DEVELOPING DOWNLOAD MANAGER

VENU GOPAL REDDY KONDA-021006
CHAVALI SURYA SATISH-021020

DEPARTMENT OF ELECTRONICS AND COMMUNICATIONS
JAYPEE UNIVERSITY OF INFORMATION TECHNOLOGY
WAKNAGHAT

MAY 2006

CERTIFICATE

This is to certify that the work entitled, "Developing Juit Download Manager" submitted by Venu Gopal Reddy Konda in partial fulfillment for the award of degree of Bachelor of Technology in Electronics and Communications of Jaypee University of Information Technology has been carried out under my supervision. This work has not been submitted partially or wholly to any other university for the award of this or any other degree or diploma

Vivek Sehgal

Vivek Sehgal
[Sr Lecturer]

ACKNOWLEDGEMENT

We hereby express our gratitude's to our mentor who earnestly co-operated with us during this project. We thank Department of Electronics and Communications for co-operating in every possible way for the completion of project.

TABLE OF CONTENTS

LIST OF ABBREVATIONS

API	-	Application Programming Interface.
URL	-	Uniform Resource Locator.
J2SDK	-	Java 2 Standard Development Kit.
HTTP	-	Hypertext Transfer Protocol.
FTP	-	File Transfer Protocol.
HTTPS	-	Hyper Text Transfer Protocol over SSL
SSL	-	Secure Socket Layer.
DM	-	Download Manager.
GUI	-	Graphical User Interface.
JAVAC	-	Java Compiler.
IDE	-	Integrated Development Environment.
JVM	-	Java Virtual Machine.
JRE	-	Java Run-Time Environment.
AWT	-	Abstract Window Toolkit

ABSTRACT

Our project developing download manager is concerned with proper download of HTTP files with various functionalities that enables the user to exercise various options such as pause, resume, download etc. This download manager manages to resume the data from the byte it has stopped previously rather than downloading the whole data again. it belongs to the category of desktop application software. Various enhancements such as extending support to FTP, HTTPS (secured), bit-torrent files is possible.

CHAPTER-1

OVERVIEW

The Download Manager uses a simple but effective GUI interface built in java's Swing libraries. The Download Manager window is shown here. The use of Swing gives the interface a good, modern look and feel.

The GUI maintains a list of downloads that are currently being managed. Each download in the list reports its URL, size of the file in bytes, progress as a percentage towards completion, current status. The downloads can each be in one of the following states: Downloading, Paused, Complete, Error, Cancelled. The GUI also has controls for adding downloads to the list and for changing the state of each of the downloads in the list. When a download in the list is selected, depending on its current state, it can be paused, resumed, cancelled, or removed from the list altogether.

The download Manager is broken into a few classes for natural separation of functional components. These are the **Download, DownloadsTableManager, ProgressRenderer,** and **Download Manager** classes, respectively. The Download manager class is responsible for the GUI interface and makes use of the DownloadsTableModel and ProgressRenderer classes for displaying the current list of downloads. the Download class represents a “managed” download and is responsible for performing the actual downloading of a file. In the following sections, we will go through each of these classes in detail, highlighting their inner workings and explaining how they relate to each other.

CHAPTER-2

WHY JAVA???

The two main reasons why we choice Java For developing this programming language are :

Security

Portability.

Rich GUI's

Security:-When ever you connect to internet there is always a chance that your computer getting infected. A download manager should provide security that no virus gets downloaded along with the original files.

Java has a rich Security API's build into its JRE. Moreover Java answers all it's the security concerns by providing a "firewall" between a networked application and your computer.

And all the Java programs are runned in its JVM and java does not allow the programs to access system resources by default and thus making your system secure.

Portability:-Many types of computers and operating systems are in use throughout the world – and many are connected to the Internet. We wanted to develop a platform independent Download Manager , so that our DM can be ported to very platform like Microsoft Windows , Linux , Unix , Solaris ,Mac ,BSD with very few changes to the code.

Rich GUI'S:-DM is a Desktop Application which is used by general computer users. So we want to make our DM simple , easy-to-use , nice look. For this we have to use rich graphical user interfaces .Fortunately Java comes with two different graphical interface API'S .they are Swing and AWT. swing components are light weight components and AWT components are heavy weight components .both parallely provide rich graphical user interfaces.

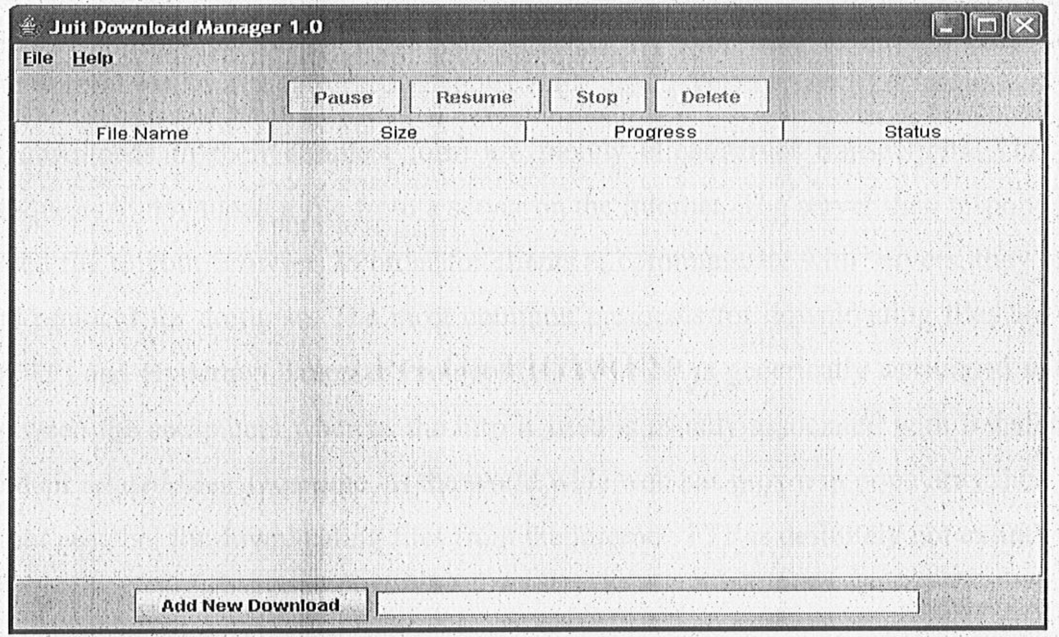
In Addition to these three there are many others areas in which Java is preferred.

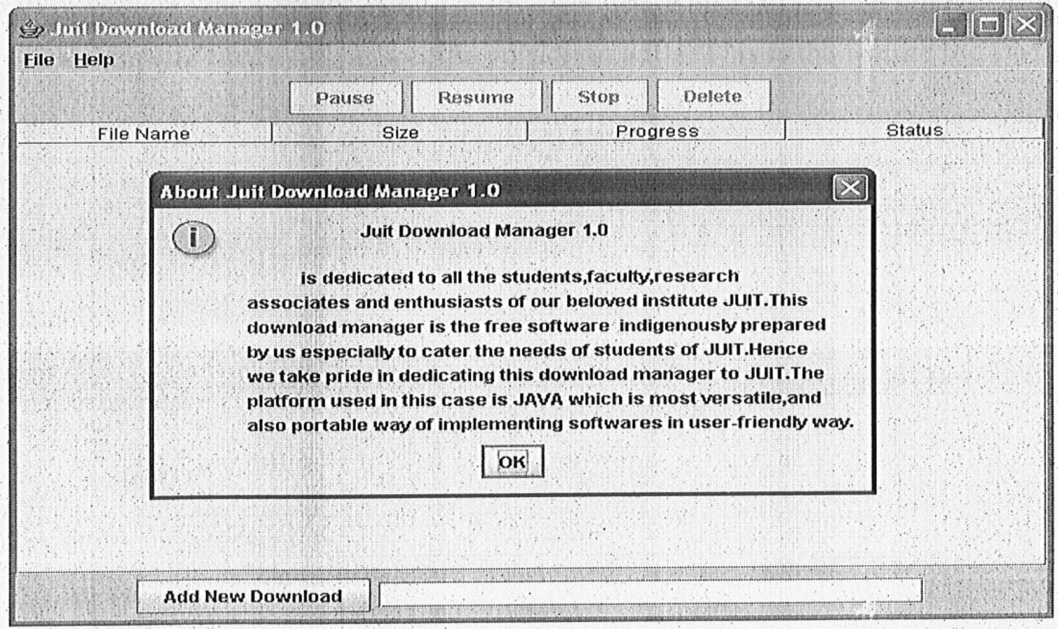
They are:-

Simple
Object-oriented
Robust
Multithreaded
Architecture-neutral
Interpreted
High performance
Distributed
Dynamic

LIST OF FIGURES

GRAPHICAL USER INTERFACE (GUI) OF INTERNET DOWNLOAD MANAGER





Juit Download Manager 1.0

is dedicated to all the students,faculty,research associates and enthusiasts of our beloved institute JUIT.This download manager is the free software indigenously prepared by us especially to cater the needs of students of JUIT.Hence we take pride in dedicating this download manager to JUIT.The platform used in this case is JAVA which is most versatile,and also portable way of implementing softwares in user-friendly way.

CHAPTER-3

UNDERSTANDING INTERNET PROTOCOLS & DOWNLOADS

To Understand and appreciate the Download Manager we first need to have a close look at how internet downloads work.

Internet downloads in their simplest form are merely client/server transactions. The client, your browser, requests to download a file from a server on the Internet. The server then responds by sending the requested file to your browser .In order for clients to communicate with servers, they must have an established protocol for doing so. The most common protocols for downloading files are file transfer protocol (FTP) and Hypertext Transfer Protocol (HTTP).FTP is generically associated with exchange of files between the computers whereas the http is used is usually associated with transferring of web pages and their related files .Overtime, as the world wide web has grown in popularity, http has become the dominant protocol for downloading files from the Internet. FTP is definitely not extinct though.

For brevity's sake the DM developed here will only support HTTP downloads. the difference between these two forms of lies in the way files can be requested from servers. With the antiquated HTTP1.0 client can only request that a server send it a file, whereas with HTTP1.1, a client can request that a server send it a complete file or only a specific portion of a file. This is the feature the DM is built on.

CHAPTER-4

UNIFORM RESOURCE LOCATOR

A Uniform Resource Locator (URL) is a string of characters conforming to a standardized format, which refers to a resource on the Internet (such as a document or an image), by its location. For example, the URL of our University is http://www.juit.ac.in.

An HTTP URL, commonly called a web address, is usually shown in the address bar of a web browser.

Tim Berners-Lee created the URL in 1991 to allow the publishing of hyperlinks on the World Wide Web, a fundamental innovation in the history of the Internet. Since 1994, the URL has been subsumed into the more general Uniform Resource Identifier (URI), but URL is still a widely used term.

The U in URL has always stood for Uniform, but it is sometimes described as Universal, perhaps because URI did mean Universal Resource Identifier before RFC 2396.

URIs and URLs

Every URL is a type of Uniform Resource Identifier (URI), or, more precisely, the set of URLs is a proper subset of the set of URIs. A URI identifies a particular resource while a URL both identifies a resource and indicates how to locate it. To illustrate the distinction consider the URI urn:ietf:rfc:1738 which identifies IETF RFC 1738 without indicating where to find the text of this RFC. Now consider three URLs for three separate documents containing the text of this RFC:
http://www.ietf.org/rfc/rfc1738.txt
http://www.w3.org/Addressing/rfc1738.txt
http://rfc.sunsite.dk/rfc/rfc1738.html

Each URL uniquely identifies each document and thus is a URI itself, but URL syntax is such that the identifier allows one to also locate each of these documents. Thus, a URL functions as the document's address.

Historically, the terms have been almost synonymous as almost all URIs have also been URLs. For this reason, many definitions in this article mention URIs instead of URLs; the discussion applies to both URIs and URLs.

URL scheme

A URL is classified by its scheme, which typically indicates the network protocol used to retrieve a representation of the identified resource over a computer network. A URL begins with the name of its scheme, followed by a colon, followed by a scheme-specific part.

Some examples of URL schemes:

Http - HTTP resources

Https - HTTP over SSL

Ftp - File Transfer Protocol

Mailto - E-mail address

ldap - Lightweight Directory Access Protocol lookups

File - resources available on the local computer or over a local file sharing network

News - Usenet newsgroups

Gopher - the Gopher protocol

Telnet - the TELNET protocol

Data - the Data: URL scheme for inserting small pieces of content in place

Some of the first URL schemes, such as the still-popular "mailto", "http", "ftp", and "file" schemes, along with the general syntax of URLs, were first detailed in 1994 in Request for Comments RFC 1630, superseded within a year by the more refined RFC 1738 and RFC 1808. Some of the schemes defined in that document are still in effect, while others have fallen into disuse or have been redefined by later standards. Meanwhile, the definition of the general syntax of URLs has forked into a separate line of URI specifications: RFC 2396 (1998) and RFC 2732 (1999), both of which are obsolete but still widely referenced by URL scheme definitions; and the current standard, STD 66 / RFC 3986 (2005).

Generic URL syntax

All URLs, regardless of scheme, must conform to a generic syntax. Each scheme can impart its own requirements for the syntax of the scheme-specific part, but the URL must still conform to the generic syntax.

Using a limited subset of characters compatible with the printable subset of the ASCII repertoire, the generic syntax allows a URL to represent a resource's address, regardless of the original format of the components of the address.

Schemes using typical connection-based protocols use common "generic URI" syntax, defined below:

scheme://authority/path?query#fragment

The authority typically consists of the name or IP address of a server, optionally followed by a colon and a TCP port number. It may also contain a username and password for authenticating to the server.

The path is a specification of a location in some hierarchical structure, using a slash ("/") as delimiter between components.

The query typically expresses parameters of a dynamic query to some database, program, or script residing on the server.

The fragment identifies a portion of a resource, often a location in a document.

Example: HTTP URLs

The URLs employed by HTTP, the protocol used to transmit web pages, are the most popular kind of URI and can be used as an example to scheme://host:port/path?parameter=value#anchor

URI references

The term URI reference means a particular instance of a URI, or portion thereof, as used in, for instance, an HTML document, in order to refer to a particular resource. A URI reference often looks just like a URL or the tail end of a URL. URI references introduce two new concepts: the distinction between absolute and relative references, and the concept of a fragment identifier.

An absolute URL is a URI reference that is just like a URL defined above; it starts with a scheme followed by a colon and then a scheme-specific part. A relative URL is a URI reference that comprises just the scheme-specific part of a URL, or some trailing component thereof. The scheme and leading components are inferred from the context in which the URL reference appears: the base URI (or base URL) of the document containing the reference.

A URI reference can also be followed by a hash sign ("#") and a pointer to within the resource referenced by the URI as a whole. This is not a part of the URI as such, but is intended for the "user agent" (browser) to interpret after a representation of the resource has been retrieved. Therefore, it is not supposed to be sent to the server in HTTP requests.

<u>Examples of absolute URLs:</u>

<u>http://distrowatch.com/dwres.php?resource=major</u>

http://distrowatch.com/table.php?distribution=ubuntu

<u>Examples of relative URLs:</u>

// /wiki/Train

/wiki/Train

<u>Case-sensitivity</u>

According to the current standard, the scheme and host components are case-insensitive, and when normalized during processing, should be lowercase. Other components should be assumed to be case-sensitive. However, in practice case-sensitivity of the components other than the protocol and hostname are up to the webserver and operating system of the system hosting the website.

URLs in everyday use

An HTTP URL combines into one simple address the four basic items of information necessary to retrieve a resource from anywhere on the Internet:

the protocol to use to communicate,

the host (server) to communicate with,

the network port on the server to connect to,

the path to the resource on the server (for example, its file name).

A typical URL can look like:

http:// distrowatch.com:80/wiki/Special:Search?search=train&go=Go

In the example above:

Http is the protocol,

distrowatch.com is the host,

80 is the network port number on the server (as 80 is the default value for the HTTP protocol, this portion could have been omitted entirely),

/wiki/Special:Search is the resource path,

?search=train&go=Go is the query string; this part is optional.

Most web browsers do not require the user to enter "http://" to address a webpage, as HTTP is by far the most common protocol used in web browsers. Likewise, since 80 is the default port for http it is not usually specified. One usually just enters a partial URL such as http://distrowatch.com/dwres.php?resource=major . To go to a homepage one usually just enters the host name, such as http://distrowatch.com/

Since the HTTP protocol allows a server to respond to a request by redirecting the web browser to a different URL, many servers additionally allow users to omit certain parts of the URL, such as the "www." part, or the trailing slash if the resource in question is a directory. (Note: Omitting 'www.' is a feature of the DNS redirection, which may be performed at a top-level server and not on the HTTP server - but this distinction is transparent to an end-user). However, these omissions technically make it a different URL, so the web browser cannot make these adjustments, and has to rely on the server to respond with a redirect. It is possible, but due to tradition rare, for a web server to serve two different pages for URLs that differ only in a trailing slash.

Note that in http://distrowatch.com/dwres.php?resource=major the hierarchical order of the five elements is com (generic top-level domain) - distrowatch (second-level domain) - en (sub domain) -

dwres.php ; i.e. before the first slash from right to left, then the rest from left to right.
For a more extensive discussion of HTTP URLs and their use, see above.

The big picture

The term URL is also used outside the context of the World Wide Web. Database servers specify URLs as a parameter to make connections to it. Similarly any Client-Server application following a particular protocol may specify a URL format as part of its communication process.

Example of a database URL **:**

jdbc:datadirect:oracle://myserver:1521;sid=testdb

If a webpage is uniquely and more or less permanently defined by a URL it can be linked to (see also permalink, deep linking). This is not always the case, e.g. a menu option may change the contents of a frame within the page, without this new combination having its own URL. A webpage may also depend on temporarily stored information. If the webpage or frame has its own URL, this is not always obvious for someone who wants to link to it: the URL of a frame is not shown in the address bar of the browser, and a page without address bar may have been produced. The URL may be derivable from the source code and/or "properties" of various components of the page.

Apart from the purpose of linking to a page or page component, one may want to know the URL to show the component alone, and/or to lift restrictions such as a browser window without toolbars, and/or of a small non-adjustable size.

Web servers also have the ability to redirect URLs if the destination has changed, allowing sites to change their structure without affecting existing links. This process is known as URL redirection.

CHAPTER-5

PROXY SERVER

A proxy server is a computer that offers a computer network service to allow clients to make indirect network connections to other network services. A client connects to the proxy server, then requests a connection, file, or other resource available on a different server. The proxy provides the resource either by connecting to the specified server or by serving it from a cache. In some cases, the proxy may alter the client's request or the server's response for various purposes.
A proxy server can also serve as a firewall.

Web proxies

A common proxy application is a caching Web proxy. This provides a nearby cache of Web pages and files available on remote Web servers, allowing local network clients to access them more quickly or reliably.

When it receives a request for a Web resource (specified by a URL), a caching proxy looks for the resulting URL in its local cache. If found, it returns the document immediately. Otherwise it fetches it from the remote server, returns it to the requester and saves a copy in the cache. The cache usually uses an expiry algorithm to remove documents from the cache, according to their age, size, and access history. Two simple cache algorithms are Least Recently Used (LRU) and Least Frequently Used (LFU). LRU removes the documents that have been left the longest, while LFU removes the least popular documents.

Web proxies can also filter the content of Web pages served. Some censorware applications — which attempt to block offensive Web content — are implemented as Web proxies. Other web proxies reformat web pages for a specific purpose or audience; for example, Skweezer reformats web pages for cell phones and PDAs. Network operators can also deploy proxies to intercept computer viruses and other hostile content served from remote Web pages.

A special case of web proxies are "CGI proxies." These are web sites which allow a user to access a site through them. They generally use PHP or CGI to implement the proxying functionality. CGI proxies are frequently used to gain access to web sites blocked by corporate or school proxies. Since they also hide the user's own IP address from the web sites they access through the proxy, they are sometimes also used to gain a degree of anonymity.

Proxy Transparency

Many organizations — including corporations, schools, and families — use proxy servers to enforce network use policies (see censorware) or provide security and caching services. Usually, the web proxy is not transparent to the client application: it must be configured to use the proxy, manually or with a configuration script. Thus, the user can evade the proxy by simply resetting the client configuration, except in the case where the proxy is used instead of a NAT router to share an internet connection or a LAN. Such proxies may be difficult to configure for applications requiring a large port range out going and may only be able to route inward to a single server for a given UDP or TCP port (see Wingate 2.x versions). However such proxies may have more extensive logging or more customizable security than a simple NAT router box.

A transparent proxy or transproxy (also known as a forced proxy) combines a proxy server with NAT so that connections are routed into the proxy without client-side configuration. However, RFC 3040 defines this type as intercepting proxy.

Both NAT and transproxies are somewhat controversial in the Internet technical community, since both violate the end-to-end principle upon which TCP/IP was designed.

The term proxy is also used in a different sense in the Session Initiation Protocol (SIP) used in many modern voices over IP systems. A SIP Proxy, unlike a Web proxy, does not handle the content of client data.

Open proxies, abuse, and detection

An open proxy is a proxy server which will accept client connections from any IP address and make connections to any Internet resource. Abuse of open proxies is currently implicated in a significant portion of e-mail spam delivery. Spammers frequently install open proxies on unwitting end users' Microsoft Windows computers by means of computer viruses designed for this purpose. Internet Relay Chat (IRC) abusers also frequently use open proxies to cloak their identities.

Because proxies could be implicated in abuse, system administrators have developed a number of ways to refuse service to open proxies. IRC networks such as the Blitzed network automatically test client systems for known types of open proxy [1]. Likewise, an email server may be configured to automatically test e-mail senders for open proxies, using software such as Michael Tokarev's proxycheck [2].

Groups of IRC and electronic mail operators run DNSBLs publishing lists of the IP addresses of known open proxies, such as AHBL, CBL [3], NJABL [4], and SORBS.

The ethics of automatically testing clients for open proxies are controversial. Some experts, such as Vernon Schryver, consider such testing to be equivalent to an attacker portscanning the client host. [5] Others consider the client to have solicited the scan by connecting to a server whose terms of service include testing.

Reverse proxies

A reverse proxy is a proxy server that is installed in the neighborhood of one or more webservers. All traffic coming from the Internet and with a destination of one of the webservers is going through the proxy server. There are several reasons for installing reverse proxy servers:
Security: the proxy server is an additional layer of defense and therefore protects the webservers further up the chain
Encryption / SSL acceleration: when secure websites are created, the SSL encryption is often not done by the webserver itself, but by a reverse proxy that is equipped with SSL acceleration hardware. See Secure Sockets Layer.
Load distribution: the reverse proxy can distribute the load to several webservers, each webserver serving its own application area. In such a case, the reverse proxy may need to rewrite the URLs in each webpage (translation from externally known URLs to the internal locations)
Serve/cache static content: A reverse proxy can offload the webservers by caching static content like pictures and other static graphical content (See Squid cache)

Compression: the proxy server can optimize and compress the content to speed up the load time.

Split proxies

A split proxy is essentially a pair of proxies installed across two computers. Since they are effectively two parts of the same program, they can communicate with each other in a more efficient way than they can communicate with a more standard resource or tool such as a website or browser. This is ideal for compressing data over a slow link, such as a wireless or mobile data service and also for reducing the issues regarding high latency links (such as satellite internet) where estabilishing a TCP connection is time consuming. Taking the example of web browsing, the user's browser is pointed to a local proxy which then communicates with its other half at some remote location. This remote server fetches the requisite data, repackages it and sends it back to the user's local proxy, which unpacks the data and presents it to the browser in the standard fashion .

Anonymous proxy risks

In using a proxy server (for example, anonymizing HTTP proxy), all data sent to the service being used (for example, HTTP server in a website) must pass through the proxy server before being sent to the service, mostly in unencrypted form. It is therefore possible, and has been demonstrated (see, for example, Sugarcane) for a malicious proxy server to record everything sent to the proxy: including unencrypted logins and passwords.

By chaining proxies which do not reveal data about the original requestor, it is possible to obfuscate activities from the eyes of the user's destination. However, more traces will be left on the intermediate hops, which could be used or offered up to trace the user's activities. If the policies and administrators of these other proxies are unknown, the user may fall victim to a false sense of security just because those details are out of sight and mind.

The bottom line of this is to be wary when using proxy servers, and only use proxy servers of known integrity (e.g., the owner is known and trusted, has a clear privacy policy, etc.), and never use proxy servers of unknown integrity. If there is no choice but to use unknown proxy servers, do not pass any private information (unless it is properly encrypted) through the proxy.

CHAPTER-6

HYPERTEXT TRANSFER PROTOCOL

Hypertext Transfer Protocol (HTTP) is the method used to transfer or convey information on the World Wide Web. It is a patented open internet protocol whose original purpose was to provide a way to publish and receive HTML pages.

Development of HTTP was coordinated by the World Wide Web Consortium and working groups of the Internet Engineering Task Force, culminating in the publication of a series of RFCs, most notably RFC 2616, which defines HTTP/1.1, the version of HTTP in common use today.

HTTP is a request/response protocol between clients and servers. The originating client, such as a web browser, spider, or other end-user tool, is referred to as the user agent. The destination server, which stores or creates resources such as HTML files and images, is called the origin server. In between the user agent and origin server may be several intermediaries, such as proxies, gateways, and tunnels.

An HTTP client initiates a request by establishing a Transmission Control Protocol (TCP) connection to a particular port on a remote host (port 80 by default; see a list of well-known ports). An HTTP server listening on that port waits for the client to send a Request Message.

Upon receiving the request, the server sends back a status line, such as "HTTP/1.1 200 OK", and a message of its own, the body of which is perhaps the requested file, an error message, or some other information.

Resources to be accessed by HTTP are identified using Uniform Resource Identifiers (URIs) (or, more specifically, URLs) using the http: or https URI schemes.

<u>Request Message</u>

The request message consists of the following:

Request line, such as GET /images/logo.gif HTTP/1.1, which requests the file logo.gif from the /images directory

Headers, such as Accept-Language: en

An empty line

An optional message body

The request line and headers must all end with CRLF (i.e. a carriage return followed by a line feed). The empty line must consist of only CRLF and no other whitespace.

Some headers are optional, while others (such as Host) are required by the HTTP/1.1 protocol.

Request methods

HTTP defines eight methods indicating the desired action to be performed on the identified resource.

GET

Requests a representation of the specified resource. By far the most common method used on the Web today.

HEAD

Asks for the response identical to the one that would correspond to a GET request, but without the response body. This is useful for retrieving meta-information written in response headers, without having to transport the entire content.

POST

Submits user data (e.g. from a HTML form) to the identified resource. The data is included in the body of the request.

PUT

Uploads a representation of the specified resource.

DELETE

Deletes the specified resource (rarely implemented).

TRACE

Echoes back the received request, so that a client can see what intermediate servers are adding or changing in the request.

OPTIONS

Returns the HTTP methods that the server supports. This can be used to check the functionality of a web server.

CONNECT

For use with a proxy that can change to being an SSL tunnel.

Methods GET and HEAD are defined as safe, i.e. intended only for information retrieval. Unsafe methods (such as POST, PUT and DELETE) should be displayed to the user in a special way (e.g. as buttons rather than links), making the user aware of possible side effect of their actions (e.g. financial transaction).

Methods GET, HEAD, PUT and DELETE are defined to be idempotent, meaning that multiple identical requests should have the same effect as a single request. Also, the methods OPTIONS and TRACE should not have side effects, and so are inherently idempotent.

Despite the specified idempotence of GET requests, in practice, GET requests are often used to pass HTML form values or other data to an HTTP server. These requests can cause changes on the server, through CGI execution, which may result in different effects for successive identical requests. For example, an HTML page may use a link to cause the deletion of a database record; merely GET-ing a particular URL on a server will cause the CGI application on the server to delete a record, thus causing a change of the server's state and possibly making identical following requests to this URL to fail, on account of the database record already being deleted. This behavior is technically discouraged (non-idempotent actions should ideally be initiated by a POST request) but is very common on the modern World Wide Web. Such behavior can cause problems because various schemes for caching web pages, such as search engines, which by design GET pages before a user initiates a request, can cause unintentional changes on a server.

HTTP servers are supposed to implement at least GET and HEAD methods and, whenever possible, also OPTIONS method.

HTTP versions

HTTP differs from other TCP-based protocols such as FTP, because HTTP has different protocol versions:

0.9

Deprecated. Was never widely used. Only supports one command, GET. Does not support headers. Since this version does not support POST the client can't pass much information to the server.

HTTP/1.0

Still in wide use, especially by proxy servers. Allows persistent connections (alias keep-alive connections, more than one request-response per TCP/IP connection) when explicitly negotiated; however, this only works well when not using proxy servers.

HTTP/1.1

Current version; persistent connections enabled by default and works well with proxies. Also supports request pipelining, allowing multiple requests to be sent at the same time, allowing the server to prepare for the workload and potentially transfer the requested resources more quickly to the client.

Status codes

In HTTP/1.0 and since, the first line of the HTTP response is called the status line and includes a numeric status code (such as "200") and a textual reason phrase (such as "OK"). The way the user agent handles the response primarily depends on the code and secondarily on the response headers. Custom status codes can be used since if the user agent encounters a code it does not recognize, it can use the first digit of the code to determine the general class of the response. [1]

Also, the standard reason phrases are only recommendations and can be replaced with "local equivalents" at web developer's discretion. If the status code indicated a problem, the user agent might display the reason phrase to the user to provide further information about the nature of the problem. The standard also allows the user agent to attempt to interpret the reason phrase, though this might be unwise since the standard explicitly specifies that status codes are machine-readable and reason phrases are human-readable.

In practice, the reason phrase is unlikely to reach the user and is never interpreted. All modern web browsers rely on the status code to determine the handling and the response body to inform the user. However, the reason phrase may be logged and thus custom reason phrases might help in debugging.

See list of HTTP status codes for a list of all widely known status codes and associated standard reason phrases.

HTTP connection persistence

In HTTP/0.9 and HTTP/1.0, a client sends a request to the server, the server sends a response back to the client. After this, the connection is closed. HTTP/1.1, however, supports persistent connections. This enables the client to send a request and get a response, and then send additional requests and get additional responses. The TCP connection is not released for the multiple additional requests, so the relative overhead due to TCP is much less per request. The use of persistent connection is often called keep alive. It is also possible to send more than one (usually between two and five) request before getting responses from previous requests. This is called pipelining.

There is a HTTP/1.0 extension for connection persistence, but its utility is limited due to HTTP/1.0's lack of unambiguous rules for delimiting messages. This extension uses a header called Keep-Alive, while the HTTP/1.1 connection persistence uses the Connection header. Therefore a HTTP/1.1 may choose to support either just HTTP/1.1 connection persistence, or both HTTP/1.0 and HTTP/1.1 connection persistence. Some HTTP/1.1 clients and servers do not implement connection persistence or have it disabled in their configuration

HTTP connection closing

Both HTTP servers and clients are allowed to close TCP/IP connections at any time (i.e. depending on their settings, their load, etc.). This feature makes HTTP ideal for the World Wide Web, where pages regularly link to many other pages on the same server or to external servers.

Closing an HTTP/1.1 connection can be a much longer operation (from 200 milliseconds up to several seconds) than closing an HTTP/1.0 connection, because the first usually needs a linger close while the second can be immediately closed as soon as the entire first request has been read and the full response has been sent.

HTTP session state

https: is a URI scheme syntactically identical to the http: scheme used for normal HTTP connections, but which signals the browser to use an added encryption layer of SSL/TLS to protect the traffic. SSL is especially suited for HTTP since it can provide some protection even if only one side to the communication is authenticated. In the case of HTTP transactions over the Internet, typically only the server side is authenticated.

Example

Below is a sample conversation between an HTTP client and an HTTP server running on www.example.com, port 80.

Client request (followed by a blank line, so that request ends with a double newline, each in the form of a carriage return followed by a line feed):

```
GET /index.html HTTP/1.1
Host: www.example.com
```

The "Host" header distinguishes between various DNS names sharing a single IP address, allowing name-based virtual hosting. While optional in HTTP/1.0, it is mandatory in HTTP/1.1.

Server response (followed by a blank line and text of the requested page):

```
HTTP/1.1 200 OK
Date: Mon, 23 May 2005 22:38:34 GMT
Server: Apache/1.3.27 (Unix)  (Red-Hat/Linux)
Last-Modified: Wed, 08 Jan 2003 23:11:55 GMT
Etag: "3f80f-1b6-3e1cb03b"
```

```
Accept-Ranges: bytes
Content-Length: 438
Connection: close
Content-Type: text/html; charset=UTF-8
```

List of HTTP status codes

The following is a list of HTTP response status codes and standard associated phrases, intended to give a short textual description of the status. These status codes are specified by RFC 2616, along with additional, unstandardized status codes sometimes used on Web.

The first digit of the status code specifies one of five classes of response.

1xx Informational

Request received, continuing process.

100: Continue

101: Switching Protocols

2xx Success

The action was successfully received, understood, and accepted.

200: OK

201: Created

202: Accepted

203: Non-Authoritative Information

204: No Content

205: Reset Content

206: Partial Content

3xx Redirection

Further action must be taken in order to complete the request.

300: Multiple Choices

301: Moved Permanently

302: Moved Temporarily (HTTP/1.0)

302: Found (HTTP/1.1)

303: See Other (HTTP/1.1)

304: Not Modified

305: Use Proxy

Many HTTP clients (such as Mozilla and Internet Explorer) don't correctly handle responses with this status code.

306: (no longer used, but reserved)

307: Temporary Redirect

4xx Client Error

The request contains bad syntax or cannot be fulfilled.

400: Bad Request

401: Unauthorized

See basic authentication scheme and digest access authentication.

402: Payment Required

403: Forbidden

404: Not Found

405: Method Not Allowed

406: Not Acceptable

407: Proxy Authentication Required

408: Request Timeout

409: Conflict

410: Gone

411: Length Required

412: Precondition Failed

413: Request Entity Too Large

414: Request-URI Too Long

415: Unsupported Media Type

416: Requested Range Not Satisfiable

417: Expectation Failed

5xx Server Error

The server failed to fulfill an apparently valid request.

500: Internal Server Error

501: Not Implemented

502: Bad Gateway

503: Service Unavailable

504: Gateway Timeout

505: HTTP Version Not Supported

509: Bandwidth Limit Exceeded

This status code, while used by many servers, is not an official HTTP status code.

CHAPTER-7

BASIC AUTHENTICATION SCHEME

Here is a typical transaction between an HTTP client and an HTTP server running on the local machine (localhost). It is comprised of the following steps.

The client asks for a page that requires authentication but does not provide a user name and password. Typically this is because the user simply entered the address or followed a link to the page.

The server responds with the 401 response code and provides the authentication realm.

At this point, the client will present the authentication realm (typically a description of the computer or system being accessed) to the user and prompt for a user name and password. The user may decide to cancel at this point.

Once a user name and password have been supplied, the client re-sends the same request but includes the authentication header.

In this example, the server accepts the authentication and the page is returned. If the user name is invalid or the password incorrect, the server might return the 401 response code and the client would prompt the user again.

Note: A client may pre-emptively send the authentication header in its first request, with no user interaction required.

Client request (no authentication):

```
GET /private/index.html HTTP/1.0
Host: localhost
```

Server response:

```
HTTP/1.0 401 Unauthorised
Server: SokEvo/1.0
Date: Sat, 27 Nov 2004 10:18:15 GMT
WWW-Authenticate: Basic realm="SokEvo"
Content-Type: text/html
Content-Length: 311
```

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
 "http://www.w3.org/TR/1999/REC-html401-19991224/loose.dtd">
<HTML>
 <HEAD>
  <TITLE>Error</TITLE>
  <META HTTP-EQUIV="Content-Type" CONTENT="text/html; charset=ISO-8859-1">
 </HEAD>
 <BODY><H1>401 Unauthorised.</H1></BODY>
</HTML>
```

Client request (user name "Aladdin", password "open sesame"):

```
GET /private/index.html HTTP/1.0
Host: localhost
Authorization: Basic QWxhZGRpbjpvcGVuIHNlc2FtZQ==
```

Server response:

```
HTTP/1.0 200 OK
Server: SokEvo/1.0
Date: Sat, 27 Nov 2004 10:19:07 GMT
Content-Type: text/html
Content-Length: 10476
```

CHAPTER-8

HTTPS

https is a URI scheme which is syntactically identical to the http: scheme normally used for accessing resources using HTTP. Using an https: URL indicates that HTTP is to be used, but with a different default port and an additional encryption/authentication layer between HTTP and TCP. This system was invented by Netscape Communications Corporation to provide authentication and encrypted communication and is widely used on the Web for security-sensitive communication, such as payment transactions.

How it works

Strictly speaking, https is not a separate protocol, but refers to the combination of a normal HTTP interaction over an encrypted secure socket layer (SSL) or transport layer security (TLS) transport mechanism. This ensures reasonable protection from eavesdroppers and man in the middle attacks.

The default TCP port of an https: URI is 443 (for unsecured HTTP, the default is 80).
To prepare a web-server for accepting https connections the administrator must create a public key certificate for the web-server. These certificates can be created for Linux based servers with tools such as OpenSSL's ssl-ca [1] or SuSE's gensslcert. This certificate must be signed by a certificate authority of one form or another, who certifies that the certificate holder is who they say they are. Web browsers are generally distributed with the signing certificates of major certificate authorities such as VeriSign, so that they can verify certificates signed by them.

Organizations may also run their own certificate authority, particularly if they are responsible for setting up browsers to access their own sites (for example, sites on a company intranet), as they can trivially add their own signing certificate to the defaults shipped with the browser.

Finally, for a single site, self-signed certificates can be the ideal solution. It is important to understand though, that unless the certificate can be verified in some way (for example, phoning the certificate owner to verify its checksum), there is a risk of a man in the middle attack.

The system can also be used for client authentication, in order to restrict access to a web-server to only authorized users. For this, typically the site administrator creates certificates for each user which are loaded into their browser, although certificates signed by any certificate authority the server trusts should work. These normally contain the name and e-mail of the authorized user, and are automatically checked by the server on each reconnect to verify the user's identity, potentially without ever entering a password.

<u>Caveats</u>

The level of protection depends on the correctness of the implementation by the web browser and the server software and the actual cryptographic algorithms supported.

A common misconception among credit card users on the Web is that https: fully protects their card number from thieves. In reality, an encrypted connection to the Web server only protects the credit card number in transit between the user's computer and the server itself. It doesn't guarantee that the server itself is secure, or even that it hasn't already been compromised by an attacker.

Attacks on the Web sites that store customer data are both easier and more common than attempts to intercept data in transit. Merchant sites are supposed to immediately forward incoming transactions to a financial gateway and retain only a transaction number, but they often save card numbers in a database. It is that server and database that is usually attacked and compromised by unauthorized users.

Because SSL operates below http and has no knowledge of the higher level protocol, SSL servers can only present one certificate for a particular IP/port combination. This means that in most cases it is not feasible to use name based virtual hosting with https.

CHAPTER-9

FILE TRANSFER PROTOCOL

FTP or file transfer protocol is a commonly used protocol for exchanging files over any network that supports the TCP/IP protocol (such as the Internet or an intranet). There are two computers involved in an FTP transfer: a server and a client. The FTP server, running FTP server software, listens on the network for connection requests from other computers. The client computer, running FTP client software, initiates a connection to the server. Once connected, the client can do a number of file manipulation operations such as uploading files to the server, download files from the server, rename or delete files on the server and so on. Any software company or individual programmer is able to create FTP server or client software because the protocol is an open standard. Virtually every computer platform supports the FTP protocol. This allows any computer connected to a TCP/IP based network to manipulate files on another computer on that network regardless of which operating systems are involved (if the computers permit FTP access). There are many existing FTP client and server programs, and many of these are free.

Overview

FTP is commonly run on two ports, 20 and 21, and runs exclusively over TCP. The FTP server listens on port 21 for incoming connection from FTP clients. A connection on this port forms the control stream, on which commands are passed to the FTP server. For the actual file transfer to take place, a different connection is required. Depending on the transfer mode, the client (active mode) or the server (passive mode) can listen for the incoming data connection. Before file transfer begins, the client and server also negotiate the port of the data connection. In case of active connections (where the server connects to the client to transfer data), the server binds on port 20 before connecting to the client. For passive connections, there is no such restriction.

While data is being transferred via the data stream, the control stream sits idle. This can cause problems with large data transfers through firewalls which time out sessions after lengthy periods of idleness.

While the file may well be successfully transferred, the control session can be disconnected by the firewall, causing an error to be generated.

Objectives of FTP

The objectives of FTP, as outlined by its RFC, are:
To promote sharing of files (computer programs and/or data).
To encourage indirect or implicit use of remote computers.
To shield a user from variations in file storage systems among different hosts.
To transfer data reliably and efficiently.

Criticisms of FTP

Passwords and file contents are sent in clear text, which can be intercepted by eavesdroppers. There are protocol enhancements that circumvent this.
Multiple TCP/IP connections are used, one for the control connection, and one for each download, upload, or directory listing. Firewall software needs additional logic to account for these connections.
It is hard to filter active mode FTP traffic on the client side by using a firewall, since the client must open an arbitrary port in order to receive the connection. This problem is largely resolved by using passive mode FTP.
It is possible to abuse the protocol's built-in proxy features to tell a server to send data to an arbitrary port of a third computer; see FXP.
FTP is an extremely high latency protocol due to the number of commands needed to initiate a transfer.
No integrity check on the receiver side. If transfer is interrupted the receiver has no way to know if the received file is complete or not. It is necessary to manage this externally for example with MD5 sums or cyclic redundancy checking.

Security problems

FTP is an inherently insecure method of transferring files because there is no way for the original FTP specification to transfer data in an encrypted fashion. What this means is that under most network configurations, user names, passwords, FTP commands and transferred files can be "sniffed" or viewed by someone else on the same network using a protocol analyzer (or "sniffer"). It should be noted that this is a problem common to many Internet protocols written prior to the creation of SSL such as

HTTP, SMTP and Telnet. The common solution to this problem is to use SFTP (SSH File Transfer Protocol) which is based on SSH, or FTPS (FTP over SSL), which adds SSL or TLS encryption to FTP.

FTP return codes

See also: List of all FTP server return codes.

FTP server return codes indicate their status by the digits within them. Brief explanations of various digits' meanings are given below:

1yz: Positive Preliminary reply. The action requested is being initiated but there will be another reply before it begins.

2yz: Positive Completion reply. The action requested has been completed. The client may now issue a new command.

3yz: Positive Intermediate reply. The command was succesful, but a further command is required before the server can act upon the request.

4yz: Transient Negative Completion reply. The command was not successful, but the client is free to try the command again as the failure is only temporary.
5yz: Permanent Negative Completion reply. The command was not successful and the client should not attempt to repeat it again.

x0z: The failure was due to a syntax error.

x1z: This response is a reply to a request for information.

x2z: This response is a reply relating to connection information.

x3z: This response is a reply relating to accounting and authorisation.

x4z: Unspecified.

x5z: This response is a reply relating to the file system.

Anonymous FTP

Many sites that run FTP servers enable so-called "anonymous ftp". Under this arrangement, users do not need an account on the server. The user name for anonymous access is typically 'anonymous' or 'ftp'. This account does not need a password. Although users are commonly asked to send their email addresses as their passwords for authentication, usually there is trivial or no verification, depending on the FTP server and its configuration. Internet Gopher has been suggested as an alternative to anonymous FTP.

Data format

While transferring data over the network, two modes can be used
ASCII mode
Binary mode

The two types differ in the way they send the data. When a file is sent using an ASCII-type transfer, the individual letters, numbers and characters are sent using their ASCII character codes. The receiving machine saves these in a text file in the appropriate format (for example, a Unix machine saves it in a Unix format, a Macintosh saves it in a Mac format). Hence if an ASCII transfer is used it can be assumed plain text is sent, which is stored by the receiving computer in its own format.

Sending a file in binary mode is different. The sending machine sends each file bit for bit and as such the recipient stores the bitstream as it receives it.

By default, most FTP clients use ASCII mode. Some clients try to determine the required transfer-mode by inspecting the file's name or contents.

FTP and web browsers

Most recent web browsers and file managers can connect to FTP servers, although they may lack the support for protocol extensions such as FTPS. This allows manipulation of remote files over FTP through an interface similar to that used for local files. This is done via an FTP URL, which takes the form ftp(s)://<ftpserveraddress> (e.g., ftp://ftp.gimp.org/). A password can optionally be given in the URL, e.g.: ftp(s)://<login>:<password>@<ftpserveraddress>:<port>. Most web-browsers require the use of passive mode FTP, which not all FTP servers are capable of handling.

FTP over SSH

The practice of tunneling a normal FTP session over an SSH connection.

Since FTP (unusual for a TCP/IP protocol that is still in use) uses multiple TCP connections, it is particularly difficult to tunnel over SSH. With many SSH clients, attempting to set up a tunnel for the control channel (the initial client-to-server connection on port 21) will only protect that channel; when data is transferred, the FTP software at either end will set up new TCP connections (data channels) which will bypass the SSH connection, and thus have no confidentiality, integrity protection, etc.

If the FTP client is configured to use passive mode and to connect to a SOCKS server interface that many SSH clients can present for tunneling, it is possible to run all the FTP channels over the SSH connection.

Otherwise, it is necessary for the SSH client software to have specific knowledge of the FTP protocol, and monitor and rewrite FTP control channel messages and autonomously open new forwarding for FTP data channels. Version 3 of the SSH Communications Security Corp. software is an example of software supporting this. [1]

FTP over SSH is sometimes referred to as secure FTP; this should not be confused with other methods of securing FTP, such as with SSL/TLS (FTPS). Other methods of transferring files using SSH which are not related to FTP include SFTP or SCP; in both of these, the entire conversation (credentials and data) is always protected by the SSH protocol.

CHAPTER-10

DOWNLOAD MANAGER DESCRIPTION

DOWNLOAD CLASS

The download class is the workhorse of the DM .Its primary purpose is to download a file and save the files contents to the disk. Each time a new download is added to the DM, a new download object is instantiated to handle the download.

The DM has the ability to download multiple files at once. To achieve this, its necessary for each of the simultaneous downloads to run independently. its also necessary for each individual download to manage its own state so that it can be reflected in the GUI. This is accomplished with the DOWNLOAD class.

THE DOWNLOAD VARIABLES

Download begins by declaring several static final variables that specify the various constants used by class. Next four instance variables are declared. The URL variable holds the internet URL for the file being downloaded; the size variable holds the size of the download file in bytes; the downloaded variable holds the number of bytes that have been downloaded thus far; and the status variable indicates the downloads current status.

THE DOWNLOAD CONSTRUCTOR

Downloads constructor is passed a reference to the URL to download in the form of a url object, which is assigned to the url instance variable. It then sets the remaining instance variables to their initial states and calls the download() method. Notice that the size is set to -1 to indicate there is no size yet.

THE DOWNLOAD METHOD()

The download method creates a new thread object, passing it a reference to the invoking download instance . As mentioned before, its necessary for each download to run independently. In order for the download class to act alone, it must execute in its own thread.

THE RUN METHOD

When the run() method executes, the actual downloading gets under way. Because of its size and importance, we will examine it closely.

First, run() sets up variables for the network stream that the downloads contents will be read from and sets up the file that the down load's contents will be written to. Next a connection to the download URL will be opened by calling **url.openconnection().**since we know that the download manager supports only HTTP downloda , the connection is cast to the **HttpURLConnection** type. Casting the connection as an **HttpURLConnection allows** us to take the advantage of the http specific connection features such as **responsecode()** method. Note that calling **url.openconnection()** does not actually create a connection to the URL server. It simply creates a new url connection instance associated with the URL that later will be used to connect to the server.

Setting request properties allows extra request information to be sent to the server the download will be coming from. In this case, the "Range" property is set. This is critically important, as the "range" property specifies the range of bytes that is being requested for download from the server. Normally all of the files bytes are downloaded at once. However if a download has been interrupted or paused; only the down load's remaining bytes should be retrieved. Setting the "range" property is the foundation for the download managers operation.

The "range" property is defined in this form

Start-byte-end-byte.

For example, "0-12345". However, the end byte of the range is optional. If the end byte is absent, the range ends at the end of the file. The run() method never specifies the end byte because downloads must run until the entire range is downloaded, unless paused or interrupted.

The **connection.connect()** method is called to make the actual connection to the download server. Next , the response code returned by the server is checked. The HTTP protocol has a list of response codes that indicate a server's response to a request . HTTP response codes are organized into numeric ranges of 100,and the 200 range indicates success. The server's response code is validated for being in the 200 range by calling **connection.getResponsecode()** and dividing by 100.If the value of this division is 2, then the connection was successful .
Next, **run()** gets the content length by calling **connection.getContentLength().** The content length represents the number of bytes in the requested file. If the content length is less than 1, the error() method is called .The error() method updates the down load's status to **ERROR** ,and then calls **statechanged().**

As we can see, instead of assigning the content length to the **size** variable unconditionally, it only gets assigned if it hasn't already been given a value. The reason for this is because the content length reflects how many bytes the server will be sending .if anything other than a 0-based start range is specified; the content length will only represent a portion of the files size. The SIZE variable has to be set to the complete size of the down load's file.

The next few lines of code shown here to create a new **randomAccessFile** using the filename portion of the down load's URL that is retrieved with a call to the **getFileName() method.**

The **RandomAccessFile** is opened in ”rw” mode, which specifies that the file can be written to and read from. Once, run() seeks to the end of the file by calling the **file.seek()** method, passing in the **downloaded** variable. This tells the file to position itself at the number of bytes that have been downloaded-in other words, at the end. its necessary to position the file at the end in case a download has been resumed, the newly downloaded bytes are appended to the file they don't overwrite any previously downloaded bytes. After preparing the output file, a network stream handle to the open server connection is obtained by calling **connection.getInputStream().** The loop is set up to run until the downloads status variable changes from DOWNLOADING. Inside the loop, a byte buffer array is created to hold the bytes that will be downloaded. The buffer is sized according to how much of the download is left to complete. If there is more left to download than the **MAX_BUFFER_SIZE, the MAX_BUFFER_SIZE** is used to size the buffer. Otherwise, the buffer is sized exactly at the number of bytes left to download. Once the buffer is sized appropriately, the downloading takes place with a

stream.read() call. This call reads bytes from the server and places them into the buffer, returning the count of how many bytes were actually read. If the number of bytes read equals -1, then downloading has completed and the loop is exited. Otherwise, downloading is not finished and the bytes that have been read are written to disk with a call to **file.write().**Then the downloaded variable is updated ,reflecting the number of bytes downloaded thus far. Finally, inside the loop, the **stateChanged()** method is invoked.

If the downloads status is still **downloading, this** means that the loop exited because downloading has been completed. Otherwise, the loop was exited because the down load's status changed to something other than **DOWNLOADING.**

The **run()** method wraps up with the **catch** and **finally** blocs shown
If an exception is thrown during the download process, the catch block captures the exception and calls the error() method. The finally block ensures that if the file and stream connections have been opened, they get closed whether an exception has been thrown or not.

The StateChanged()Method

In order for the DM to display up-to-date information on each of the downloads its managing, it has to know each time a downloads information changes. To handle this, the Observer software design pattern is used. The observer pattern analogous to an announcement's mailing list where several people register to receive announcements.

The download class employs the observer pattern by extending java built in observable utility class. Extending the observable class allows classes that implement java's observer interface to register themselves with the **download** class to receive change notifications.

ACTION AND ACCESSOR METHODS

The **download** class has numerous action and accessor methods for controlling a download and getting data from it. Each of the **pause(), resume(),** and **cancel()** action methods simply does as its name implies: pauses, resumes, or cancels the download, respectively. Similarly, the **error()** method marks the download as having an error. The **getUrl(), getSize() ,getProgress,** and **getStatus()** accessor methods each return their current respective values.

THE PROGRESS RENDERER CLASS

The progressrenderer class is a small utility class that is used to render the current progress of a download listed in the GUI 's "Downloads" jtable instance. Normally, a jtable instance renders each cell data as text. However, often it's particularly useful to render a cell's data as something other than text. In the download manager case, we want to render each of the table's progress column cells s progress bars. The progressrenderer class shown makes that possible.

The progressrenderer class takes the advantage of the fact that swings jtable class has a rendering system that can accept "plug-ins" for rendering table cells. To plug in to this rendering system, first, the progressrenderer class has to implement swings tablecellrenderer interface. Second, a progressrenderer instance has to be registered with a jtable instance.doing so instructs the jtable instance as to which cells should be rendered with "plug-in".

The gettablecellrenderercomponent() method wraps up by returning a reference to its class. This works because the progressrenderer class is a subclass of JprogressBar, which is a descendent of AWT component class.

THE DOWNLOADS TABLE MODEL CLASS

The downloadstablemodel class houses the DM list of downloads and is the backing data source for the GUI's "downloads"jtable instance.

The downloadstablemodel class is shown here. Notice that it extends abstactablemodel and implements the observer interface.

The addDownload() Method

The addDownload() method ,shown adds a new download object to list of managed downloads and a row to the table

The cleardownload() method

The cleardownload() method, removes a download from the list of managed downloads.

The getcolumnclass() method

The getcolumnclass() method ,shown returns the class type for the data displayed in the specified column

The getvalueat() method

The getvalueat method, shown is called to get the current value that should be displayed for each of the table's cells

The update() method

The update() method is shown here. It fulfills the observer interface contract allowing the downloadstablemodel class to receive notifications from download objects when they change

THE DOWNLOADMANAGER CLASS

Now that the foundation has been laid by explaining each of the DM helper classes, we can look closely at the DM's class. The DM class is responsible for creating and running the DM's GUI. This class has a main() method declared so on execution it will be invoked first. The main() method instantiates a new download manager class instance and then calls its show() method, which causes it to be displayed.

The downloadmanager variables

Download manager starts off by declaring several instance variables, most of which hold references to the GUI controls finally, clearing instance variable is Boolean flag that tracks whether or not a download is currently being cleared from the downloads table.

The verify url() method

The verify url() method is called by the action add method each time a download is added to the download manager. The verify url() method first verifies that the URL entered is an HTTP url since only HTTP is supported .next, the url being verified is used to construct a new url class instance .if the URL is that a file is actually specified in the url.

The table selection changed method()

This method starts by seeing if there is already a row currently selected by checking if the selected download variable is null. If the select download variable is not null, download manager removes itself as an observer of the download so that it is no longer receives change notifications. Next the clearing flag is checked. If the table is not empty and the clearing flag is false, then first the select download variable is updated with the download corresponding to his row selected.

The updatebuttons() method

The updatebuttons() method updates the state of all the buttons on the button panel based on the state of the selected download.

Handling action events

Each of the DM GUI controls registers an action listener that invokes its respective action method. For example if any button is clicked, an action event is generated and each of the button's registered action listeners is notified.

COMPILING AND RUNNING THE DOWNLOAD MANAGER

We have used JCreator3.5 as our IDE for developing this Download Manager. This IDE is compatible with J2SDK1.5 which is the latest version of java. We used JCreator because it is very light to use and takes very fewer System resources.

We used Exe4J3.0 for making a windows executable file.

ENHANCING DOWNLOAD MANAGER

The Juit Download Manager 1.0 is fully functional, with the ability to pause and resume downloads as well as download multiple files at once, however, there are several enhancements we would like to make in our future releases like proxy server support, FTP, HTTPS support, and drag and drop support. A particularly appealing enhancement is a scheduling feature that lets you schedule a download at a specific time, perhaps in the middle of the night when system resources are plentiful.

CONCLUSION

We hereby conclude that download manager illustrated is dedicated to all the students, faculty, research associates and enthusiasts of our beloved institute JUIT. This download manager is the free software indigenously prepared by us especially to cater the needs of students of JUIT. Hence we take pride in dedicating this download manager to JUIT. The platform used in this case is JAVA which is most versatile, adaptable and also portable way of implementing software's in user-friendly way.

BIBILOGRAPHY

Referred Books:

S/N	*Title*	*Authors*
1	Java How to Program	H.M.Deitel & P.J.Deitel
2	Developing Intranet Applications Using Java	Jerry Ablan
3	Java RMI	William Grosso
4	Java Network Programming	Elliotte Rusty Harold
5	CORBA Networking With Java	George M. Doss
6	Thinking In Java	Bruce Eckel

Research Paper

Java Language Specification 3rd Edition by James Gosling

Web Pages

1) www.java.sun.com
2) www.java.net.
3) Orkut Communities
4) Yahoo Groups