors
* Inheritance and Overriding Methods
* Abstract Methods and and Interfaces
* Access modifiers
* Streams, Sockets, and sending the Email Message

Classes and Objects

In [object-oriented programming](http://en.wikipedia.org/wiki/Object-oriented_programming), a **class** is a construct that is used as a [blueprint](http://en.wikipedia.org/wiki/Blueprint) (or [template](http://en.wikipedia.org/wiki/Template)) to create [objects](http://en.wikipedia.org/wiki/Object_(object-oriented_programming)) of that class. This blueprint describes the state and behavior that the objects of the class all share. An object of a given class is called an [instance](http://en.wikipedia.org/wiki/Instantiation_(computer_science)) of the class. The class that contains (and was used to create) that instance can be considered as the [type](http://en.wikipedia.org/wiki/Data_type) of that object, e.g. an object instance of the "Fruit" class would be of the type "Fruit".

Creating classes and Objects

**Declaring Classes**

class *MyClass* {

//field, constructor, and method declarations

}

This is a *class declaration*. The *class body* (the area between the braces) contains all the code that provides for the life cycle of the objects created from the class: constructors for initializing new objects, declarations for the fields that provide the state of the class and its objects, and methods to implement the behavior of the class and its objects.

In general, class declarations can include these components, in order:

1. Modifiers such as *public*, *private*, and a number of others that you will encounter later.
2. The class name, with the initial letter capitalized by convention.
3. The name of the class's parent (superclass), if any, preceded by the keyword *extends*. A class can only *extend* (subclass) one parent.
4. A comma-separated list of interfaces implemented by the class, if any, preceded by the keyword *implements*. A class can *implement* more than one interface.
5. The class body, surrounded by braces, {}.

**Declaring Member Variables**

There are several kinds of variables:

* Member variables in a class—these are called *fields*.
* Variables in a method or block of code—these are called *local variables*.
* Variables in method declarations—these are called *parameters*.

The Bicycle class uses the following lines of code to define its fields:

public int cadence;

public int gear;

public int speed;

Field declarations are composed of three components, in order:

1. Zero or more modifiers, such as public or private.
2. The field's type.
3. The field's name.

**Defining Methods**

Here is an example of a typical method declaration:

public double calculateAnswer(double wingSpan, int numberOfEngines, double length, double grossTons) {

//do the calculation here

}

The only required elements of a method declaration are the method's return type, name, a pair of parentheses, (), and a body between braces, {}.

More generally, method declarations have six components, in order:

1. Modifiers—such as public, private, and others you will learn about later.
2. The return type—the data type of the value returned by the method, or void if the method does not return a value.
3. The method name—the rules for field names apply to method names as well, but the convention is a little different.
4. The parameter list in parenthesis—a comma-delimited list of input parameters, preceded by their data types, enclosed by parentheses, (). If there are no parameters, you must use empty parentheses.
5. An exception list—to be discussed later.
6. The method body, enclosed between braces—the method's code, including the declaration of local variables, goes here.

**Providing Constructors for Your Classes**

A class contains constructors that are invoked to create objects from the class blueprint. Constructor declarations look like method declarations—except that they use the name of the class and have no return type. For example, Bicycle has one constructor:

public Bicycle(int startCadence, int startSpeed, int startGear) {

gear = startGear;

cadence = startCadence;

speed = startSpeed;

}

To create a new Bicycle object called myBike, a constructor is called by the new operator:

Bicycle myBike = new Bicycle(30, 0, 8);

new Bicycle(30, 0, 8) creates space in memory for the object and initializes its fields.

**Creating Objects**

As you know, a class provides the blueprint for objects; you create an object from a class. Each of the following statements taken from the [CreateObjectDemo](http://java.sun.com/docs/books/tutorial/java/javaOO/examples/CreateObjectDemo.java) program creates an object and assigns it to a variable:

**Point originOne** = new Point(23, 94);

**Rectangle rectOne** = new Rectangle(originOne, 100, 200);

**Rectangle rectTwo** = new Rectangle(50, 100);

The first line creates an object of the [Point](http://java.sun.com/docs/books/tutorial/java/javaOO/examples/Point.java) class, and the second and third lines each create an object of the [Rectangle](http://java.sun.com/docs/books/tutorial/java/javaOO/examples/Rectangle.java) class.

Each of these statements has three parts (discussed in detail below):

1. **Declaration**: The code set in **bold** are all variable declarations that associate a variable name with an object type.
2. **Instantiation**: The new keyword is a Java operator that creates the object.
3. **Initialization**: The new operator is followed by a call to a constructor, which initializes the new object.
4. **Using Objects**
5. Once you've created an object, you probably want to use it for something. You may need to use the value of one of its fields, change one of its fields, or call one of its methods to perform an action.
6. **Referencing an Object's Fields**

Object fields are accessed by their name. Code that is outside the object's class must use an object reference or expression, followed by the dot (.) operator, followed by a simple field name, as in:

objectReference.fieldName

Instance and Class Variables and Methods

The variables and methods that are associated with a particular object are called instance variables and methods.

When a method or variable is declared as static, it indicates to the compiler that the member belongs to the class rather than the instance. These methods and variables are called as class methods and variables, because these belong only to class.

Instance variables and methods are accessed by referring to an object, but class variables and methods are accessed by referring to a class.

Example:

public class Bicycle{

private int cadence;

private int gear;

private int speed;

**// add an instance variable for the object ID**

private int id;

**// add a class variable for the number of Bicycle objects instantiated**

private **static** int numberOfBicycles = 0;

**......**

**}**

**Packages**

A **Java package** is a mechanism for organizing [Java](http://en.wikipedia.org/wiki/Java_(programming_language)) [classes](http://en.wikipedia.org/wiki/Class_(computer_science)) into [namespaces](http://en.wikipedia.org/wiki/Namespace_(computer_science)) similar to the modules of [Modula](http://en.wikipedia.org/wiki/Modula). Java packages can be stored in compressed files called [JAR files](http://en.wikipedia.org/wiki/JAR_file), allowing classes to download faster as a group rather than one at a time. Programmers also typically use packages to organize classes belonging to the same category or providing similar functionality.

* A package provides a unique namespace for the types it contains.
* Classes in the same package can access each other's package-access members.

Overloading Constructors

Constructors can be overloaded just as regular methods.

Constructors can be overloaded just as regular methods. Here is an example of a class with an overloaded constructor:

Code:

public class date

{

int year;

int month;

int day;

date(int year)

{

this.year = year;

month = 1;

day = 1;

}

date(int year/int month,int day)

{

this.year = year;

this.month = month;

this.day = day;

}

}

|  |  |
| --- | --- |
| You could create an object of that class with statements like either of the following:  Quote:   |  | | --- | | date a = new date(2010]; date b = new date(2010,5,15); |   Naturally, the first statement will call the first constructor, because it has just one parameter. The second statement has three parameters, so it will call the second constructor. |

**Using the this Keyword**

Within an instance method or a constructor, this is a reference to the *current object* — the object whose method or constructor is being called. You can refer to any member of the current object from within an instance method or a constructor by using this.

**Using this with a Field**

The most common reason for using the this keyword is because a field is shadowed by a method or constructor parameter.

For example, the Point class was written like this

public class Point {

public int x = 0;

public int y = 0;

**//constructor**

**public Point(int a, int b) {**

**x = a;**

**y = b;**

**}**

}

but it could have been written like this:

public class Point {

public int x = 0;

public int y = 0;

**//constructor**

**public Point(int x, int y) {**

**this.x = x;**

**this.y = y;**

**}**

}

Each argument to the constructor shadows one of the object's fields — inside the constructor **x** is a local copy of the constructor's first argument. To refer to the Point field **x**, the constructor must use **this.x**.

**Using this with a Constructor**

From within a constructor, you can also use the this keyword to call another constructor in the same class. Doing so is called an *explicit constructor invocation*. Here's another Rectangle class, with a different implementation from the one in the [Objects](http://java.sun.com/docs/books/tutorial/java/javaOO/objects.html) section.

public class Rectangle {

private int x, y;

private int width, height;

public Rectangle() {

**this(0, 0, 0, 0);**

}

public Rectangle(int width, int height) {

**this(0, 0, width, height);**

}

public Rectangle(int x, int y, int width, int height) {

this.x = x;

this.y = y;

this.width = width;

this.height = height;

}

...

}

This class contains a set of constructors. Each constructor initializes some or all of the rectangle's member variables. The constructors provide a default value for any member variable whose initial value is not provided by an argument. For example, the no-argument constructor calls the four-argument constructor with four 0 values and the two-argument constructor calls the four-argument constructor with two 0 values. As before, the compiler determines which constructor to call, based on the number and the type of arguments.

If present, the invocation of another constructor must be the first line in the constructor.

**Inheritance and Overriding Methods**

A class that is derived from another class is called a *subclass* (also a *derived class*, *extended class*, or *child class*). The class from which the subclass is derived is called a *superclass* (also a *base class* or a *parent class*).

The idea of inheritance is simple but powerful: When you want to create a new class and there is already a class that includes some of the code that you want, you can derive your new class from the existing class. In doing this, you can reuse the fields and methods of the existing class without having to write (and debug!) them yourself.

A subclass inherits all the *members* (fields, methods, and nested classes

**An Example of Inheritance**

Here is the sample code for a possible implementation of a Bicycle class that was presented in the Classes and Objects lesson:

public class Bicycle {

// **the Bicycle class has three *fields***

public int cadence;

public int gear;

public int speed;

// **the Bicycle class has one *constructor***

public Bicycle(int startCadence, int startSpeed, int startGear) {

gear = startGear;

cadence = startCadence;

speed = startSpeed;

}

// **the Bicycle class has four *methods***

public void setCadence(int newValue) {

cadence = newValue;

}

public void setGear(int newValue) {

gear = newValue;

}

public void applyBrake(int decrement) {

speed -= decrement;

}

public void speedUp(int increment) {

speed += increment;

}

}

A class declaration for a MountainBike class that is a subclass of Bicycle might look like this:

public class MountainBike extends Bicycle {

// **the MountainBike subclass adds one *field***

public int seatHeight;

// **the MountainBike subclass has one *constructor***

public MountainBike(int startHeight, int startCadence, int startSpeed, int startGear) {

super(startCadence, startSpeed, startGear);

seatHeight = startHeight;

}

// **the MountainBike subclass adds one *method***

public void setHeight(int newValue) {

seatHeight = newValue;

}

}

MountainBike inherits all the fields and methods of Bicycle and adds the field seatHeight and a method to set it. Except for the constructor, it is as if you had written a new MountainBike class entirely from scratch, with four fields and five methods. However, you didn't have to do all the work. This would be especially valuable if the methods in the Bicycle class were complex and had taken substantial time to debug.

An instance method in a subclass with the same signature (name, plus the number and the type of its parameters) and return type as an instance method in the superclass *overrides* the superclass's method.

Abstract Classes and Abstract methods

**Abstract Methods and Classes**

An *abstract class* is a class that is declared abstract—it may or may not include abstract methods. Abstract classes cannot be instantiated, but they can be subclassed.

An *abstract method* is a method that is declared without an implementation (without braces, and followed by a semicolon), like this:

abstract void moveTo(double deltaX, double deltaY);

If a class includes abstract methods, the class itself *must* be declared abstract, as in:

public abstract class GraphicObject {

// declare fields

// declare non-abstract methods

abstract void draw();

}

When an abstract class is subclassed, the subclass usually provides implementations for all of the abstract methods in its parent class. However, if it does not, the subclass must also be declared abstract.

**Note:** All of the methods in an *interface* (see the [Interfaces](http://java.sun.com/docs/books/tutorial/java/IandI/createinterface.html) section) are *implicitly* abstract, so the abstract modifier is not used with interface methods (it could be—it's just not necessary).

**Abstract Classes versus Interfaces**

Unlike interfaces, abstract classes can contain fields that are not static and final, and they can contain implemented methods. Such abstract classes are similar to interfaces, except that they provide a partial implementation, leaving it to subclasses to complete the implementation. If an abstract class contains *only* abstract method declarations, it should be declared as an interface instead.

Multiple interfaces can be implemented by classes anywhere in the class hierarchy, whether or not they are related to one another in any way. Think of Comparable or Cloneable, for example.

By comparison, abstract classes are most commonly subclassed to share pieces of implementation. A single abstract class is subclassed by similar classes that have a lot in common (the implemented parts of the abstract class), but also have some differences (the abstract methods).

**An Abstract Class Example**

In an object-oriented drawing application, you can draw circles, rectangles, lines, Bezier curves, and many other graphic objects. These objects all have certain states (for example: position, orientation, line color, fill color) and behaviors (for example: moveTo, rotate, resize, draw) in common. Some of these states and behaviors are the same for all graphic objects—for example: position, fill color, and moveTo. Others require different implementations—for example, resize or draw. All GraphicObjects must know how to draw or resize themselves; they just differ in how they do it. This is a perfect situation for an abstract superclass. You can take advantage of the similarities and declare all the graphic objects to inherit from the same abstract parent object—for example, GraphicObject, as shown in the following figure.

Classes Rectangle, Line, Bezier, and Circle inherit from GraphicObject

First, you declare an abstract class, GraphicObject, to provide member variables and methods that are wholly shared by all subclasses, such as the current position and the moveTo method. GraphicObject also declares abstract methods for methods, such as draw or resize, that need to be implemented by all subclasses but must be implemented in different ways. The GraphicObject class can look something like this:

abstract class GraphicObject {

int x, y;

...

void moveTo(int newX, int newY) {

...

}

abstract void draw();

abstract void resize();

}

Each non-abstract subclass of GraphicObject, such as Circle and Rectangle, must provide implementations for the draw and resize methods:

class Circle extends GraphicObject {

void draw() {

...

}

void resize() {

...

}

}

class Rectangle extends GraphicObject {

void draw() {

...

}

void resize() {

...

}

}

Interfaces and Implementation

An **interface** in the [Java programming language](http://en.wikipedia.org/wiki/Java_(programming_language)) is an [abstract type](http://en.wikipedia.org/wiki/Abstract_type) that is used to specify an [interface](http://en.wikipedia.org/wiki/Interface_(computer_science)) (in the generic sense of the term) that [classes](http://en.wikipedia.org/wiki/Class_(computer_science)) must implement. Interfaces are declared using the **interface** [keyword](http://en.wikipedia.org/wiki/Java_keywords), and may only contain [method](http://en.wikipedia.org/wiki/Method_(computer_science)) [signatures](http://en.wikipedia.org/wiki/Signature_(computer_science)) and constant declarations (variable declarations that are declared to be both static and final). An interface may never contain method definitions.

As interfaces are implicitly *abstract*, they cannot be directly instantiated except when instantiated by a class that *implements* the said interface. The class must implement all of the methods described in the interface, or be an [abstract class](http://en.wikipedia.org/wiki/Abstract_class). Object references in Java may be specified to be of an interface type; in which case, they must either be [null](http://en.wikipedia.org/wiki/Null_pointer), or be bound to an object that implements the interface.

One benefit of using interfaces is that they simulate [multiple inheritance](http://en.wikipedia.org/wiki/Multiple_inheritance). All classes in Java (other than [java.lang.Object](http://java.sun.com/javase/6/docs/api/java/lang/Object.html), the [root class](http://en.wikipedia.org/wiki/Top_type) of the Java [type system](http://en.wikipedia.org/wiki/Type_system)) must have exactly one [base class](http://en.wikipedia.org/wiki/Base_class); [multiple inheritance](http://en.wikipedia.org/wiki/Multiple_inheritance) of classes is not allowed. Furthermore, a Java class may implement, and an interface may extend, any number of interfaces; however an interface may not implement an interface.

1. interface IntExample{
3. /\*
4. Syntax to declare method in java interface is,
5. <modifier> <return-type> methodName(<optional-parameters>);
6. IMPORTANT : Methods declared in the interface are implicitly public and abstract.
7. \*/
9. public void sayHello();
10. }
11. }
12. /\*
13. Classes are extended while interfaces are implemented.
14. To implement an interface use implements keyword.
15. IMPORTANT : A class can extend only one other class, while it
16. can implement n number of interfaces.
17. \*/
19. public class JavaInterfaceExample implements IntExample{
20. /\*
21. We have to define the method declared in implemented interface,
22. or else we have to declare the implementing class as abstract class.
23. \*/
25. public void sayHello(){
26. System.out.println("Hello Visitor !");
27. }
29. public static void main(String args[]){
30. //create object of the class
31. JavaInterfaceExample javaInterfaceExample = new JavaInterfaceExample();
32. //invoke sayHello(), declared in IntExample interface.
33. javaInterfaceExample.sayHello();
34. }
35. }

Access Modifiers

**1. private  
2. protected  
3. default  
4. public**

public access modifier

Fields, methods and constructors declared public (least restrictive) within a public class are visible to any class in the Java program, whether these classes are in the same package or in another package.

private access modifier

The private (most restrictive) fields or methods cannot be used for classes and Interfaces. It also cannot be used for fields and methods within an interface. Fields, methods or constructors declared private are strictly controlled, which means they cannot be accesses by anywhere outside the enclosing class. A standard design strategy is to make all fields private and provide public getter methods for them.

protected access modifier

The protected fields or methods cannot be used for classes and Interfaces. It also cannot be used for fields and methods within an interface. Fields, methods and constructors declared protected in a superclass can be accessed only by subclasses in other packages. Classes in the same package can also access protected fields, methods and constructors as well, even if they are not a subclass of the protected member’s class.

Default – only classes within the package have access to default attributes and behaviour.

**Introduction to Applet Programming**

* Writing Applets
* Other Applet Class Functionality
* Graphics and Double Bufferring
* Creating a User Interface Using the AWT
* Showing an Html Document

Applet Security

**Introduction**

Applet is java program that can be embedded into HTML pages. Java applets runs on the java enables web browsers such as mozila and internet explorer. Applet is designed to run remotely on the client browser, so there are some restrictions on it. Applet can't access system resources on the local computer. Applets are used to make the web site more dynamic and entertaining.

**Advantages of Applet:**

* Applets are cross platform and can run on Windows, Mac OS and Linux platform
* Applets can work all the version of Java Plugin
* Applets runs in a sandbox, so the user does not need to trust the code, so it can work without security approval
* Applets are supported by most web browsers
* Applets are cached in most web browsers, so will be quick to load when returning to a web page
* User can also have full access to the machine if user allows

Writing Applets

Instead of writing a program that has a main() method as in the case of an application, applets are any class that subclasses the Applet class and override some of its methods.

A simple Applet

import java.awt.\*;

import java.applet.\*;

class HelloWorldApplet extends Applet

{

// Default constructor

public void HelloWorld()

{

// Call parent constructor

super();

}

Overridden paint method

public void paint ( Graphics g )

{

g.setBackground ( Color.white );

g.setColor ( Color.blue );

g.drawString ( "Hello world!", 0, size().height - 5);

}

}

The <Applet> Tag

Here is an example of a simple APPLET tag:

<applet code="MyApplet.class" width=100 height=140></applet>

This tells the viewer or browser to load the applet whose compiled code is in MyApplet.class (in the same directory as the current HTML document), and to set the initial size of the applet to 100 pixels wide and 140 pixels high.

|  |  |  |  |
| --- | --- | --- | --- |
| **Attribute** | **Value** | **Description** | **DTD** |
| code | *URL* | Specifies the file name of a Java applet | TF |
| object | *name* | Specifies a reference to a serialized representation of an applet | TF |

## Optional Attributes

|  |  |  |  |
| --- | --- | --- | --- |
| **Attribute** | **Value** | **Description** | **DTD** |
| align | left right top bottom middle baseline | Specifies the alignment of an applet according to surrounding elements | TF |
| alt | *text* | Specifies an alternate text for an applet | TF |
| archive | *URL* | Specifies the location of an archive file | TF |
| codebase | *URL* | Specifies a relative base URL for applets specified in the code attribute | TF |
| height | *pixels* | Specifies the height of an applet | TF |
| hspace | *pixels* | Defines the horizontal spacing around an applet | TF |
| name | *name* | Defines the name for an applet (to use in scripts) | TF |
| vspace | *pixels* | Defines the vertical spacing around an applet | TF |
| width | *pixels* | Specifies the width of an applet | TF |

The <Object> Tag

The <object> tag replaces <applet> in html 4.0.

In this example, the CLASSID is used to specify the name of the java code to execute.

<Object CLASSID = “java:HelloWorld.start” height=100 width=100>

</Object>

The <Param> tag

<Param> tags can be placed within the <Applet> or new <Object> tag to send additional information to the applet.

Example:

Public class Salutation extends Applet

{

Private String s;

Public void init()

{

s = getParameter(“salutation”);

}

Public void paint(Graphics g)

{

g.drawString(salutation,10,10);

}

}

The Html code:

<Applet code=”Salutation.class” width=200 height=250>

<param name=”salutation” value=”Greetings and Saltutations”> </param>

</Applet>

Applet methods

A list of the applet methods that can be overridden are as follows:

* Init() - This method is called after the web page containing an applet is loaded into a browser.
* Start() - This method is called every time the user comes back to the web page containing the applet.
* Stop () - This method is called every time the user leaves the web page containing the applet.
* Paint () - This method is called any time the applet’s repaint() method is called.
* Update() - This method is called every time the applet is repainted. By default, it erases the entire screen.
* getAppletInfo() - This method is called by the browser when information about the applet is requested by the user.
* getParameterInfo() - This method is called to find out what kind of parameters the applet can handle.

Resources Available to Applets

A number of resources are available to applets.

The methods that provide access to resources are:

AudioClip getAudioClip() - This method can be used to load an audio clip from a URL.

* Image getImage() - This method is used to load images from an URL.
* Void play(URL url) - This methods plays a loaded audio clip.

Methods that provide access to information about an applet are:

* URL getCodeBase() - This method returns the URL from which the applet’s class files were loaded.
* URL getDocumentBase() - This method returns the URL from which the applet’s HTML document was loaded.
* AppletContext getAppletContext() - This method returns an object of type AppletContext. An AppletContext object is useful for loading and viewing web pages.
* showStatus( String msg) - This method displays the msg in the browser’s status bar.
* resize() - This method resizes the applet’s bounds.

Graphics and Double Buffering

Some of the most commonly used methods od Graphics class are:

* void drawstring(String string, int x, int y) - Draws a string to the screen at the x, y coordinates.
* void clearRect(int x, int y, int width, int height) - Clears a rectangle on the screen with its top-left corner located at x,y and the specified width and height.
* void fillRect (int x, int y, int width, int height) - Fills a rectangle on the screen with its top-left corner located at x,y and the specified width and height.
* void drawRect (int x, int y, int width, int height) - Draws a rectangle on the screen with its top-left corner located at x,y and the specified width and height.
* Void dispose() - Frees resources associated with a graphics context.

Example

**import java.awt.\*;**

**import java.applet.\*;**

**public class graph\_ex extends Applet**

**{**

**public void paint (Graphics g)**

**{ setBackground(Color.cyan);**

**g.drawString("Here are a selection of blank shapes.",20,40);**

**g.drawLine(20,40,200,40);**

**g.setColor(Color.blue);**

**g.drawLine(20,50,70,90);**

**g.setColor(Color.red);**

**g.drawRect(100,50,32,55);**

**g.setColor(Color.green);**

**g.drawOval(150,46,60,60);**

**g.setColor(Color.magenta);**

**g.drawArc(230,50,65,50,30,270);**

**g.setColor(Color.black);**

**g.drawString("Here are the filled equivalents.",20,140);**

**g.drawLine(20,140,200,140);**

**g.setColor(Color.yellow);**

**g.fillRect(100,150,32,55);**

**g.setColor(Color.pink);**

**g.fillOval(150,146,60,60);**

**g.setColor(Color.darkGray);**

**g.fillArc(230,150,65,50,30,270);**

**}**

**}**

Double Bufferring

Drawing in applets is almost always done with double-buffering.   
This means that drawing is first done to an offscreen image, and when all   
is done, the offscreen image is drawn on the screen.   
This reduces the nasty flickering applets otherwise have.

import java.applet.\*;   
import java.awt.event.\*;   
import java.awt.\*;

public class DoubleBuffering extends Applet implements MouseMotionListener   
{   
     // The object we will use to write with instead of the standard screen graphics   
     Graphics bufferGraphics;   
     // The image that will contain everything that has been drawn on   
     // bufferGraphics.   
     Image offscreen;   
     // To get the width and height of the applet.   
     Dimension dim;   
     int curX, curY;

     public void init()    
     {   
          // We'll ask the width and height by this   
          dim = getSize();   
          // We'll redraw the applet eacht time the mouse has moved.   
          addMouseMotionListener(this);   
          setBackground(Color.black);   
          // Create an offscreen image to draw on   
          // Make it the size of the applet, this is just perfect larger   
          // size could slow it down unnecessary.   
          offscreen = createImage(dim.width,dim.height);   
          // by doing this everything that is drawn by bufferGraphics   
          // will be written on the offscreen image.   
          bufferGraphics = offscreen.getGraphics();   
     }

      public void paint(Graphics g)    
     {   
          // Wipe off everything that has been drawn before   
          // Otherwise previous drawings would also be displayed.   
          bufferGraphics.clearRect(0,0,dim.width,dim.width);   
          bufferGraphics.setColor(Color.red);   
          bufferGraphics.drawString("Bad Double-buffered",10,10);   
          // draw the rect at the current mouse position   
          // to the offscreen image   
          bufferGraphics.fillRect(curX,curY,20,20);   
          // draw the offscreen image to the screen like a normal image.   
          // Since offscreen is the screen width we start at 0,0.   
          g.drawImage(offscreen,0,0,this);   
     }

     // Always required for good double-buffering.   
     // This will cause the applet not to first wipe off   
     // previous drawings but to immediately repaint.   
     // the wiping off also causes flickering.   
     // Update is called automatically when repaint() is called.

     public void update(Graphics g)   
     {   
          paint(g);   
     }

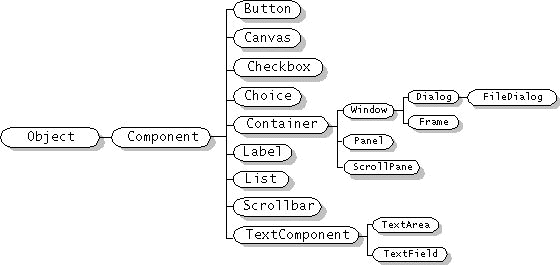
     // Save the current mouse position to paint a rectangle there.   
     // and request a repaint()   
     public void mouseMoved(MouseEvent evt)    
     {   
          curX = evt.getX();   
          curY = evt.getY();   
          repaint();   
     }

     // The necessary methods.   
     public void mouseDragged(MouseEvent evt)    
     {   
     }

 }

**Creating a User Interface Using the AWT**

The following figures show the inheritance hierarchies for all the AWT component classes.



**Example**

|  |
| --- |
| **import**java.awt.\*; **import**java.awt.event.\*;  **public class**DataEntry {   **public static void**main(String[] args) {     Frame frm=**new**Frame("DataEntry frame");     Label lbl = **new**Label("Please fill this blank:");     frm.add(lbl);     frm.setSize(350,200);     frm.setVisible(**true**);     frm.addWindowListener(**new**WindowAdapter(){       **public void**windowClosing(WindowEvent e){         [System](http://www.roseindia.net/java/example/java/awt/dataentry-frame-in-java.shtml" \t "_top).exit(0);       }     });         Panel p = **new**Panel();     Panel p1 = **new**Panel();         Label jFirstName = **new**Label("First Name");     TextField lFirstName = **new**TextField(20);     Label jLastName =**new**Label("Last Name");     TextField lLastName=**new**TextField(20);     p.setLayout(**new**GridLayout(3,1));     p.add(jFirstName);         p.add(lFirstName);     p.add(jLastName);     p.add(lLastName);     Button Submit=**new**Button("Submit");     p.add(Submit);     p1.add(p);     frm.add(p1,BorderLayout.NORTH);   } } |

**JavaBeans, JARs, and Safe Computing**

* **What are JavaBeans?**
* The Basic Structure of a Bean
* **The JavaBeans API**
* **Application Builder Support**
* **What are JAR files?**
* **Code signing and JAR Security**
* **The Java.security API**
* **Signing code with javakey**

**What are JavaBeans?**

* JavaBeans are reusable components of a Java application.
* Once a JavaBean is created it may be used with several Java applications at the same time.
* Beans are added to the application with the Archive attribute of the Applet tag.

<Applet code=”javaApp.class” Archives=”Bean1.jar”>

* Components make sense because they can be used again and again in as many Java applications as needed.
* Often JavaBeans will be used in distributed environment may be transferred across a low-bandwidth internet connection.
* The JavaBeans API, coupled with the platform-independent Java system on which it is based, creates the platform-independent component solution.

**The Basic Structure of a Bean**

* A bean is comprised of two primary things: data and methods that act on the data.
* The data part of a bean completely describes the state of the bean; the methods provide a way for the bean’s state to be modified and for actions to be taken accordingly.
* The fundamental parts of a bean:

JavaBeans Component

Data

Methods

* A bean can have methods with different types of access.
* For example, private methods are accessible within the internal of a bean, but protected methods are accessible both internally and in derived beans.
* The public methods are accessible internally, from derived beans and form other parties such as applications and other components, like JavaScript and VBScript.

The JavaBeans API

The main component services in the JavaBeans API are:

* Property Management
* Introspection
* Event handling
* Persistence
* Application builder support

**Property Management**

* The property management facilities in the JavaBeans API are responsible for handling all interactions relating to bean properties.
* Properties reflect the internal state of a bean and constitute a data part of a bean’s structure.
* Properties are discrete, named attributes of a bean that determine its appearance and behaviour.
* The major issues addressed by the JavaBeans API property management facilities are:
* Accessor Methods - An accessor method is a public method defined in a bean that directly reads or writes the value of a particular property. Each property in a bean must have a corresponding pair of accessor methods: one for reading the property and one for writing. The accessor methods responsible for reading are known as getter methods and those responsible for writing are known as setter methods.
* Indexed Properties - Indexed Properties are properties that represent an array of values.
* Bound and Constrained Properties - Bound properties are properties that provide notifications to an interested party based on changes in the property value. Constrained Properties are properties that enable an interested party to perform a validation on a new property value before accepting the modification.
* Introspection
* The introspection facilities in the JavaBeans API define the mechanism by which components make their internal structure readily available to the outside world.
* These facilities consists of the functional overhead necessary to enable development tools to query a bean for its internal structure, including the interfaces, methods and member variables that comprise the bean.
* The introspection services provided by the JavaBeans API are divided into 2 parts: low-level services and high-level services.
* The low-level API services are responsible for enabling wide access to the structural internals of a bean.
* The high-level services use the low-level services behind the scenes to provide access to limited portions of a bean’s internals.
* Reflection and Design Patterns
* The reflection services gather information about a bean and determine its public properties, methods and events by applying simple design patterns.
* Design patterns are rules applied to a bean’s method definitions that determine information about the bean.
* Explicit Bean Information
* The explicit introspection facility involves creating bean information class that specifies various pieces of information about a bean including a property list, method list, and event list.
* The Introspector
* The introspection facilities provide an introspector used to obtain explicit bean information for a bean.
* The introspector is responsible for traversing the inheritance tree of a bean to determine the explicit bean information for all parent beans.
* Event Handling
* The event handling facilities in the JavaBeans API specify an event-driven architecture that defines interactions among beans and applications.
* A bean capable of generating events is considered an event source; an application or bean capable of responding to an event is considered an event listener.
* Event sources and listeners are connected by an event registration mechanism that is part of the event – handling facilities.
* When the source generates an event, a specified method is called on the event listener with an event state object being sent along as its argument.
* Unicast and Multicast Event Sources
* Although most practical event sources support multiple listeners, the event-handling facilities provide for event sources that choose to limit their audience to a single listener. These sources are called unicast event sources; their counterparts are called multicast event sources.
* The primary functional difference between the two is that unicast event sources throw an exception if an attempt is made to register more than one listener.
* Event Adapters
* Event adapters act as intermediaries between event sources and listeners.
* Event adapters sit between sources and listeners and provide a way of inserting specialized event-delivery behavior into the standard source/listener event model.
* Persistence
* The persistence facilities in the JavaBeans API specify the mechanism by which beans are stored and retrieved within the context of a container.
* The information stored through persistence consists of all parts of a bean necessary to restore the bean to a similar internal state and appearance.
* Application Builder Support
* The application builder facilities provide the overhead necessary to edit manipulate beans using visual application builder tools.
* Application builder tools rely heavily on these facilities to enable a developer to visually layout and edit beans while constructing an application.
* Property Editors and Sheets
* A property sheet is a visual interface that provides editors for each public property defined for a bean.
* The individual editors used in a property sheet are called property editors.
* Customizers
* Customizers are user interfaces that provide a specialized way of visually editing bean properties.

What are JAR Files?

* A JAR file allows multiple resources to be bundled into a single, compressed archive file.
* Applets and resources download quicker when they are compressed.
* JAR files include a manifest file in the archive. This manifest gives message digests of the component files in the archive.

Manipulating JAR files with jar

* jar is a tool included with the JDK, which is used to create and manipulate JAR files.
* Options that can be used with jar are:
* v - Tells jar to generate verbose output about the actions it is performing
* f - Specifies the filename to manipulate

Creating a JAR file

* To create an archive, use jar with the c flag.
* The command for creating JAR file is:

jar cvf sample.jar sampleApplet.class SG.gif sampleSound.au

Listing the Contents of a JAR File

* The option t tells jar to list the table of contents for the JAR file.
* jar tvf sample.jar

Extracting a File from a JAR File

* The X option specifies to extract files.

jar xvf sample.jar sampleGraphic.gif

Code Signing and JAR Security

* The javakey utility included with the JDK provides facilities for managing identities and certificates, and for signing code.
* Along with a jar utility used to generate JAR files, javakey allows to sign code and place the class file into a single archive.

**Integrating Java and JavaScript**

* The Browsers Role
* Accessing a Java Applet with JavaScript
* Accessing JavaScript from a Java Applet

The Browsers Role

* In order for java and javascript to interact on web pages, both have to be active and enabled in the user’s browser.
* Netscape Navigator and Internet Explorer both provide an interface for controlling how java and javascript are received.

Internet Explorer

* The scripting languages available in IE, Jscript and VBScript are automatically enabled each time the browser starts.
* The steps to make sure Java is active in IE are:
  1. From IE menu bar, choose view | options
  2. Select the Advanced tab.
  3. Scroll down the list of advanced options to Java VM. Select the JavaJIT compiler enabled check box.
  4. Select the Enable java logging check box if you want a log file created for java applets.

Netscape Navigator

* To make sure both features are active in Netscape Navigator, follow these steps:
  1. Choose Edit | Preferences
  2. Select the Advanced category from the left frame.
  3. Java and JavaScript check boxes are available in the right frame. If they have been disabled, they can be enabled by checking the boxes.

Accessing a Java Applet with JavaScript

* JavaScript can be used to modify applet behavior by creating a bridge between the Java Applet and the Html object model.
* Any programmable Html element can control the applet with the help of JavaScript.
* Netscape supports a direct access model for JavaScript and Java interaction, Explorer treats Java applets like ActiveX controls.
* The bridge is created in a JavaScript using the java keyword Packages, which is supported only in Netscape.
* This keyword is used by the JavaScript application t invoke the Java method directly.
* For example, the following code invokes the Java class system of the java.lang package and displays the string as an error message in the java console.
* function tryit()

{

var java = packages.java.lang.System.err

Java.println(“Error messages can be sent directly to Java”);

}

* While the IE browser doesn’t support a direct access, it does provide a much broader Html element from which the java applet may be accessed by JavaScript.
* These elements, known as programmable elements, can have an event assigned to them that activates an applet and control the applet’s functionality.
* Example
* <html>
* <head>
* <style>
* .hand { cursor:hand }
* </style>
* <script>
* Function aunapp()
* {
  + Document.applets.sampleapp.start()

}

</script>

</head>

<body>

<applet id=”sampleapp” code=”sampleapp.class”>

<param id=pict name=image value=”rose.gif”>

</applet>

<H1 classs=”hand” id=”init” onclick =” runapp()” title=”start the application”> </h1>

<input type=button value=”stop” onclick = “document.applets.sampleapp.stop()”>

</body>

</html>

### Calling Java Methods

Using the new JavaScript Packages object, JavaScript can invoke native Java methods directly.

Groups of related classes are combined in a construct called a package. Classes from a package are usable by outside classes by using the import command.

An example in all applets is the java package. One section of the package, java.awt.Graphics, is imported into every applet to give the paint method the additional methods it needs to add items to the applet screen. Because all applets are outside of the java package, its classes or subsets of those classes are imported into the applet for local use.

The syntax to call a Java package directly is

[Packages.]*packageName.className.methodName*

The object name is optional for the three default packages-java, sun, and netscape. These three can be referenced by their package name alone:

java.*className.methodName*

sun.*className.methodName*

netscape.*className.methodName*

Together with the package name, the object and class names can result in some unwieldy and error-prone typing. This is why you can also create new variables using the Package product:

var System = Packages.java.lang.System;

System.out.println("Hello from Java in JavaScript.");

### Controlling Java Applets

Controlling an applet with a script is a fairly easy matter, but it does require a knowledge of the applet you're working with. Any public variable, method, or property within the applet is accessible through JavaScript.

There are five basic activities common to all applets, as opposed to one basic activity for applications. An applet has more activities to correspond to the major events in its life cycle on the user's browser.

None of the activities have any definitions. You must override the methods with a subclass within your applet.

Initialization-Occurs after the applet is first loaded. This can include creating objects, setting state variables, and loading images.

Starting-After initialization or stopping, an applet is started. The difference between initialization and starting is that initialization only happens once, while starting can occur many times.

Painting-This method is how the applet actually gets information to the screen, from simple lines and text to images and colored backgrounds. Painting can occur a lot of times in the course of an applets life.

Stopping-Stopping suspends the applet execution and stops it from using system resources. This can be important since an applet continues to run even after a user leaves the page.

Destroying-This is the extreme form of stop. Destroying an applet begins a clean-up process in which running threads are terminated and objects are released.

With this information in hand, getting started begins with the <APPLET> tag. It helps to give a name to your applet to make JavaScript references to it easier to read This isn't absolutely necessary as JavaScript creates an array of applets when the page is loaded. However, it does make for a much more readable page.

<APPLET CODE="UnderConstruction" NAME="AppletConstruction" WIDTH=60 HEIGHT=60>

</APPLET>

To use a method of the applet from JavaScript, use the following syntax:

document.*appletName.methodOrProperty*Using the Applets Array

an array is created that contains a reference to every instance of the <applet> tag. It is used to reference all the applets on a pge.

Syntax:

document.all.appletID.methodProperty

document.all(n).methodProperty

document.applets[index].methodOrProperty  
document.applets[appletName].methodOrProperty

**Starting and Stopping an Applet with JavaScript**

One of the easy methods of controlling applet behavior is starting and stopping its execution. This can be accomplished using the start and stop methods-common to every applet. Use a form and two buttons to add the functions to your Web page (see Figure 38.4):

<FORM>

<INPUT TYPE="button" VALUE="Start" onClick="document.*appletName*.start()">

<INPUT TYPE="button" VALUE="Stop" onClick="document.*appletName*.stop()">

</FORM>

**Public in Java**

You can also call other methods, depending on their visibility to the world outside the applet. Any method or variable with a public declaration can be called by JavaScript.

The syntax to call applet methods from JavaScript is simple and can be integrated with browser events such as the preceding button code snippet:

document.*appletName.methodName*(*arg1,arg2,arg3*)

**Accessing JavaScript from a Java Applet**

Java can take a direct look at your HTML page through JavaScript objects. This requires the use of the netscape.javascript.JSObject class when the applet is created.

To include the JSObject class as part of your applet, use the import command as you would normally include any other class package.

import netscape.javascript.JSObject;

An important addition is also necessary in the applet tag-MAYSCRIPT. This is a security feature which gives specific permission for the applet to access JavaScript objects.

<APPLET CODE="colorPreview.class" WIDTH=50 HEIGHT=50 NAME="Preview" MAYSCRIPT>

Without it, any attempt to access JavaScript from the applet results in an exception. If you wish to exclude an applet from accessing the page, simply leave out the MAYSCRIPT parameter.

There are five subsets of classes within the java package:

lang-These classes and interfaces are the core of the Java language. This subset includes the Runnable interface (used for threading) and the basic data types (Boolean, character, class, integer, object, string, etc.). It also includes the System class, which provides access to system-level behaviors.

util-This group of utility interfaces and classes aren't crucial to running Java, but they provide ways to make programming easier. It includes utilities to generate random numbers, stacks, hash tables, and dates.

awt-The Abstract Windowing Toolkit (also known as Another Windows Toolkit) contains the graphical items to help create user interfaces and other graphical items. It includes interfaces for a layout manager and menu container, along with classes for form elements, colors, keyboard and mouse events, fonts, images, menus, and windows.

Io-Used for passing information in and out of applets and applications, this subset includes classes for sending and receiving input streams and files, not including networking activity (see net).

Net-This subset of classes has the tools and operations for working over a network. This group includes methods and interfaces to handle URLs, URL content, and socket   
connections.

### Java and JavaScript Values

JSObject gives Java the ability to look at and change objects defined through JavaScript. This requires certain assumptions, especially when passing or receiving values from Java. Every JavaScript value is assigned some form from java.lang.Object to ensure compatibility.

* Objects-Any object sent or received from Java remains in its original object wrapper.
* Java numbers-Since JavaScript doesn't support the variety of numerical types as Java (byte, char, short, int, long, float, and double), they lose their specific type and become a basic JavaScript number.
* JavaScript numbers-There's no way to tell what kind of number Java may be receiving from JavaScript. So, all JavaScript numbers are converted to Java floats.
* Booleans and Strings-These are passed essentially unchanged. Java Booleans become JavaScript Booleans and vice versa. The same occurs with strings.

### Looking at the JavaScript Window

In order to get a handle on JavaScript objects, including form items and frames, you must create an object to hold the current Navigator window first. getWindow provides the means.

First, you'll need to create a new: variable of type JSObject:

JSObject jsWin;

Then, using the JSObject class, assign the window to the variable:

jsWin = JSObject.getWindow(this);

This type of work is typically accomplished within the applet's init() method.

After you have a handle on the window, you can start to break it apart into its various components with getMember. This method returns a specific object from the next level of precedence. For example, to get a handle on a form on a Web page with a form called response, the following set of statements can be used:

jsWin = JSObject.getWindow(this);

JSObject jsDoc = (JSObject) jsWin.getMember("document");

JSObject responseForm = (JSObject) jsDoc.getMember("response");

In JavaScript, this form is referred to as window.document.response. Note that each JavaScript object is assigned to its own variable in Java and is not a property of a parent object. The form in Java is contained in responseForm, not jsWin.jsDoc.responseForm.

So far, you've only retrieved broad objects, such as windows and forms. Getting a specific value from JavaScript follows the same principles, although now you need a Java variable of the proper type to hold the results instead of an instance of JSObject.

Using the preceding form, let's say there's a text field (name), a number (idNum), and a checkbox (member). Each of these values is retrieved from JavaScript using the following commands:

jsWin = JSObject.getWindow(this);

JSObject jsDoc = (JSObject) jsWin.getMember("document");

JSObject responseForm = (JSObject) jsDoc.getMember("response");

JSObject nameField = (JSObject) responseForm.getMember("name");

JSOBject idNumField = (JSObject) responseForm.getMember("idNum");

JSOBject memberField = (JSObject) responseForm.getMember("memberField");

String nameValue = (String) nameField.getMember("value");

Float idNumValue = (Float) idNumField.getMember("value");

Boolean memberValue = (Boolean) memberField.getMember("checked");

This chunk of code becomes a bit unwieldy, especially when there are several values needed from JavaScript. If you need to access more than several elements on a page, it helps to create a new method to handle the process.

protected JSObject getElement(String formName, String elementName) {

JSObject jsDoc = (JSObject) JSObject.getWindow().getMember("document");

JSObject jsForm = (JSObject) jsDoc.getMember(formName);

JSObject jsElement = (JSObject) jsElement.gerMember(elementName);

return jsElement;

}

This simple method creates the intervening JSObjects needed to get to the form element, making the retrieval as easy as knowing the form and element name.

To change a JavaScript value, use the JSObject setMember method in Java. The syntax is setMember(*name*, *value*), with the name of the JavaScript object and its new value.

JSObject nameField = getElement("response","name");

nameField.setMember("name","Your Name Here");

This snippet uses the getElement method just defined to get the name element from the response form, and then uses the JSObject method setMember to set its value to Your Name Here. This is equivalent to this.*name* = *newValue* in JavaScript.

The two methods covered in this section (getWindow, getMember), are the basic methods used when interfacing with JavaScript. Together, it makes receiving values from an HTML page by way of JavaScript a straightforward task, even if it is a little cumbersome in the number of statements needed to accomplish it.

#### Getting Values Using Indexes

If your applet is designed to work with a variety of HTML pages which may contain different names for forms and elements, you can use the JavaScript arrays with the JSObject slot methods. If the desired form is always the first to appear on the document and the element is the third, then the form name is forms[0] and the element is elements[2].

After retrieving the document object using getWindow and getMember, use getSlot(*index*) to return a value within it. For example, in an HTML document containing three forms, the second is retrieved into Java using the following commands:

JSOBject jsWin = JSObject.getWindow(this);

JSObject jsDoc = (JSObject) jsWin.getMember("document");

JSObject jsForms = (JSObject) jsDoc.getMember("forms");

JSObject jsForm1 = (JSObject) jsForms.getSlot(1);

Using setSlot, the same process is used to load a value into an array. The syntax is

*JSObject*.setSlot(*index*,*value*);

where the index is an integer and the value is a string, Boolean or float.

### Using JavaScript Methods in Java

The netscape class package provides two methods to call JavaScript methods from within an applet-call and eval. The syntax between the two is slightly different, but the outcome is the same. Note that you need a handle for the JavaScript window before you can use these methods.

There are two ways to invoke these methods. The first uses a specific window instance, while the second uses getWindow to create a JavaScript window just for the expression.

jsWin.*callOrEval*(*arguments*)

JSOBject.getWindow().*callOrEval*(*arguments*)

The call method separates the method from its arguments. This is useful for passing Java values to the JavaScript method. The syntax is call("*method*", *args*), where the method is the name of the method you want to call and the arguments you want to pass are contained in an array.

eval, on the other hand, uses a string which appears identical to the way a method is called within JavaScript. The syntax is eval("*expression*"), where the expression is a complete method name and its arguments, such as document.writeln("Your name here.'"). Including it in the eval expression results in eval("document.writeln(\"Your name here.\");").

**UNIT – IV**

**CGI and Controlling the Web from the Server**

**Putting your Server to Work**

* Server – Side Programs and Scripts
* Advantages to Server- Side Applications
* Common Server – Side applications
* Placing Server Programs in your Web Site

**Server – Side Programs and Scripts**

* Server – Side Programs and Scripts are applications that run on a web server machine rather than on the user’s machine.
* Server – Side applications can perform a variety of functions, such as working with databases, searching for keywords in documents, and creating custom log files.
* Unlike client – side scripting, server – side applications can be either scripts or programs.
* Programs are compiled applications, whereas scripts are interpreted.

Compiling versus Interpreting

....

**Advantages to Server – Side Applications**

* Cross – Platform / Cross – Browser Support
* More options for applications
* Increased power
* Code integrity

Cross – Platform / Cross – Browser Support

* A major concern when programming for the client side is that the type machine, configuration, or web browser with which the user is visiting are not known.
* These large unknowns are the environments in which the program must be able to run.

More options for Applications

* Server- side programming offers more control and flexibility.
* On the server, there is a ability to access outside information, such as data in databases.
* You can run complex programs in high level programming language, or perform an electronic payment transaction to fulfil online orders.
* The only restriction is that you return you results to the user’s web browser in formatting that can be properly displayed in a web browser.

Increased Power

* Over the last several years, the performance level of client machines has risen dramatically, and the gap in performance between these two machines has narrowed.
* As a result, more and more functionality is being distributed to these more powerful client machines, easing the load on the server machine.
* Because you cannot always count on the performance level of the user’s machine, you want to code your processor – intensive applications for the server side.
* High performance web server machines are still more powerful than the average computer being used for web browsing.
* Even at high loads, the web server can quickly run the program and sends the result back to the user’s web browser.

Code Integrity

* The advantage to coding applications on the server side is that the source code is not available to other users.

Common Server – Side Applications

* Server – side programming is a powerful way to extend the functionality of the web site.
* Most of the applications implemented on the server need to be there for some reason, usually because of access to server – side resources.
* Some of the server – side applications are:
* Search Engines
* Database access
* Chat and bulletin board services
* Form handlers

Search Engines

* A search engine is the program that enables users to search through documents on the web site.
* Simple search engine may only have some predefined categories for which you can search for matches.
* Complex search engines allow keyword searches, returning all the documents that contain the supplied keywords.
* However, a keyword search is much more difficult to implement for a programmer, and can be much more time consuming for the machine performing the search.
* Search engines must be implemented on the server side.
* The documents or document index reside on the serve machine. The search engine must look through these items on the web server in order to find matches to the user’s search string.

Database Access

* Databases are powerful way to store information. When data is stored in a database, it is easy to maintain by adding, deleting, and modifying records.
* The world wide web can act as a database access system. The front end, where the user sends the request for data and then views the results, is the web browser.
* After the request reaches the web server, which requires the data retrieval from a database, a server side application must handle the request.
* The server side program, usually CGI script, then queries the database for the necessary information, formats the results for display in a web browser, and sends the information back to the user’s web browser.

Chat and Bulletin Board Services

* Chat is the act of communicating with other people online in real time. This can be done with text xhat, graphical chat, or voice chat.
* Bulletin boards are structure that allow users to read messages from other users, and post their own messages.

Form Handlers

* In order for a user’s data to be sent to a database, form data must be sent to the web server.
* CGI script is needed to decode the data and place it in the database.

Placing Server Programs in the Web Site

* Several options are available for integrating server – side programming on the server side.
* The various options are
* Direct linking to the program
* Server – Side includes
* Handling form actions
* Calling from client – side scripts

Direct linking to the program

* A convenient CGI scripts feature is that they can be directly referenced in a link from an Html page.
* When the web server receives a request for a CGI script, it runs the script and returns the results to the user’s web browser.
* Example

<a href=<http://www.somedomain.com/cgi-bin/myscript.pl>>

* When the user clicks the link, the CGI program is executed and returns the latest information to the web browsers.

Server – Side Includes

* Server – side includes are actually an Html feature that allows you to embed other items in an html file.
* The following is the syntax for a server- side include :

<! -- #command name=”value1” name2=”value2” .... -- >

* The syntax for server – side include that calls a CGI program makes use of the exec command and looks like the following

<! -- #exec cgi = “ / cgi - bin /cgi-script.pl” -- >

* Server – side includes work through a process known as server parsing. With server parsing, the web server skims through html file’s contents looking for server based commands, such as server side includes.
* When the web server finds one, it performs the requested action and places the results of the action in the place of the server-side include statement.

Handling Form Actions

* The most common method of integrating a CGI script into web site is by setting it up to handle form data. This is done by using the action attribute in the <form> tag.
* For example, the following <Form> tag sends any form data entered by the user to the CGI script form-handler.pl when the form is submitted.

<form method=post action=”/cgi-bin/form-handler.pl”>

Calling from Client – Side Scripts

* CGI scripts can also be called from a client- side script. This method is similar to direct linking to a CGI script.
* The only difference is that the client – side script is requesting the URL rather than the user clicking a link with a CGI script in the URL.
* JavaScript statement to call a CGI script is:

Document.location = <http://www.somedomain.com/cgi-bin/myscript.pl>

Integrating Directly into the Web Server

* CGI applications are a powerful way to add functionality to the web server.
* However, there are some disadvantages to using CGI. First, CGI applications are not run as part of the web server. Whenever a web server receives a request for a CGI application, it must launch a separate application – the CGI application. This can cause a slight performance drain on some systems, especially when many users are simultaneously accessing CGI application.
* An alternative to the CGI is integrating application directly into the web server. This is done in one of two ways.
* The first way is to program application as a server plug-in, utilizing the web server’s API. The other method is to make use of server- side scripting, a new feature of the more popular web servers.

Server Plug – Ins

* Server – side plug – ins can extend the functionality of the web server.
* By writing the application as a plug-in, the application actually becomes part of the web server.
* Plug – ins in general are made possible through the Application Programming Interface (API). The API contains a set of functions and procedures that can be called from the application.
* Server – side plug-ins operate more efficiently than CGI application because plug – ins are initialized when the web server is started.
* There is also less overhead, because the web server does have to launch a separate application.

Server – Side Scripting

* Because server – side plug ins are more efficient than CGI applications, but more challenging to program, there has been a recent push to support server – side scripting.
* Server – side script is parsed and run by the web server before the web page is sent to the web browser.
* Two server side scripting methods have emerged from two of the more popular web servers.
* LiveWire is Netscape Communications Corporation’s server side scripting solution, and Active Server Pages is Microsoft’s.

LiveWire

* LiveWire is the name associated with server – side scripting for Netscape’s web server.
* LiveWire enables us to embed javascript in the html pages, which gets parsed by the web server before the document is sent to the web browser.
* A major difference between server- side javascript with LiveWire and client – side javascript is that the server-side scripts are compiled into bytecode before being placed on the web server.

Active Server Pages

* ASP is Microsoft’s implementation of server – side scripting.
* ASP operates in a manner similar to LiveWire, except that ASP pages are not to be compiled to bytecode.
* When the Microsoft web server receives a request for an ASP page, it parses the code, executing any server – side commands.
* After it has finished parsing the page, the results are sent to the user’s web browser.
* ASP provides native support for combining html, vbscript and jscript. Developers can combine code from either of the scripting languages into the html code of an ASP.

**Traditional CGI Programming**

* What is the CGI?
* HTTP Connections and Headers
* Sending Data to CGI Application
* Returning Data from CGI Application
* Calling CGI Applications

What is the CGI?

* CGI stands for Common Gateway Interface and is set of standards around the communication between web server and server side applications.
* These standards provide the gateway through which data ca pass between the web server and CGI application.
* The CGI specification defines how web servers will make information available to CGI applications and how CGI applications will return data to the web server.

Programming Language Choices

* Because CGI programming and scripting don’t refer to a specific language for coding applications for web sites, a language has to be chosen with which to work.
* The most common languages used for CGI applications are Perl, C, C++, TCL, Unix shells, Java, Visual Basic, and AppleScript.

HTTP Connections and Headers

* An HTTP connection is the communication channel between the web browser and the web server.
* Most HTTP connections begin on the client side, with the web browser sending an HTTP request to a web server for a document.
* At the beginning of the request is a section referred to as the request header. The request header contains information about the request and about the web browser requesting the information.
* HTTP Request Headers

|  |  |
| --- | --- |
| **Header  [↓](http://en.wikipedia.org/wiki/List_of_HTTP_headers)** | **Description** |
| Accept | Content-Types that are acceptable |
| Accept-Charset | Character sets that are acceptable |
| Accept-Encoding | Acceptable encodings |
| Accept-Language | Acceptable languages for response |
| Accept-Ranges | Allows the server to indicate its acceptance of range requests for a resource |
| Authorization | Authentication credentials for HTTP authentication |
| Cache-Control | Used to specify directives that MUST be obeyed by all caching mechanisms along the request/response chain |
| Connection | What type of connection the user-agent would prefer |
| [Cookie](http://en.wikipedia.org/wiki/HTTP_Cookie) | an HTTP cookie previously sent by the server with Set-Cookie (below) |
| Content-Length | The length of the request body in [octets](http://en.wikipedia.org/wiki/Octet_(computing)) (8-bit bytes) |
| Content-Type | The [mime type](http://en.wikipedia.org/wiki/Mime_type) of the body of the request (used with POST and PUT requests) |
| Date | The date and time that the message was sent |
| Expect | Indicates that particular server behaviors are required by the client |
| From | The email address of the user making the request |
| Host | The domain name of the server (for [virtual hosting](http://en.wikipedia.org/wiki/Virtual_hosting)), mandatory since HTTP/1.1 |
| If-Match | Only perform the action if the client supplied entity matches the same entity on the server. This is mainly for methods like PUT to only update a resource if it has not been modified since the user last updated it. |
| If-Modified-Since | Allows a *304 Not Modified* to be returned if content is unchanged |
| If-None-Match | Allows a *304 Not Modified* to be returned if content is unchanged, see [HTTP ETag](http://en.wikipedia.org/wiki/HTTP_ETag) |
| If-Range | If the entity is unchanged, send me the part(s) that I am missing; otherwise, send me the entire new entity |
| If-Unmodified-Since | Only send the response if the entity has not been modified since a specific time. |
| Max-Forwards | Limit the number of times the message can be forwarded through proxies or gateways. |
| Pragma | Implementation-specific headers that may have various effects anywhere along the request-response chain. |
| Proxy-Authorization | Authorization credentials for connecting to a proxy. |
| Range | Request only part of an entity. |
| [Referer](http://en.wikipedia.org/wiki/HTTP_referer) | This is the address of the previous web page from which a link to the currently requested page was followed. |
| TE | The transfer encodings the user agent is willing to accept: the same values as for the response header Transfer-Encoding can be used, plus the "trailers" value (related to the "[chunked](http://en.wikipedia.org/wiki/Chunked_transfer_encoding)" transfer method) to notify the server it accepts to receive additional headers (the trailers) after the last, zero-sized, chunk. |
| Upgrade | Ask the server to upgrade to another protocol. |
| User-Agent | The [user agent](http://en.wikipedia.org/wiki/User_agent) string of the user agent |
| Via | Informs the server of proxies through which the request was sent. |
| Warn | A general warning about possible problems with the entity body. |

* After the web server receives an HTTP request from a web browser, it evaluates the request, returning the requested document if it exists, and then shuts down the HTTP connection.
* HTTP Response Headers

|  |  |
| --- | --- |
| **Header  [↓](http://en.wikipedia.org/wiki/List_of_HTTP_headers)** | **Description** |
| Accept-Ranges | What partial content range types this server supports |
| Age | The age the object has been in a [proxy](http://en.wikipedia.org/wiki/Proxy_server) cache in seconds |
| Allow | Valid actions for a specified resource. To be used for a *405 Method not allowed* |
| Cache-Control | Tells all caching mechanisms from server to client whether they may cache this object |
| Content-Encoding | The type of encoding used on the data |
| Content-Language | The language the content is in |
| Content-Length | The length of the response body in [octets](http://en.wikipedia.org/wiki/Octet_(computing)) (8-bit bytes) |
| Content-Location | An alternate location for the returned data |
| Content-Disposition | An opportunity to raise a "File Download" dialogue box for a known MIME type |
| Content-MD5 | A [Base64](http://en.wikipedia.org/wiki/Base64)-encoded binary [MD5](http://en.wikipedia.org/wiki/MD5) sum of the content of the response |
| Content-Range | Where in a full body message this partial message belongs |
| Content-Type | The [mime type](http://en.wikipedia.org/wiki/Mime_type) of this content |
| Date | The date and time that the message was sent |
| [ETag](http://en.wikipedia.org/wiki/HTTP_ETag) | An identifier for a specific version of a resource, often a [Message Digest](http://en.wikipedia.org/wiki/Hash_function), see [ETag](http://en.wikipedia.org/wiki/HTTP_ETag) |
| Expires | Gives the date/time after which the response is considered stale |
| Last-Modified | The last modified date for the requested object, in [RFC 2822 format](http://www.ietf.org/rfc/rfc2822.txt) |
| [Location](http://en.wikipedia.org/wiki/HTTP_location) | Used in redirection, or when a new resource has been created. |
| Pragma | Implementation-specific headers that may have various effects anywhere along the request-response chain. |
| Proxy-Authenticate | Request authentication to access the proxy. |
| [Refresh](http://en.wikipedia.org/w/index.php?title=HTTP_refresh&action=edit&redlink=1) | Used in redirection, or when a new resource has been created. This refresh redirects after 5 seconds. (This is a proprietary/non-standard header extension introduced by Netscape and supported by most web browsers.) |
| Retry-After | If an entity is temporarily unavailable, this instructs the client to try again after a specified period of time. |
| Server | A name for the server |
| Set-Cookie | an [HTTP cookie](http://en.wikipedia.org/wiki/HTTP_cookie) |
| Trailer | The Trailer general field value indicates that the given set of header fields is present in the trailer of a message encoded with [chunked transfer-coding](http://en.wikipedia.org/wiki/Chunked_transfer_encoding). |
| Transfer-Encoding | The form of encoding used to safely transfer the entity to the user. [Currently defined methods](http://www.iana.org/assignments/http-parameters) are: [chunked](http://en.wikipedia.org/wiki/Chunked_transfer_encoding), compress, deflate, gzip, identity. |
| Vary | Tells downstream proxies how to match future request headers to decide whether the cached response can be used rather than requesting a fresh one from the origin server. |
| Via | Informs the client of proxies through which the response was sent. |
| Warning | A general warning about possible problems with the entity body. |
| WWW-Authenticate | Indicates the authentication scheme that should be used to access the requested entity. |

Sending data to CGI application

* When started by the web server, CGI programs and scripts run as separate applications on the web server machine. They are not integrated within the web server environment.
* Because of this, CGI application does not have native access to the data sent from the web browser in the HTTP request. This includes both the HTTP request headers and any user specified data, such as in an html form.
* CGI specification defines how information passes between web server and CGI applications.

The GET and POST Request Methods

* These are the two ways in which a request can be sent to the web server.
* GET request is used to specify a request for a document from the web server. POST methods are used when form data is being sent from the user’s web browser to the web server.

Appending Information to URLs

* Two forms of additional information can be appended to a URL with the GET method.
* The first form is additional path information. Path information is usually the path to a resource on the server machine, and is usually used only with CGI applications.
* For example, with server side image maps, the URL sent by the browser is in the following form:

<http://www.castingguild.com/cgi-bin/imagemap/maps/groups.map>? 201, 118

* The URL to the CGI application is as follows:

<http://www.castingguild.com/cgi-bin/imagemap>

* The first part of additional information is as follows:

/maps/ groups.map

* This is some extra path information being sent to the CGI script.
* The second part of additional information is as follows:

? 201, 118

* This second form is referred to as a query string. It is used to supply additional data that the application may need.

The POST method and Standard Input

* Web servers send data to CGI applications through either environment variables or standard input.
* The only time standard input is used when the request method used by the web browser is the POST method.

Through Environment Variables

* Environment variables contain useful information about the HTTP request, the web browser and the web server.
* CGI environment variables

|  |  |
| --- | --- |
| Key | Value |
| DOCUMENT\_ROOT | The root directory of your server |
| HTTP\_COOKIE | The visitor's cookie, if one is set |
| HTTP\_HOST | The hostname of the page being attempted |
| HTTP\_REFERER | The URL of the page that called your program |
| HTTP\_USER\_AGENT | The browser type of the visitor |
| HTTPS | "on" if the program is being called through a secure server |
| PATH | The system path your server is running under |
| QUERY\_STRING | The query string (see GET, below) |
| REMOTE\_ADDR | The IP address of the visitor |
| REMOTE\_HOST | The hostname of the visitor (if your server has reverse-name-lookups on; otherwise this is the IP address again) |
| REMOTE\_PORT | The port the visitor is connected to on the web server |
| REMOTE\_USER | The visitor's username (for .htaccess-protected pages) |
| REQUEST\_METHOD | GET or POST |
| REQUEST\_URI | The interpreted pathname of the requested document or CGI (relative to the document root) |
| SCRIPT\_FILENAME | The full pathname of the current CGI |
| SCRIPT\_NAME | The interpreted pathname of the current CGI (relative to the document root) |
| SERVER\_ADMIN | The email address for your server's webmaster |
| SERVER\_NAME | Your server's fully qualified domain name (e.g. www.cgi101.com) |
| SERVER\_PORT | The port number your server is listening on |
| SERVER\_SOFTWARE | The server software you're using (e.g. Apache 1.3) |

Returning Data from CGI Application

* After CGI application is finished executing, it needs to return a result to the user’s web browser. This result will be Html formatted text.
* Whether returning the header or returning other data, CGI applications return results by sending data to standard output.

Calling CGI Applications

* There are several ways for calling CGI applications.
* The most common way CGI applications are called is from the ACTION attribute in the <form> tag.
* CGI applications can also be called by directly linking to them in the <A> tag.
* Server – side includes are another common way to call CGI applications.

**The Anatomy of a CGI Application**

* Creating the HTML page
* Creating the Form
* Handling the Form Data

Creating the HTML Page

* The first step in creating a CGI application to handle form data is to set up the HTML page and the form elements.

Planning your Form

* Carefully planning form and the elements it contains makes programming task much easier.
* For example, with the gender question, a better way would be to use a <select> statement, such as this:

<select name=”gender”>

<option> Male

<option> Female

</select>

Creating the Form

* After you have planned the questions you want to ask and the form elements used to receive the responses, you are ready to create HTML form.

Handling the Form Input

* CGI application needs to perform the following tasks:

1. Receive the form data
2. Decode the form data
3. Insert the data into the database
4. Return a result to the user

Receiving the User’s Data

* CGI application receives its data from two sources. One source is environment variables, which the web server sets when it starts the application. The second source is available only when form data is being sent with the POST request method.
* For example,

<form action=”/ungroomd / cgi-bin/ survey.pl” method=”post”>

* In the above <form> tag, the request method is set to POST. So the form data are sent through standard input to the CGI script.
* In Perl, the request method can be checked by using the following lines:

if( $ ENV(“REQUEST\_METHOD”) ne “POST”)

{

&Print\_Error(“This script can only be used with POST method”);

Exit(1);

}

**Decoding the User’s Data**

* The web browser encodes some of the characters in the user’s data before sending it to the web server.
* These data should be decoded before storing it into the database.

The algorithm for decoding form data is:

1. Read the query string from $ENV{QUERY\_STRING}.
2. If the $ENV{REQUEST\_METHOD} is POST, determine the size of the request using $ENV{CONTENT\_LENGTH} and read that amount of data from the standard input. Append this data to the data read from the query string, if present (this should be joined with "&").
3. Split the result on the " &" character, which separates name-value pairs (the format is name=value&name=value...).
4. Split each name-value pair on the "s" character to get the name and value.
5. Decode the URL-encoded characters in the name and value.
6. Associate each name with its value(s); remember that each option name may have multiple values.

A form sends its parameters as the body of a POST request, or as the query string of a GET request. However, it is possible to create a form that uses the POST method and direct it to a URL containing a query string. Thus, it is possible to get a query string with a POST request.

Here is a first attempt at our subroutine:

sub parse\_form\_data {

my %form\_data;

my $name\_value;

my @name\_value\_pairs = split /&/, $ENV{QUERY\_STRING};

if ( $ENV{REQUEST\_METHOD} eq 'POST' ) {

my $query = "";

read( STDIN, $query, $ENV{CONTENT\_LENGTH} ) == $ENV{CONTENT\_LENGTH}

or return undef;

push @name\_value\_pairs, split /&/, $query;

}

foreach $name\_value ( @name\_value\_pairs ) {

my( $name, $value ) = split /=/, $name\_value;

$name =~ tr/+/ /;

$name =~ s/%([\da-f][\da-f])/chr( hex($1) )/egi;

$value = "" unless defined $value;

$value =~ tr/+/ /;

$value =~ s/%([\da-f][\da-f])/chr( hex($1) )/egi;

$form\_data{$name} = $value;

}

return %form\_data;

}

Putting Data in a Simple Database

Steps for placing the data into a text file database are:

* Formatting the data for the database
* Opening the database
* Locking the database
* Appending the new record
* Unlocking the database
* Closing the database

Formatting the data for the database

* When the form data are added to the database, some special formatting of the data is needed before inserting it.
* This formatting could include adding additional fields based on user input, changing other fields based on some standards, or removing any special characters the data might contain.

Opening the Database

* With all the fields formatted for the database, the fields can be combined into a database record and can be appended to the database file.
* The first step is to open the database file. This can be done with the following Perl statement:

Open(DATABASE, “@database\_file”) ||

die “content-type: text/html\n\n Cannot open database.”;

Locking the Database

* In order to lock the database file, file system function has to be used that can modify access permission to the file.
* In PERL, this can be done with the flock statement, such as:

flock(DATABASE, 2);

* The first parameter is the stream associated with an open file.
* The number being sent as the second parameter is a code to the flock function. The number 2 denotes the locking the file.

Appending the New record

* In PERL, by using the filestream and print statement, the reccoed can be appended to the database file.
* Example

Print DATABASE $quote, $data\_received{“name”} . $quote . “,” .

$quote . $data\_received {“email”} . $quote . “,” .

$quote . $data\_received {“married”} . $quote . “,” .

$quote . $data\_received {“yearsmarried”} . $quote . “,” .

$quote . $data\_received {“relationship”} . $quote . “,” .

.

.

$quote . $data\_received {“comments”} . $quote . “\n” ;

Unlocking and Closing the database

* With the new record appended to the database, the lock on the database has to released.
* Subroutine used to unlock the database is:

sub unlock\_file

{

flock(DATABASE, 8);

}

* Closing the database file is even easier and is done with the following statement:

Close(DATABASE);

**Server – Specific Technologies : Netscape ONE Versus Microsoft Windows DNA**

* Technology Components of Netscape ONE
* Technology Components of Microsoft WindowsDNA
* Differences between the Netscape and Microsoft Platforms
* Web Server Extensions
* Server – Side JavaScript with Netscape’s LiveWire
* Server – Side scripting with Active Server Pages

Technology Components of Netscape ONE

* Netscape ONE is a group of technologies packaged under one corporate philosophy. The philosophical basis of these products is crossware support – providing the tools to build applications that can run in many environments – on many different types of machines.
* Netscape ONE technologies are:
* Components – Application objects built using a variety of languages such as Java, JavaScript, or C++, and interfaced with each other using CORBA / IIOP and the JavaBeans model.
* Directory Services - Provide techniques to programmatically access directory- based information.
* DynamicHTML - DynamicHTML is the capability to modify or position traditional Html elements using a combination of style sheets and scripting.
* IFC - Internet Foundation Classes that were a set of classes for working with interface components, and are now integrated into the sun JFC.
* Java - Netscape supports java, both from the client application, navigator, and from the server.
* JavaScript - Netscape’s scripting language, which is the fist scripting language to be used with client – side browsers.
* Metadata - Metadata controls things such as the base location of a web page for the resolution of relative URLs and other information.
* Netcaster - Netcape’s Channel Technology.
* Nethelp - Based on the capability to create HTML based help files, viewable online and based on open standards.
* Plug – ins - Netscape’s capability of embedding external software components into web pages for client – sie or server – side access.
* Security - Netscape has several security components, such as object signing, SmartCards, and digital certificates.
* Software Distribution - Netscape also supports several techniques for software publication, such as publishing to the server, and the SmartUpdate feature.

Technology Components of Microsoft WindowsDNA

* Microsoft also has its own platform – WindowsDNA or Distributed inteNet applications Architecture – and with it Microsoft lists its key technologies, Intenet and other.
* Microsoft DNA groupings are:
* Windows platforms - Includes the Windows operating systems and win32 technologies
* Common Object Model - Contains the object development structure that forms the framework of all the DNA technologies
* Internet Technologies - Including Internet Explorer, Dynamic HTML, Internet Information Server, the Microsoft site Server, and Java technologies
* Windows DNA Services - Including security, directory, transaction, message queuing, database management, data access, e-mail, scripting, and systems management
* Windows DNA Tools - Includes the company’s visual tools such as Visual Basic and Visual C++; component creation, such as ActiveX controls; team development with Visual SourceSafe; content creation, such as FrontPage; and web site development with Visual InterDev.
* Other Windows DNA server applications - Includes office